Exchanging Data and Ontological Definitions in Multi-Agent-Contexts Systems

Stefania Costantini and Giovanni De Gasperis

Department of Computer Science and Engineering and Mathematics
University of L'Aquila, Italy, email: {stefania.costantini,giovanni.degasperis}@univaq.it

Abstract. In recent work we have extended DACMAS, which is a formalization of ontology-based and commitment-based multi-agent systems. Our extension allows a system to include not only agents but also external contexts. The objective is that of modeling real-world situations where agents not only interact among themselves, but also consult external heterogeneous data- and knowledge-bases to extract useful information. In this paper we further enhance the approach, so that a querying agent is enabled to specify in ontological terms which data it intends to extract from a context, or vice versa needs to be made aware of ontological assumptions the received results are based upon.

1 Introduction

In Multi-agent systems (MAS), ontologies constitute a key point in practical applications for agents defining and maintaining their own knowledge base, and for inter-agent communication. It is generally acknowledged that the ontology used for communication must be agreed upon and understood by the agents composing a MAS, and in fact virtually all Agent Communication Languages (ACLs) allow for the specification of the ontology for communication. In interesting recent work [1], Calvanese De Giacomo and Montali take the explicit stance that data maintained by the agents impact on the dynamics of a MAS, and to the evolution of inter-agent communication. They thus formulate an approach, called DACMAS (Data-Aware Commitment-based MAS) based upon the DRL-Lite Description Logic [2]: in DACMAS, a set of agents is augmented with an *institutional* agent which owns a "global" TBox, representing the basic concepts, relations and constraints which are proper of the domain in which the agents operate. Each agent in a DACMAS is then equipped with a local ABox which is consistent with the global TBox. As agents composing a DACMAS are autonomous entities, mutual consistency of the ABoxes is not required. Agents are also equipped with a set of reactive and proactive rules, where proactive rules model communications, and reactive rules model internal updates.

Communication in DACMASs is based upon *commitments* (cf. [3, 4] and the references therein), which constitute a relatively recent but very well-established approach to agent communication. In DACMASs, the institutional agent is aware of all agents participating in the system and of their interaction, and implements the "Commitment machine", which manages commitments lifecycle. More precisely, agents in a DACMAS communicate by interchanging *events*, which may then lead to commitment formation. For instance, a potential buyer sends an event to a seller in order to register as customer. When (via an internal update) the seller does so, it sends to the buyer events representing offers. Only later, in case, e.g., the customer decides to buy, commitments are (conditionally) created by the institutional agent about, e.g., payment and delivery.

In [5] we have extended the DACMAS approach to include the possibility, for agents, to gather information from external heterogeneous data sources. In real-world applications in fact, decisions (such as, e.g., whether to accept a customer or whether to issue an order to a certain seller, whether to enroll a candidate student, whether to concede a loan and at which conditions, etc.) are taken, via suitable reasoning strategies, after consulting a number of information sources that can be not only internal, but also external to the agent system, and that cannot be in general standardized. We considered as a basis the DACMAS approach because it introduces the element of logical ontologies within simple rule-based agents, of which they provide a specification only concerning the few kinds of reactive and proactive rules mentioned above. Therefore, in our opinion the approach can be seen as a useful general framework that can be specialized to many rule-based languages, in which to program the reasoning and deliberative components of agents, or further generalized communication modalities. Such languages can be, e.g., AgentSpeak (cf. [6] and the references therein), GOAL (cf. [7] and the references therein) 3APL (cf. [8] and the references therein), METATEM(cf. [9] and the references therein), KGP (cf [10] and the references therein), DALI (cf. [11, 12]), or other agent-oriented languages [13]. The DACMAS approach is interesting also for its potential for affordable verification, again due to the rule-based nature of DACMAS agents.

In the Artificial Intelligence and Knowledge Representation field, the managed Multi-Context Systems (mMCS) approach has been proposed to model information exchange among heterogeneous entities [14, 15]. In this proposal, there is a number of (information) contexts that are essentially kinds of heterogeneous reasoning entities, which include so-called *bridge rules* that enable interaction. MCSs assume real heterogeneity and partial information about information sources, wherein interesting comprehensive existing approaches such as, e.g., "Agents and Artifacts" (cf. [16] and the references therein) assume that external sources can be described, or even manipulated by agents of the MAS. This is often not the case, as external knowledge bases are "opaque" and what is known is limited to the kind of information that can be obtained.

The approach of DACMACSs (Data-Aware Commitment-based managed Multi-Agent-Context Systems) [5] integrates DACMASs and mMCS. This by augmenting the set of participating agents by a set of contexts, and by equipping agents with special communicative rules that are a variant of bridge rules. We have assumed that contexts, though heterogeneous in nature, accept relational-like queries, and in particular datalog queries (that correspond to the select-project-join fragment of SQL) with negation. The main features of the DACMACS proposal are in summary the following. (1) Agents can query (sets of) contexts, but contexts cannot query agents. (2) Agents are equipped, like contexts in MCSs, with bridge rules for knowledge interchange. Their application is however, differently from MCSs, not automatic but rather determined via special *trigger rules*, which allow a bridge rule to be invoked upon certain conditions and/or according to a certain timing. (3) The result of a bridge rule is interpreted as an agent-generated internal event, which is captured by reactive rules which may determine modifications to the agent's ABox according to suitable reasoning techniques for the analysis and the incorporation of the newly acquired knowledge.

