Knowledge Acquisition via Non-Monotonic Reasoning in Distributed Heterogeneous Environments

Stefania Costantini

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica (DISIM),
Universitá degli Studi dell'Aquila, Italy, stefania.costantini@univaq.it

Abstract. The role of data and knowledge exchange is becoming increasingly important. The approach of DACMAS [1] proposes a quite general modeling of Multi-Agent Systems (MAS), including data representation in a MAS via DRL-Lite Ontologies. Yet, data/knowledge acquisition from heterogeneous sources which are not agents and which are external to the MAS is not provided. In the Knowledge Representation and Reasoning field, this topic is coped with by mM-CSs (Managed Multi-Context Systems). In this paper, we propose an integration of the two approaches into DACMACSs. The aim is to obtain an enhanced integrated flexible framework where non-monotonicity is present: in the modalities for defining knowledge acquisition; in the conditions for triggering the acquisition and for knowledge exploitation.

1 Introduction

The importance of data and knowledge exchange in Artificial Intelligence applications is constantly increasing. In many application fields it is particularly important to comprise and elaborate information provided by multiple sources. Distributed autonomous evolving applications are the realm of intelligent software agents and Multi-Agent Systems (MAS). Many approaches to MAS are based upon computational logic, where logical agents are able to reason, and to perform non-monotonic reasoning when needed (cf. the Proceedings of the "Computational Logic in Multi-Agent Systems" (CLIMA) Workshop Series). Logic-based data management and exchange are therefore important issues in logical agents. Such agents are required to perform non-monotonic reasoning both in defining and executing patterns for knowledge exchange and in the modalities for knowledge exploitation.

DACMAS (Data-Aware Commitment-based Multi-Agent Systems) [1] is a quite general model of multi-agent systems, which explicitly introduces elements of knowledge representation in the specification of such systems. Knowledge and data in DACMASs are in fact represented according to the DRL-Lite Description Logic [2]. A DACMAS always includes an *institutional* agent which owns a "global" TBox, specifying the domain in which the MAS operates, whereas each participating agent is equipped with its local ABox. The DACMAS approach is particularly interesting because, apart from a general specification of data management and communicative features, it remains very general about an agent program's definition, and can be thus specialize to many existing agent-oriented logic languages.

However, the possibility for agents to interact with components of a different nature which are part of the agents' environment is an important aspect which is lacking in DACMASs. As a matter of fact, even the most comprehensive software engineering approaches such as, e.g., "Agents and Artifacts" (cf. [3] and the references therein) assume that external sources can be either "agentified" or "wrapped", as in fact, citing the proposers, "An artifact is a computational, programmable system resource, that can be manipulated by agents, residing at the same abstraction level of the agent abstraction class". In a real setting this may be too strong an assumption, at least concerning external knowledge bases. What must be necessarily known about such external sources is that they can be queried and should return certain kinds of information: however, modification is in general not allowed, and availability of a better description cannot be taken for granted. Moreover, wrapping a source according to a certain specification would imply that all agents in a MAS have a uniform view of that source. This assumption may be reductive under the perspective of knowledge representation and reasoning: in fact, each agent might in principle interpret, elaborate, reason upon and incorporate the acquired knowledge in a way which is especially tailored to its tasks and objectives.

In the Artificial Intelligence and Knowledge Representation field, the Multi-Context Systems (MCS) approach has been proposed to model information exchange among heterogeneous sources [4–6]. MCSs are defined so as to drop the assumption of making such sources in some sense homogeneous: rather, the approach deals explicitly with their different representation languages and semantics. Heterogeneous sources are called "contexts" and in the MCS understanding they are fundamentally different from agents as they do not have reactive, proactive and social capabilities, but can simply be queried and updated. MCSs have evolved from the simplest form [4] to managed MCS (mMCS) [7], and reactive mMCS [6] for dealing with external inputs such as a stream of sensor data.