In this paper we intend to further extend the approach so as to make it more realistic in terms of data management. In DACMACS we have assumed that contexts, though heterogeneous in nature, accept relational-like queries, and in particular datalog queries[1] (that correspond to the select-project-join fragment of SQL) with negation. Here, we first of all assume that all agents and contexts in a DACMACS are equipped with their local TBox. For agents, it can be considered to be an addition to the global TBox. We still consider contexts as "black-boxes", though we assume them to be able to access their associated TBox. In this way, contexts can be able to cope with the case where a querying agent is not able to specify in ontological terms the features of the data it intends to obtain (for instance, a University office may ask the tax office for the names of low-income students who should be given some practical and/or financial help, without however being aware of the precise regulations according to which a student can be considered to be "low-income"). A context will now be enabled to retrieve such ontological definitions in its own TBox. Also, the requesting agent can be made aware of ontological assumptions the results are based upon (in the example, the regulations for obtaining advantages). In extended DACMACS therefore, agents are more "demanding" w.r.t. external sources.

Extended DACMACS agents are then able to safely incorporate ontological definitions provided by external sources int their own TBox, so as to accordingly incorporate acquired data into their ABox.

The paper is organized as follows: in Sections 2 we provide the necessary background notions about mMCSs, DACMASs and DACMACSs. In Section 3 we present and illustrate, also by means of examples, the extended approach. We also discuss whether properties of mMCSs and DACMACSs extend to the new formalization. Finally, in Section 4 we conclude.

2 Background

2.1 mMCS

(Managed) Multi-Context systems [15, 18] aim at making it possible to build systems that need to access multiple possibly heterogeneous data sources, called "contexts", despite the variety of formalisms and languages these data sources can be based upon. The device for modeling the necessary information flow among contexts is constituted by "bridge rules", which are similar to datalog rules with negation (cf., e.g., [17]). Bridge rules allow for inter-context communication: in fact, each element in their "body" explicitly includes the indication of the context from which information is to be obtained.

Reporting from [15], a logic L is a triple $(KB_L; Cn_L; ACC_L)$, where KB_L is the set of admissible knowledge bases of L, which are sets of KB-elements ("formulas"); Cn_L is the set of acceptable sets of consequences (in [15] they are called "belief sets", but we prefer to abstract away from "mentalist" concepts), whose elements are data items or "facts"; $ACC_L : KB_L \rightarrow 2^{Cn_L}$ is a function which defines the semantics

[1] c.f., e.g., [17] for a survey about datalog and the references therein for more information.

of L by assigning each knowledge-base an "acceptable" set of consequences. A multi-context system (MCS) $M = (C_1, \ldots, C_n)$ is a heterogeneous collection of contexts $C_i = (L_i; kb_i; br_i)$ where L_i is a logic, $kb_i \in KB_{L_i}$ is a knowledge base and br_i is a set of bridge rules. Each such rule is of the following form, where the left-hand side s is called the *head*, also denoted as $hd(\rho)$, the right-hand side is called the *body*, also denoted as $body(\rho)$, and the comma stand for conjunction.

$$s \leftarrow (c_1 : p_1), \ldots, (c_j : p_j), not\,(c_{j+1} : p_{j+1}), \ldots, not\,(c_m : p_m).$$

For each bridge rule included in a context C_i, it is required that $kb_i \cup s$ belongs to KB_{Li} and, for every $k \leq m$, c_k is a context included in M, and each p_k belongs to some set in KB_{L_k}. The meaning is that s is added to the consequences of kb_i whenever each p_r, $r \leq j$, belongs to the consequences of context c_r, while instead each p_s, $j < s \leq m$, does not belong to the consequences of context c_s. If $M = (C_1, \ldots, C_n)$ is an MCS, a data state ("belief state" in the terminology of [15]) is a tuple $S = (S_1, \ldots, S_n)$ such that each S_i is an element of Cn_i. Desirable data states are those where each S_i is acceptable according to ACC_i. A bridge rule ρ is applicable in a data state iff for all $1 \leq i \leq j : p_i \in S_i$ and for all $j + 1 \leq k \leq m : p_k \notin S_k$. Let $app(S)$ be the set of bridge rules which are applicable in a data state S.

We will now introduce managed MCS (mMCS) though in a simplified form with respect to [15] (in particular, our formulation is in between those of [15] and [18]). While in standard MCSs the head s of a bridge rule is simply added to the "destination" context's data state kb, in managed MCS kb is subjected to an elaboration w.r.t. s according to a specific operator *op* and to its intended semantics: rather than simple addition, *op* can determine, e.g. deletion of other formulas upon addition of s, or any kind of elaboration and revision of kb w.r.t. s. Formula s itself can be elaborated by *op*, for instance with the aim of making it compatible with kb's format.

For a logic L, $F_L = \{s \in kb \mid kb \in KB_L\}$ is the set of formulas occurring in its knowledge bases. A *management base* is a set of operation names (briefly, operations) OP, defining elaborations that can be performed on formulas, e.g., addition of, revision with, etc. For a logic L and a management base OP, the set of operational statements that can be built from OP and F_L is $F_L^{OP} = \{o(s) \mid o \in OP, s \in F_L\}$. The semantics of such statements is given by a management function, which maps a set of operational statements and a knowledge base into a modified knowledge base. In particular, a management function over a logic L and a management base OP is a function

$$mng : 2^{F_L^{OP}} \times KB^L \rightarrow 2^{KB_L} \setminus \emptyset$$

Bridge rules for context C_i have the same format as in standard MCSs, except that the head now belongs to F_L^{OP}, and is then of the form $o(s)$.

Semantics of (simplified) mMCS are in terms of *equilibria*. A data state $S = (S_1, \ldots, S_n)$ is an equilibrium for an MCS $M = (C_1, \ldots, C_n)$ iff, for $1 \leq i \leq n$, there exists $kb'_i = mng_i(app(S), kbi)$ such that $S_i \in ACC_i(kb'_i)$. Thus, an equilibrium is a global data state composed of acceptable data sets, one for each context, considering however inter-context communication determined by bridge rules and the elaboration resulting from the operational statements and the management function.