In this paper we propose to combine DACMAS with mMCS. The aim is to obtain the formalization of multi-agent systems which are able to flexibly interact with heterogeneous external information sources, by means of suitable agent-oriented modalities. In mMCSs, communication among contexts occurs via special non-monotonic "bridge rules", which are assumed to be automatically applied whenever applicable. We allow also agents to be equipped with bridge rules. However, we devise an entirely new mechanism for bridge-rule activation. In the proposed approach, that we call DACMACS (Data-Aware Commitment-based managed Multi-Agent-Context Systems) agents do not mandatorily apply bridge rules. Rather, being autonomous entities, DACMACS agents are able to apply bridge rules proactively and non-monotonically, depending upon requirements which are specific for each agent.

The main features of the proposal are the following. (1) Agents can query (sets of) contexts, but contexts cannot query agents. (2) Agents are equipped with bridge rules, whose application is however activated 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.

The integration among DACMASs and mMCSs has been purposedly devised so as to be smooth, thus requiring as few modifications to the formal machinery as possible. DACMACS therefore: (i) is based upon the combination of well-established existing technologies; (ii) provides a uniform representation and a formal semantics for the resulting class of systems; (iii) preserves most of the interesting formal properties of both mMCSs and DACMASs that therefore extend to DACMACS; (iv) introduces however novel agent-oriented interaction patterns; (v) introduces semantic notions that constitute an enhancement over both DACMASs and reactive mMCSs.

The paper is organized as follows: in Sections 2 and 3 we provide the necessary background notions about mMCSs and DACMASs. In Section 4 we present and illustrate the new approach of DACMACSs, and discuss its properties. In Section 5 we present (in a nutshell, for lack of space) an example that we consider as representative of a potential real-world application domain of DACMACSs.

2 Background: mMCS

Managed Multi-Context systems (mMCS) [5, 7, 6]) model the information flow among multiple possibly heterogeneous data sources. The device for doing so is constituted by "bridge rules", which are similar to datalog rules (cf., e.g., [8] for a survey about datalog and the references therein for more information) but allow for knowledge acquisition from external sources, as in each element of their "body" the "context", i.e. the source, from which information is to be obtained is explicitly indicated. In the short summary of mMCS provided below we basically adopt the formulation of [6], which is simplified w.r.t. [7].

Reporting from [5], 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, whose elements are data items or "facts" (in [5] these sets are called "belief sets"; we adopt the more neutral terminology of "data sets"); $ACC_L : KB_L \rightarrow 2^{Cn_L}$ is a function which defines the semantics of L by assigning to each knowledge-base a set of "acceptable" sets of consequences. A managed Multi-Context System (mMCS) $M = (C_1, \ldots, C_n)$ is a heterogeneous collection of contexts where $C_i = (L_i; kb_i; br_i)$ and L_i is a logic, $kb_i \in KB_{L_i}$ is a knowledge base (below "knowledge base") and br_i is a set of bridge rules. Each such rule is of the following form, where the left-hand side $o(s)$ 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.

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

For each bridge rule included in a context C_i, it is required that $kb_i \cup o(s)$ 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 $o(s)$ 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_w, $j < w \leq m$, does not belong to the consequences of context c_w. While in standard MCSs the head s of a bridge rule is simply added to the "destination" context's knowledge base kb, in managed MCS the kb is subjected to an elaboration w.r.t. s according to a specific operator o and to its intended semantics. Formula s itself can be

elaborated by o, for instance with the aim of making it compatible with kb's format, or by performing more involved elaboration.

If $M = (C_1, \ldots, C_n)$ is an MCS, a data state (or, equivalently, belief/knowledge state), is an n-uple $S = (S_1, \ldots, S_n)$ 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 knowledge 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 data state S.