Equilibria may not exist, or may contain inconsistent data sets (local inconsistency, w.r.t. *local consistency*). A management function is called *local consistency (lc-) preserving* iff, for every given management base, kb' is consistent. It can be proved that a mMCS where all management functions are lc-preserving is locally consistent. *Global consistency* would require the S_i's to be consistent with each other, but this stronger property is not required in this context.

Intuitively, a data state is an equilibrium whenever the application of a bridge rule according to the destination context's strategy for incorporating new knowledge produces a result which is compatible with the context's semantics for its data sets. Notice that bridge rules are intended to be applied whenever they are applicable.

2.2 DR-Lite in a Nutshell

We assume as known the basic notions about Description Logic and ontologies [2]. In particular we consider here DLR-Lite [2, 19], of which however we provide only the very basic elements, as far as they are needed in what follows. In general terms, in this logic (1) A TBox is a finite set of assertions specifying: concepts and relations; inclusion and disjunction among concepts/relations; key assertions for relations. (2) An ABox is a finite set of assertions concerning concept and relation membership. In essence, a TBox describes the structure of the data/knowledge, and the ABox specifies the actual data/knowledge instance. (3) In DLR-Lite, data can be queried via UCQs (Union of Conjunctive Queries) and ECQs (Existential Conjunctive Queries): the latter are FOL (First-Order Logic) queries involving negation, conjunction and the existential quantifier, whose atoms are UCQs.

In DLR-Lite, concepts C and relations R are built from *atomic concepts* N and *atomic relations* P (of arity ≥ 2) where: a concept is either an atomic concept or a unary relation or a conjunction between concepts; a relation is either an atomic relation or any projection of an n-ary relation.

Formally, the semantics of DL-Lite is defined in terms of first-order interpretations consisting of a nonempty interpretation domain and an interpretation function which maps concept conjunction over the intersection of their interpretations, and treats relations in the obvious way.

A DLR-Lite TBox is a finite set of assertions of the form:

$$\begin{array}{ll} E_1 \sqsubseteq E_2 & \text{(concept/relation inclusion)} \\ E_1 \sqsubseteq \neg E_2 & \text{(concept/relation disjointness)} \\ (key\ i_1, \ldots i_k : R) & \text{(key assertion)} \end{array}$$

where each of the i_js is between 1 and the arity n of R: in fact, the key assertion indicates which of the arguments of R compose its primary key. In the following, we use $\hat{a}$ to denote an ordered set of elements, typically for relation arguments when the arity is irrelevent, thus writing, e.g., $R(\hat{x})$.

To ensure decidability of inference and good computational properties, no relation can appear both in a key assertion and in the right hand side of a relation inclusion assertion.

A DLR-Lite ABox is a finite set of assertions of the form:

$N(a1)$ (concept membership assertion)
$P(a_1, \ldots, a_n)$ (relation membership assertion)

where P has arity n, and $a_1, \ldots, a_n$ denote constants. The semantics is provided by stating when an interpretation I satisfies assertions, which is done in the obvious way.

An Ontology $\mathcal{O}$ is a couple $\langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{T}$ is a TBox, encoding intensional knowledge, and $\mathcal{A}$ is an ABox, encoding extensional knowledge about individuals objects.

Query answers, as usual in ontologies, are formed by constants denoting individuals explicitly mentioned in the ABox. The *certain answer* to an UCQ q the set substitutions θ of the free variables of q with constants in $\mathcal{A}$ such that $q\theta$ evaluates to true in every model of $\mathcal{O}$. The *certain answer* to an ECQ defined by combining the certain answers of the composing UCQs through first-order constructs, and interpreting existential variables as ranging over the constants in $\mathcal{A}$.

DLR-Lite enjoys desirable computational properties w.r.t. query evaluation, which is tractable in both time and data complexity (please refer to the above references for discussion).

2.3 DACMASs and DACMACSs

The definition of a DACMACS (Data-Aware Commitment-based Multi-Agent-Context Systems) [5] extends that of DACMAS as the set of participating agents is augmented with a set of contexts, which are to be understood as data sources which can be consulted by agents. Both approaches are based upon DLR-Lite ontologies. In both DACMASs and DACMACSs a MAS is equipped with a global TBOX, while each agent owns its local ABox, that must be consistent with the global TBox. As agents composing a DACMAS are autonomous entities, mutual consistency of the ABoxes is not required. Bridge rules similar to those of MCSs can be defined also in DACMACS agents. Agents are moreover equipped with local management functions, which can perform any elaboration for acquiring data and for keeping each agent's local ABox consistent.

In the following, let a logic, a management base and management functions be as specified in Section 2.1. Formally, we have (where $\mathcal{T}, \mathcal{E}, \mathcal{X}, \mathcal{I}, \mathcal{C}$ and $\mathcal{B}$ are borrowed from the DACMAS definition):

Definition 1. *A DACMACS (Data-Aware Commitment-based managed Multi- Context MAS) is a tuple $\langle \mathcal{T}, \mathcal{E}, \mathcal{X}, \mathcal{Y}, \mathcal{I}, \mathcal{C}, \mathcal{B} \rangle$ where: (i) $\mathcal{T}$ is a global DLR-Lite TBox, which is common to all agents participating in the system; (ii) $\mathcal{E}$ is a set of predicates denoting events (where the predicate name is the event type, and the arity determines the content/payload of the event); (iii) $\mathcal{X}$ is a finite set of agent specifications; (iv) $\mathcal{Y}$ is a finite set of context names; (vv) $\mathcal{I}$ is a specification for the institutional agent; (vi) $\mathcal{C}$ is a contractual specification; (vii) and $\mathcal{B}$ is a Commitment Box (CBox).*

The global TBox lists, as in DACMASs, the names of all participating agents in connection to their specifications, but now it also lists the description of all available external contexts, for which (differently from agents) no specification is available within the system. An additional list of contexts locally known to an agent may be included in the agent's ABox. Each context has a name and a *role*, indicating to agents the kind of information that can be obtained from such context. For simplicity, we assume that roles are specified as constants, that context names include all the information needed for actually posing queries, and that all contexts accept datalog queries. The last assumption is without loss of generality, as most realistic query languages admit a well-defined correspondence with datalog.

An agent's specification includes a (possibly empty set of): *communicative rules*, which proactively determine events to be sent to other agents; *update rules*, which are internal reactive rules that update the local ABox upon sending/receiving an event to/from another agent. the other participants.

A communicative rule has the form

$$Q(r, \hat{x}) \textbf{ enables } EV(\hat{x}) \textbf{ to } r$$

where: Q is an ECQ query, or precisely an ECQ_l with *location argument* $@Ag$ to specify the agent Ag to which the query is directed (if omitted, then the the agent queries its own ABox); $\hat{x}$ is a set of tuples representing the results of the query; $EV(\hat{x})$ is an event supported by the system, i.e., predicate EV belongs to $\mathcal{E}$; r is a variable, denoting an agent's name. Whenever the rule is proactively applied, if the query evaluates to true (i.e., if the query succeeds) then $EV(\hat{x})$ and r are instantiated via one among the answers returned by the query, according to the agent's own choice. For instance, an agent may find the name r of the provider of a service (if several names are returned, only one is chosen) and then send to this provider a subscription request (instantiated with the necessary information $\hat{x}$) in order to be able to access the service.

Update rules are ECA-like rules[2] of the following form, where α is an action, the other elements are as before, and each rule is to be applied whenever an event is either sent or received, as specified in the rule itself:

$$\textbf{on } EV(\hat{x}) \textbf{ to } r \textbf{ if } Q(r, \hat{x}) \textbf{ then } \alpha(r, \hat{x}) \textbf{ (on-send/on-receive)}$$

Update rules may imply the insertion in the agent's ABox of new data items not previously present in the system, taken from a countably infinite domain Δ.

Each context may include bridge rules, of the form specified in section 2.1, where however the body refers to contexts only, i.e., contexts cannot query agents. In the DACMACS approach, also agents may be equipped with bridge rules, for extracting data from the contexts listed in the global TBox (while data exchange among agents occurs via explicit communication as defined in DACMASs). Bridge rules in agents are of the form:

$$A(\hat{x}) \textbf{ determinedby } E_1, \ldots, E_k, not\, G_{k+1}, \ldots, not\, G_r$$

where $A(\hat{x})$, called the *conclusion* of the rule, is an atom over tuple of arguments $\hat{x}$. The right-hand-side is called the *body* of the rule, and is a conjunction of queries on external contexts. Precisely, each of the E_is and each of the G_is (where $k > 0$ and

[2] 'ECA' rules stands for 'Event-Condition-Action' rules, and specify reaction to events. Such rules are present in some form in all agent-oriented programming languages and frameworks.

$r \geq 0$) can be either of the form $DQ_i(\hat{x_i}) : c_i$ or of the form $DQ_i(\hat{x_i}) : q_i$ where: DQ_i is a datalog query over tuple of arguments $\hat{x_i}$; c_i is a context listed in the local ABox with is role, and thus locally known to the agent; $q_i = \text{Role@inst}(role_i)$ is a context name obtained by means of a standard query Role@inst to the institutional agent inst (notation '@' is borrowed from standard DACMASs), performed by providing the context role $role_i$. We assume that all variables occurring in $A(\hat{x})$ and in each of the G_is also occur in the E_is. The comma stands (here and below) for conjunction. Assuming (without loss of generality) that all the $\hat{x_i}$s have the same arity, when the rule is *triggered* (see below) then the E_is may produce a set of tuples, some of which are discarded by the G_is. Finally, $A(\hat{x})$ is obtained as a suitable projection. Within an agent, different bridge rules have distinct conclusions. The management operations and function are defined separately (see below).

E.g., Role@inst(*student_office*) would return the name of the context corresponding to the student office. An agent-to-context datalog query is of the form $Q \,\text{:-}\, A_1, \ldots, A_n, not\, B_1, not\, B_m$ with $n+m > 0$ where the left-hand-side Q can stand in place of the right-hand-side (so, by abuse of notation ' :- ' stands for '$\equiv$'). Each the A_is is either an atom or a binary expression involving connectives such as equality, inequality, comparison, applied to variables occurring in atoms and to constants. Each atom has a (possibly empty) tuple of arguments and can be either ground, i.e., all arguments are constants, or non-ground, i.e, arguments include both constants and variables, to be instantiated to constants in the query results. All variables which occur either in Q or in the B_is also occur in the A_is. Intuitively, the conjunction of the A_is selects a set of tuples and the B_is rule some of them out. Q is essentially a placeholder for the whole query, but also projects over the wished-for elements of the resulting tuples.

While bridge rules occurring in contexts are meant to be automatically applied whenever the present data state entails the rule body, bridge rules in agents are meant to be proactively activated by the agent itself. To this aim, we have introduced suitable variants of DACMAS's communicative and update rules. In our context, a *trigger rule* has the form

$$Q(\hat{x}) \textbf{ enables } A(\hat{y})$$

where: Q is an ECQ_l query, and $\hat{x}$ a set of tuples representing the results of the query; $A(\hat{y})$ is the conclusion of exactly one of the agent's bridge rules. If the query evaluates to true, then $A(\hat{y})$ is (partially) instantiated via one among the answers returned by the query, according to the agent's own choice, and the corresponding bridge rule is triggered.