For a logic L, $F_L = \{s \in kb \,|\, kb \in KB_L\}$ is the set of formulas occurring in one of 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 $F_L^{OP} = \{o(s) \,|\, o \in OP, s \in F_L\}$. The semantics of such statements is given by a management function, which maps a set of operational statements and a knowledge base into a set of modified knowledge bases. In particular, a management function over a logic L and a management base OP is a function $mng : 2^{F_L^{OP}} \times KB_L \rightarrow 2^{KB_L} \setminus \emptyset$

Semantics of an mMCS is 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 $1 \leq i \leq n$, $kb'_i = S_i \in ACC_i(mng_i(app(S), kb_i))$.

Thus, an equilibrium is a global data state composed of acceptable data states, one for each context, encompassing inter-context communication determined by bridge rules and the elaboration resulting from the operational statements and the management functions. Equilibria may not exist (where conditions for existence have been studied, and basically require the avoidance of cyclic bridge-rules application), 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 an mMCS where all management functions are lc-preserving is locally consistent.

Notice that bridge rules are intended to be applied whenever they are applicable, so inter-context communication automatically occurs, though mediated via the management functions.

3 Background: DACMAS

We remind the reader that in DLR-Lite Ontologies [2] we have the following. (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, defined in accordance with the TBox. 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.

Knowledge and data in agents composing a DACMAS are in fact represented in DRL-Lite: a DACMAS always includes an *institutional* agent which owns a "global"

TBox, specifying properties of the domain in which the MAS operates, whereas each participating agent is equipped with its local ABox.

Communication among DACMAS agents may occur through simple event-based reaction, or via *commitments*, which are a relatively recent though very well-established general paradigm for agent interaction (cf. [9] and the references therein). A commitment $C_{x,y}\,(ant,\, csq)$ in particular relates a *debtor agent* x to a *creditor agent* y where x commits to bring about csq whenever ant holds. Commitment lifecycle (they can be created, fulfilled, canceled, etc.) is managed by a so-called "commitment machine".

Formally, a DACMAS (Data-Aware Commitment-based Multi-Agent System) is (from [1]) a tuple $\langle \mathcal{T}, \mathcal{E}, \mathcal{X}, \mathcal{I}, \mathcal{C}, \mathcal{B} \rangle$ where: (i) $\mathcal{X}$ is a finite set of agent specifications; (ii) $\mathcal{T}$ is a global DLR-Lite TBox, which is common to all agents participating in the system; (iii) $\mathcal{I}$ is a specification for the "institutional" agent; (iv) $\mathcal{E}$ 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); (v) $\mathcal{C}$ is a contractual specification; (vi) and $\mathcal{B}$ is a Commitment Box (CBox). The global TBox includes the list of the names of all participating agents in connection to their specifications. Each agent is equipped with a local ABox, consistent with the global TBox, where however the ABoxes of the various agents are not required to be mutually consistent. The set consisting of the union of the global TBox and an agent's local ABox constitutes the agent's knowledge base. The institutional agent is a special agent who is aware of every message exchanged in the system, and can query all the ABoxes. In addition, it is responsible of the management of commitments, whose concrete instances are maintained in the Commitment Box $\mathcal{B}$, and it does so based on the *Commitment Rules* in $\mathcal{C}$, which define the commitment machine. An execution semantics for DACMASs is provided in [1], in terms of a transition system constructed by means of a suitable algorithm.

Each 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.

A communicative rule has the form $Q(r, \hat{x})$ **enables** $EV(\hat{x})$ **to** r where: Q is an ECQ_l query, which is an ECQ with *location argument* l of the form $@Ag$ to specify the agent to which the query is directed (if omitted, then 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 query Q 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 can make an enquiry about the name r of the provider of some service: if several names are returned, only one is chosen. Then it sends to the selected 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[1] 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

[1] As it is well-known, "ECA" rules stands for "Event-Condition-Action" rules, and specify reaction to events.

sent or received, as specified in the rule itself:
on $EV(\hat{x})$ **to** r **if** $Q(r, \hat{x})$ **then** $\alpha(r, \hat{x})$ **(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. For instance, after subscription to a service an agent can receive offers and issue orders, the latter case determining the creation of a commitment (managed by the institutional agent).

An agent specification is a tuple $\langle sn, \mathcal{A}_{sn}, \Gamma \rangle$, where sn is the agent specification name, $\mathcal{A}_{sn}$ the local ABox, and Γ is the set of communicative and update rules characterizing the agent.

4 DACMACS

Let a logic, a management base and management functions be as specified in Section 2. The definition of a DACMACS (Data-Aware Commitment-based managed Multi-Agent-Contexts Systems) extends that of DACMASs as the set of participating agents is augmented with a set of contexts, to be understood as external data/knowledge sources which can be consulted by agents.

The global TBox specifies, as in DACMASs, the ontological specification shared by all agents in the system. We introduce explicit lists of agents' and context's names. We also introduce a global ABox, not present in DACMAS, which: links agents' names with their specification, and contexts' names with their *roles*, where a role specifies the function that a context assumes for agents. E.g., a context's name $studoff$ may correspond to context role $student\_office$, and context name $poldept$ to $police\_department$. For simplicity, we assume here that roles are specified as constants. We also assume that context names include all the information needed for actually posing queries (e.g., context names might coincide with their URIs). Each DACMACS agent may specify in its local ABox more information about context roles, or also list additional contexts (with corresponding roles) which are specifically known to that agent.

In DACMACS, agents' specification (seen below) is richer than in DACMAS, as it may include bridge rules, similar to those of mMCSs, whose activation is however proactive rather than mandatory. Each agent is equipped with local management functions.

Definition 1. *A DACMACS (Data-Aware Commitment-based managed Multi-Agent-Context System) is a tuple* $\langle \mathcal{T}, \mathcal{E}, \mathcal{N}, \mathcal{X}, \mathcal{Y}, \mathcal{I}, \mathcal{A}, \mathcal{C}, \mathcal{B} \rangle$ *where* $\mathcal{T}, \mathcal{E}, \mathcal{X}, \mathcal{I}, \mathcal{C}$ *and* $\mathcal{B}$ *are the same as for DACMASs, and: (i)* $\mathcal{N}$ *is a set of agents' names, listing the agents (beyond the institutional agent) which compose the MAS; (ii)* $\mathcal{Y}$ *is a set of contexts' names, listing the contexts which are globally known to the MAS; (iii)* $\mathcal{A}$ *is a global ABox, which is consistent with the global TBox and with all agents' local ABoxes.*

Context names in the global and local ABoxes might also be linked to the information about the related query language. However, again for the sake of simplicity though without loss of generality, we assume that all contexts accept datalog queries, in particular of the following form.

Definition 2. *An agent-to-context datalog query is defined as follows:*

$$Q \text{ :- } A_1, \ldots, A_n, not\, B_1, \ldots, not\, B_m \text{ with } n + m > 0$$

where the left-hand-side Q can stand in place of the right-hand-side. The comma stand for conjunction, and each of 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.

Each context may include bridge rules, of the form specified in section 2, where however the body refers to contexts only, i.e., contexts cannot query agents. The novelty of our approach is that also agents may be equipped with bridge rules, for extracting data from contexts. Agents' bridge rules are more general than those in mMCSs, as a bridge rule combines information extracted from a number of contexts, where each context returns data/knowledge according to a query of the form defined in Definition 2. As in DACMASs, we assume that the new data items possibly added to the agent's ABox belong to the same countably infinite domain Δ. However, as seen below bridge rules in agents are not automatically applied as in mMCSs: rather, they are proactively activated by agents upon need.

Definition 3. *A bridge rule occurring in an agent's specification has the following form.*

$A(\hat{x})$ **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 $r \geq 0$) 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 (defined according to Definition 2) over tuple of arguments $\hat{x_i}$; c_i is a context listed in the local ABox with its role, and thus locally known to the agent; $q_i = Role@inst(role_i)$ 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 for conjunction. Assuming (without loss of generality) that all the $\hat{x_i}$s have the same arity, when the rule is* triggered *(see Definition 4 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 Definition 5 below).*

E.g., Role@inst(*student_office*) returns the name of the context corresponding to the student office. There is, as anticipated before, an important difference between bridge

rules in contexts and bridge rules in agents. Each bridge rule occurring in a context is meant to be automatically applied whenever the present data state of the entire DACMACS entails the rule body. The new knowledge corresponding to the rule head is added (via the management function) to the context's knowledge base. Instead, bridge rules in agents are meant to be proactively activated by the agent itself. To this aim, we introduce new kinds of rules not present in DACMAS.