Since agents' bridge rules are executed neither automatically nor simultaneously, the definition of management function must be revised. First, notice that for each agent included in a DACMACS the underlying logic $(KB_L; Cn_L; ACC_L)$ is such that: KB_L is composed of the global TBox plus the local ABox; ACC_L is determined by the DRL-Lite semantics, according to which elements of Cn_L are computed. If an agent is equipped with n bridge rules, there will be n operators in the agent's management base, one for each bridge rule, i.e., $OP = \{op_1, \ldots, op_n\}$. Each of them which will at least make the acquired knowledge compatible with the global TBox (possibly by means of global ontologies and/or by means of the techniques discussed in [20] and/or via forms of meta-reasoning, cf., e.g., [21] for an overview). F_L^{OP} is defined as in Section 2.1, but instead of a single management function there will now be n management functions $mng_{b_1}, \ldots, mng_{b_n}$, again one per each bridge rule. They can however be factorized within a single agent's management function with signature (as in MCSs), for agent a,

$$mng^a : 2^{F_L^{OP}} \times KB^L \to 2^{KB_L} \setminus \emptyset$$

which specializes into the mng_is according to the bridge-rule head.

Whenever a bridge rule is triggered, its result is interpreted as an agent's generated event and is reacted to via a special ECA rule: this functioning is similar to *internal events* in the DALI agent-oriented programming language [22, 11]. In particular, A *bridge-update rule* has the form **on** $A(\hat{x})$ **then** $\beta(\hat{x})$ where: $A(\hat{x})$ is the conclusion of exactly one bridge rule, and $\hat{x}$ a set of tuples representing the results of the application of the bridge rule; $\beta(\hat{x})$ specifies the operator, management function and actions to be applied to $\hat{x}$, which may imply querying the ABoxes of the agent and of the institutional agent, so as to re-elaborate the agent's ABox.

We now need for DACMACS agents an enhanced agent specification (augmented w.r.t. the DACMAS one, which included only the agent's name and communication and update rules, where also the agent's ABox was left implicit). In particular, the enhanced agent specification is a tuple $\langle a, \mathcal{A}_a, mng^a, \mathit{\Pi} \rangle$, where a is the agent specification name, $\mathcal{A}_a$ is the agent's ABox, mng^a is the agent's management function and $\mathit{\Pi}$ is the set of rules characterizing the agent. In particular, $\mathit{\Pi} = \mathit{\Pi}_{cu} \cup \mathit{\Pi}_{btu} \cup \mathit{\Pi}_{aux}$, where $\mathit{\Pi}_{cu}$ is the set of communicative and update rules, $\mathit{\Pi}_{btu}$ is the set of bridge, trigger and bridge-update rules, and $\mathit{\Pi}_{aux}$ the set of the necessary auxiliary rules. Notice that, though not explicitly mentioned, auxiliary rules where implicitly present also in the definition of DACMASs, unless one considered all necessary auxiliary definitions as built-ins.

The definition of data state and of equilibria must also be extended. In particular, if M is a DACMACS where $(C_1, \ldots, C_j)$ are the composing contexts and $(A_{j+1}, \ldots, A_n)$ the composing agents, $j \geq 0, n > 0$, a data state of M is a tuple $S = (S_1, \ldots, S_n)$ such that each S_i is an element of Cn_i. If $S = (S_1, \ldots, S_n)$ is a data state for a DACMACS M, a bridge rule ρ occurring in each composing context or agent is *potentially applicable* in S iff S entails its body. For contexts, entailment is the same as in MCSs. For agents, entailment implies that all queries in the positive part of the rule body succeed w.r.t. S, and no query in the negative part of the rule body succeeds w.r.t. S. A bridge rule is *applicable in a context* whenever it is potentially applicable. A bridge rule with head $A(\hat{y})$ is *applicable in an agent* A_j whenever it is potentially applicable and there exists a trigger rule of the form $Q(\hat{x})$ **enables** $A(\hat{y})$ in the specification of A_j such that $Cn_j \models Q(\hat{x})$. Let $app(S)$ be the set of bridge rules which are applicable in data state S.

Desirable data states are those where each S_i is acceptable according to ACC_i. A data state for a DACMACS M is an equilibrium iff, for $1 \leq i \leq n$, there exists $kb'_i = mng_i(app(S), kbi)$ such that $S_i \in ACC_i(kb'_i)$.

In [5] we have extended some of the formal results which hold for mMCS and for DACMAS to the new setting. In fact, assuming that all management functions (both those related to agents and those related to contexts) are *local consistency (lc-)preserving*, then we can prove that a DACMACS M is locally consistent, i.e., that the addition of the new data does not introduce inconsistencies in the agents' ABoxes w.r.t. the global TBox. Lc-preserving management functions are those which perform "careful" updates according to [20]. The execution semantics of a DACMACS can be defined by extending the *transition system* defined for DACMAS.

3 Extensions

In this paper we propose extensions to the DACMACS approach so as to make data extraction from contexts more general and flexible. We remain for the sake of simplicity in the hypothesis of datalog queries (with negation) as defined above, which correspond to the select-join-project-setminus fragment of SQL. Notice that all variables in a datalog query are implicitly existentially quantified. Such queries correspond to DRL-Lite existential conjunctive queries (ECQs) in the simple form where a conjunct is simply an atom rather than a disjunction (cf. [23] and the references therein). Moreover, they allow for tractable ontological query-answering.

In order to allow for ontological queries, we have to assume contexts to be equipped with their private TBox. We may also assume that both agents and contexts are able to access some kind of global ontology to achieve interoperability (think, e.g., simply of a dictionary) but we will not go into detail on this aspect. In an extended DACMACS also agents will have their own private TBox. However, we assume that each agent's TBox is to be intended as the union of the global and the local TBox, and that such union is consistent. As seen below, the local TBox can be updated in consequence of data acquisition from contexts via bridge rules. We also assume that the global TBox in a DACMACS constitutes, in the terminology of [24], a *protected fragment*, in the sense that no update made to the agents' local TBox fragments is enabled to affect the global TBox.

As a motivating example, consider a University that intends to establish which students are eligible for a job on campus. The conditions are that the student's family income must be low, and the student's grades must be good or excellent. In a DACMACS representing the University, the agent *student_office_secretary* may consult the contexts *student_office* for grades and *tax_office* for family income. In this case, in DACMACS one would define a bridge rule, providing (in datalog) the ontological definition of desired results:

$eligible(stud)$ **determinedby**
$\quad low\_income(stud)\ :\ tax\_office,$
$\quad not\ nonexcellent(stud)\ :\ student\_office$

$low\_income(stud)$:- $student(stud),\ (1)$
$\quad family\_income\_less\_than\_threshold(stud, inc, inc\_threshold).$
$nonexcellent(stud)$:- $student(stud),\ (2)$
$\quad grade\_less\_than\_threshold(stud, grad, grade\_threshold).$

The assumption which is implicit in this bridge rule is that contexts are equipped with an engine which is able to parse and execute datalog queries, and that the terminology used in the query fits with the one used in contexts. Otherwise, as mentioned one might assume that contexts might try to parse queries based upon global ontologies and meta-ontologies (in case of failure in doing so, the query would consequently fail). The other assumption is that the secretary is aware of the criteria for defining an income as low and a student as excellent.

Let us assume instead that the secretary is not aware of such criteria. A first extension consists in the possibility for the querying agent of omitting query definitions, in the example (1) and (2), thus delegating to the contexts the task of finding, in their own TBox, ontological definitions suitable for answering a query (in case of failure in doing so, the query would consequently fail). A second extension allows the querying agent, in the example the secretary, to acquire from contexts not only query answers, but also the ontological definition on which they are based. In this case, the above bridge rule would be reformulated as:

$$\begin{array}{l} \mathit{eligible}(\mathit{stud})\ \textbf{determinedby} \\ \qquad \mathit{low\_income}(\mathit{stud})\ :\ \mathit{tax\_office}\ :\ \mathit{D1}, \\ \qquad \mathit{not\ nonexcellent}(\mathit{stud})\ :\ \mathit{student\_office}\ :\ \mathit{D2} \end{array}$$

where $D1$ and $D2$ are variables, to be instantiated respectively to (1) and (2) by the tax office and student office contexts. Then, the definition of a bridge rule can be reformulated as follows.

Definition 2. *In an Extended DACMACS , a bridge rule occurring in an agent's specification has the following form.*

$$A(\hat{x})\ \textbf{determinedby}\ E_1, \ldots, E_k, \mathit{not}\, G_{k+1}, \ldots, \mathit{not}\, G_r$$

$A(\hat{x})$, called the conclusion *of the rule, is an atom over tuple of arguments $\hat{x}$. The right-hand-side is called the* body *of the rule, and is a conjunction of queries on external contexts. Precisely, each of the E_is and each of the G_is (where $k > 0$ and $r \geq 0$) can be either of the form $DQ_i(\hat{x_i})\ :\ c_i$ or of the form $DQ_i(\hat{x_i})\ :\ q_i$, or of the form $DNQ_i(\hat{x_i})\ :\ c_i\ :\ D$ where: DQ_i is a datalog query over tuple of arguments $\hat{x_i}$; c_i is a context listed in the local ABox with is role, and thus locally known to the agent; $q_i = \mathit{Role@inst}(\mathit{role}_i)$ is a context name obtained by means of a standard query Role@inst to the institutional agent inst (notation '@' is borrowed from standard DACMASs), performed by providing the context role role_i; $DNQ_i(\hat{x_i})$ is a datalog atom representing a query, for which no definition is provided; D is a variable, to be instantiated by c_i to the definition of $DNQ_i(\hat{x_i})$. The contents to which D gets instantiated are coped with by means of a suitable operator and management function. We assume that all variables occurring in $A(\hat{x})$ and in each of the G_is also occur in the E_is. The comma stands for conjunction. Within an agent, different bridge rules have distinct conclusions. The management operations and function are defined separately (see previous section and Definition 4 below).*

The operator management function relative to the update of the agent's local TBox via the acquired ontological contents is unique, i.e., it is the same for every bridge rule and for every context which is queried. We extend the definition of DACMACS agent so as to introduce the local TBox together with the possibility of updating it, in consequence of the application of enhanced bridge rules. We assume to provide a unique

TBox operator and management function to be applied whenever a TBox update is performed.

Definition 3. *An agent specification is a tuple $\langle a, \mathcal{T}_a, \mathcal{A}_a, op_T, mng^a, mng_T, \Pi\rangle$, where: a is the agent specification name; $\mathcal{T}_a$ is the agent's local TBox; $\mathcal{A}_a$ is the agent's ABox; mng^a is the agent's management function; op_T and mng_T are the local TBox update operator and management function (seen in Definition 4 below); Π is the set of rules characterizing the agent. In particular, $\Pi = \Pi_{cu} \cup \Pi_{btu} \cup \Pi_{aux}$, where Π_{cu} is the set of communicative and update rules, Π_{btu} is the set of bridge, trigger and bridge-update rules, and Π_{aux} the set of the necessary auxiliary rules.*

Definition 4. *The local TBox management function has signature*

$$mng_T : op_T \times \mathcal{TB} \rightarrow 2^{\mathcal{TB}} \setminus \emptyset$$

where $\mathcal{TB}$ is the set of all possible TBoxes, so $\mathcal{T}_a \in \mathcal{TB}$.