Definition 4. *A (timed)* trigger rule *is a proactive rule of the form*

$Q(\hat{x})$ **enables** $A(\hat{y})$ $[Time \mid Frequency]$

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 query Q 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. This means, for a bridge rule of the form $A(\hat{x})$ **determinedby** *$E_1, \ldots, E_k, not\, G_{k+1}, \ldots, not\, G_r$, that the conclusion $A(\hat{x})$ is allowed to be derived if the rule is applicable, i.e., if system's present data state entails its body. The options in square brackets, if specified, state that the bridge rule with conclusion $A(\hat{y})$ should be triggered either at time instant $Time$ (according to the system's own internal measurement), or at a frequency $Frequency$, expressed in terms of time instants.*

Since agents' bridge rules are executed neither automatically nor simultaneously, we have to revise the definition of management function with respect to the original definition of Section 2. 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 global ABox and the agent's 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 then 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 may perform any form of reasoning, though it will at least make the acquired knowledge compatible with the global TBox, by means of techniques that we do not consider here. F_L^{OP} is defined as in Section 2, but instead of a single management function there will now be n management functions $mng_{b_1}, \ldots, mng_{b_n}$, one per each bridge rule. They can however be factorized within a single agent's management function with signature (as in MCSs), for agent i,

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

which specializes into the mng_{b_i}s 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:

Definition 5. *A* bridge-update rule *has the form $upon A(\hat{x})$* **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 $\hat{x}$.*

The definition of β may imply querying the ABoxes of the agent and of the institutional agent, performing knowledge format conversion, belief revision, etc., so as to

re-elaborate the agent's ABox. The minimal requirement is that of keeping the ABox consistent: i.e., in mMCS terms, the management function specified in β is assumed to be lc-preserving. DACMACS agents' specification is therefore augmented w.r.t. that of DACMAS ones:

Definition 6. *An agent specification is a tuple $\langle sn, \mathcal{A}_{sn}, \Gamma \rangle$, where sn is the agent specification name, $\mathcal{A}_{sn}$ the local ABox, and Γ is the set of rules characterizing the agent. In particular, $\Gamma = \Gamma_{cu} \cup \Gamma_{btu} \cup \Gamma_{aux}$, where Γ_{cu} is, like in DACMAS, the set of communicative and update rules; in addition Γ_{btu} is the set of bridge, trigger and bridge-update rules, and Γ_{aux} the set of the necessary auxiliary rules (defining the predicates occurring in the body of the former rules).*

The definition of data state and of equilibria must be extended with respect to those provided in Section 2, and not only because a data state now includes both contexts' and agents' sets of consequences. As mentioned, in MCSs a bridge rule is applied whenever it is applicable. This however does not in general imply that it is applied only once, and that an equilibrium, once reached, lasts forever. In fact, contexts are in general able to incorporate new data items from the external environment (which may include, as discussed in [6], the input provided by sensors). Therefore, a bridge rule is in principle re-evaluated whenever a new result can be obtained, thus leading to evolving equilibria. In DACMACSs, there is the additional issue that for a bridge rule to be applied it is not sufficient that it is applicable in the MCS sense (i.e.-, when its body is true), but it must also be triggered by a corresponding timed trigger rule.