We require op_T and thus mng_T to respect the principles stated in [24] for TBox update:

(i) **Satisfiability Preservation** Updates should preserve satisfiability of basic concepts and roles.
(ii) **Protection** Updates should preserve the protected fragment of the TBox.

A naive strategy for achieving such principles and also for preserving consistency of the ABox w.r.t. the TBox can be the following.

Definition 5. *Let $DNQ_i(\hat{x_i}) : c_i : D$ be a query to context c_i occurring in an agent a's bridge rule. Let $DNQ_i(\hat{x_i}) = R(\hat{x})$. Take as given the agent's global and local TBox. The TBox management function mng_T^a associated to the agent a can specify for the update operator op_T a behavior that generates the updated agent's local TBox as follows.*

1. *Create a new concept or relation in the local TBox (depending upon the arity of R), named R^{c_i}. That is, the query result is "indexed" via the name of the context providing its definition.*
2. *for each atom $A(\hat{y})$ occurring in the conjunction of atoms to which D has been instantiated create a new concept or relation in the local TBox (depending upon the arity of A), named A^{c_i}.*
3. *If R^{c_i} is an atomic concept and atomic concept R already exists either in the global or in the local TBox, then create the assertion $A^{c_i} \sqsubseteq A$. If A^{c_i} is an atomic concept and atomic concept A already exists either in the global or in the local TBox, then create the assertion $A^{c_i} \sqsubseteq A$.*
4. *Analyze the variable occurrences in $DNQ_i(\hat{x})$ and D and their concatenations so as to generate inclusiveness and disjointess assertions for the new concepts/relations.*
5. *Use the query results to generate new ABox assertions, that are by definition consistent with existing ones.*

The definition of data state and of bridge rule applicability is the same as in DACMACS . The definition of equilibrium has now to encompass TBox management.

Definition 6. *A data state for an Extended DACMACS M is an equilibrium iff, for $1 \leq i \leq n$, there exists $kb'_i = mng_T \circ mng_i(app(S), kbi)$ such that $S_i \in ACC_i(kb'_i)$.*

It is easy to see that:

Proposition 1. *The TBox management function mng^a_T for generic agent a specified as stated in Definition 5 fulfills the update principles (i)-(ii).*

It follows that:

Proposition 2. *Given mng^a_T as specified as stated in Definition 5 and lc-preserving management function mng_a, then the overall agent a's management function $mng^a_T \circ mng_a$ is lc-preserving.*

Therefore, local consistency of an extended DACMACS can be guaranteed by suitable TBox update management. In fact, analogously to mMCS also for DACMACSs it holds that:

Proposition 3. *Let M be an extended DACMACS such that all management functions associated to the composing agents and contexts are lc-preserving. Then M is locally consistent.*

The execution semantics of an extended DACMACS can be defined by extending the *transition system* defined for DACMAS in [1].

Nice results are provided in [1] for DACMASs about verification using a variant of μ-calculus, which is a powerful temporal logic used for model checking of finite-state transition systems, able to express both linear-time temporal logics such as LTL and branching-time temporal logics such as CTL and analogous (for the formal definition of this variant, the reader may refer to [25] and to the references therein). These results rely upon *state boundedness*, in the sense that for each agent in a DACMAS ther exists a bound on the number of data items simultaneously stored in its ABOX. These results can be extended to DACMACS assuming state-boundedness (with the same bound) for both agents' ABoxes and contexts' data instances. For extended DACMACS , we have also to assume that all the agents' and contexts' local TBoxes can be consistently merged into the single global TBox. The problem here is twofold:
(i) It must be assumed that contexts' TBoxes (as well as ABoxes, actually) are publicly available.
(ii) It must be assumed that a satisfactory merge of the TBoxes is indeed possible, where for *satisfactory* we mean without loss of information with respect to the sources.

These assumption can be realistic in specific settings, but are not in the general case. Therefore, extended DACMACS are more flexible and more suitable for real-world agent-data-context integration application. However, the added flexibility makes verification a much more problematic issue.

4 Concluding Remarks

In this paper we have extended DACMACS which are in turn a generalization of DACMAS, which is a formalization of ontology-based and commitment-based multi-agent systems. Our extension allows a system to include not only agents but also external contexts, which can return to agents not only query answers, but also the ontological definitions they are based upon. The objective is that of modeling real-world situations where agents and contexts actually constitute the analogous of a kind of data integration system. In this vision, both single agents and the overall system is able to evolve by incorporating new data/knowledge, and new ontological information. Similarly to what argued for DACMAS and DACMACS , instances of Extended DACMACS are readily implementable via standard publicly available technologies.

We are not aware of similar proposals in rule-based approaches. Future directions include a full implementation and realistic experiments. This work stems from a practical experimentation in personalized home healthcare (that for lack of space we could not report here), where we had to cope with the gap existing between the ontological description of each patient's health history and individuality, and the standard ontological descriptions which can be supposed to be available in medical knowledge sources.

References

1. Montali, M., Calvanese, D., De Giacomo, G.: Verification of data-aware commitment-based multiagent system. In: Proc. of AAMAS 2014. (2014) 157–164
2. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The descript. log. handbook: Theory, implementation, and applications. Cambridge Univ. Press (2003)
3. Singh, M.P.: Towards a formal theory of communication for multi-agent systems. In Mylopoulos, J., Reiter, R., eds.: Proc. of the 12th Intl. Joint Conf. on Artificial Intelligence, Morgan Kaufmann (1991) 69–74
4. Singh, M.P.: Commitments in multiagent systems: Some history, some confusions, some controversies, some prospects. In Paglieri, F., Tummolini, L., Falcone, R., Miceli, M., eds.: The Goals of Cognition. Essays in Honor of Cristiano Castelfranchi, College Publications, London (2012) 601–626
5. Costantini, S.: Knowledge acquisition via non-monotonic reasoning in distributed heterogeneous environments. In Calimeri, F., Ianni, G., Truszczynski, M., eds.: Logic Programming and Nonmonotonic Reasoning, 13th International Conference, LPNMR 2015, Proceedings. Lecture Notes in Computer Science, Springer (2015)
6. Bordini, R.H., Hübner, J.F.: BDI agent programming in agentspeak using *Jason* (tutorial paper). In Toni, F., Torroni, P., eds.: Computational Logic in Multi-Agent Systems, 6th International Workshop, CLIMA VI, Revised Selected and Invited Papers. Volume 3900 of Lecture Notes in Computer Science., Springer (2006) 143–164
7. Hindriks, K.V., van der Hoek, W., Meyer, J.C.: GOAL agents instantiate intention logic. In Artikis, A., Craven, R., Cicekli, N.K., Sadighi, B., Stathis, K., eds.: Logic Programs, Norms and Action - Essays in Honor of Marek J. Sergot on the Occasion of His 60th Birthday. Volume 7360 of Lecture Notes in Computer Science., Springer (2012) 196–219
8. Dastani, M., van Riemsdijk, M.B., Meyer, J.C.: Programming multi-agent systems in 3apl. In Bordini, R.H., Dastani, M., Dix, J., Fallah-Seghrouchni, A.E., eds.: Multi-Agent Programming: Languages, Platforms and Applications. Volume 15 of Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer (2005) 39–67

9. Fisher, M.: MetateM: The story so far. In Bordini, R.H., Dastani, M., Dix, J., Fallah-Seghrouchni, A.E., eds.: PROMAS. Volume 3862 of Lecture Notes in Computer Science., Springer (2005) 3–22
10. Bracciali, A., Demetriou, N., Endriss, U., Kakas, A., Lu, W., Mancarella, P., Sadri, F., Stathis, K., Terreni, G., Toni, F.: The KGP model of agency: Computational model and prototype implementation. In: Global Computing: IST/FET Intl. Workshop, Revised Selected Papers. LNAI 3267. Springer-Verlag, Berlin (2005) 340–367
11. Costantini, S., Tocchio, A.: The DALI logic programming agent-oriented language. In: Logics in Artificial Intelligence, Proc. of the 9th European Conf., Jelia 2004. LNAI 3229, Springer-Verlag, Berlin (2004)
12. Costantini, S.: The DALI agent-oriented logic programming language: References (2012) at URL `http://www.di.univaq.it/stefcost/info.htm`.
13. Bordini, R.H., Braubach, L., Dastani, M., Fallah-Seghrouchni, A.E., Gómez-Sanz, J.J., Leite, J., O'Hare, G.M.P., Pokahr, A., Ricci, A.: A survey of programming languages and platforms for multi-agent systems. Informatica (Slovenia) **30**(1) (2006) 33–44
14. Brewka, G., Eiter, T.: Equilibria in heterogeneous nonmonotonic multi-context systems. In: Proc. of the 22nd AAAI Conf. on Art. Int., AAAI Press (2007) 385–390
15. Brewka, G., Eiter, T., Fink, M.: Nonmonotonic multi-context systems: A flexible approach for integrating heterogeneous knowledge sources. In: Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday. Volume 6565 of Lecture Notes in Computer Science., Springer (2011) 233–258
16. Omicini, A., Ricci, A., Viroli, M.: Artifacts in the a&a meta-model for multi-agent systems. Autonomous Agents and Multi-Agent Systems **17**(3) (2008) 432–456
17. Apt, K.R., Bol, R.: Logic programming and negation: A survey. The Journal of Logic Programming **19-20** (1994) 9–71
18. Brewka, G., Eiter, T., Fink, M., Weinzierl, A.: Managed multi-context systems. In: IJCAI 2011, Proc. of the 22nd Intl. Joint Conf. on Art. Int., IJCAI/AAAI (2011) 786–791
19. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-lite family and relations. CoRR **abs/1401.3487** (2014)
20. Calvanese, D., Kharlamov, E., Nutt, W., Zheleznyakov, D.: Updating ABoxes in DL-lite. In: Proc. of the 4th Alberto Mendelzon Intl. Workshop on Foundations of Data Management. Volume 619 of CEUR Workshop Proceedings., CEUR-WS.org (2010)
21. Barklund, J., Dell'Acqua, P., Costantini, S., Lanzarone, G.A.: Reflection principles in computational logic. J. of Logic and Computation **10**(6) (2000) 743–786
22. Costantini, S., Tocchio, A.: A logic programming language for multi-agent systems. In: Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf.,JELIA 2002. LNAI 2424, Springer-Verlag, Berlin (2002)
23. Heymans, S., Eiter, T., Xiao, G.: Tractable reasoning with DL-programs over datalog-rewritable description logics. In: ECAI 2010 - 19th European Conf. on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proc. (2010) 35–40
24. Zheleznyakov, D., Calvanese, D., Kharlamov, E., Nutt, W.: Updating TBoxes in DL-lite. In Haarslev, V., Toman, D., Weddell, G.E., eds.: Proc. of the 23rd Intl. Workshop on Description Logics (DL 2010. Volume 573 of CEUR Workshop Proceedings., CEUR-WS.org (2010)
25. Calvanese, D., De Giacomo, G., Montali, M., Patrizi, F.: Verification and synthesis in description logic based dynamic systems (abridged version). In Faber, W., Lembo, D., eds.: Web Reasoning and Rule Systems - 7th Intl. Conf., RR 2013. Volume 7994 of Lecture Notes in Computer Science., Springer (2013)