Similarly to what is done in Linear Time Logic (LTL) we assume a discrete, linear model of time where each state/time instant can be represented by an integer number. States $t_0, t_1, \ldots$ can be seen as time instants in abstract terms, as we have $t_{i+1} - t_i = \delta$, where δ is the actual interval of time after which we assume a given system to have evolved. In DACMACSs, both agents' and contexts' knowledge base contents may change not only in consequence of communication, but also due to interaction with an external environment. Thus, each agent's or context's knowledge base can be subjected to *updates*, where an update is understood as a finite set composed of new facts asserted to be true, and/or new facts asserted to be false, and/or already-known facts changing their truth value. Updates are not in general just "blindly" accepted, rather they are incorporated into the existing knowledge by means of some kind of *update operator* which performs knowledge base revision, for which several techniques exist that for lack of space we are not able to mention here.

In the definitions below, given a DACMACS M, we assume that $(C_1, \ldots, C_j)$ are the composing contexts and $(A_{j+1}, \ldots, A_m)$ the composing agents, $j \geq 0, m > 0$; let $n = j + m$.

Definition 7. *Let M be a DACMACS including $n = j + m$ contexts and agents. Let $\Pi = \Pi_1, \Pi_2, \ldots$ be a sequence of finite updates performed to agents' and contexts' private knowledge bases at time instants $t_1, t_2, \ldots$, respectively. Thus, for $i > 0$, $\Pi_i = \langle \Pi_{iC_1}, \ldots \Pi_{iC_j}, \Pi_{iA_{j+1}}, \ldots \Pi_{iA_m} \rangle$ is a tuple composed of the updates performed to each context and agent. Let $\mathcal{U}_E$, $E \in \{C_1, \ldots, C_j, A_{j+1}, \ldots, A_m\}$, be the* update operator *that each agent or context employs for incorporating the new information, and let $\mathcal{U}$ be the tuple composed of all these operators.*

Definition 8. *Let M be a DACMACS including $n = j + m$ contexts and agents. A* timed *data state of M at time T is a tuple $S^T = (S_1^T, \ldots, S_n^T)$ such that each S_i^T is an element of Cn_i at time T. The timed data state S^0 is an equilibrium according to an analogous definition as for mMCSs, i.e. iff, for $1 \leq i \leq n$, there exists $S_i^0 \in ACC_i(mng_i(app(S^0), kb_i))$.*

Each transition from a timed data state to the next one, and consequently the definition of an equilibrium, is determined both by the update operators and by the application of bridge rules, where in a DACMACS a bridge rule is applied only when it has been triggered, according to the specified timing/frequency.

Definition 9. *Let M be a DACMACS, and let S^T be a timed data state of M at time T. A bridge rule ρ occurring in each composing context or agent is* potentially applicable *in S^T iff S^T entails its body. For contexts, entailment is the same as in MCSs. For agents, entailment implies that all queries in the rule body succeed w.r.t. S^T. 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 $Cn_j \models Q(\hat{x})$. If the trigger rule is timed and specifies an application time $\hat{T}$, then it must be $T = \hat{T}$. If the trigger rule is timed and specifies a frequency $\hat{F}$, then it must be $T = \hat{F} * N$, for some N [2]. Let $app(S^T)$ be the set of bridge rules which are applicable in data state S^T.*

In DACMACSs, bridge rules are applied after performing the update of each component's knowledge base. So, data states following the initial one S^0 are equilibria if they are composed of acceptable data sets, considering however knowledge base update, which is performed before inter-agent-context communication determined by bridge rules.

Definition 10. *Let M be a DACMACS, let E_i, $1 \leq E \leq n\}$ be any composing agent/context, $\mathcal{U}_i$ be the corresponding update operator, Π be a given update sequence and Π_T^i be the update performed upon E_i at time T. Let S^T be a timed data state of M at time $T \leq 0$. A timed data state of M at time $T + 1$ a is an equilibrium iff, for $1 \leq i \leq n$, $S_i^{T+1} \in ACC_i(mng_i(app(S^T), \mathcal{U}_i(kb_i, \Pi_T^i)))$.*

Definition 11. *Let M be a DACMACS. A* safe evolution trajectory *of M w.r.t. a sequence Π of updates is a sequence $S^0, S^1, \ldots$ where the S^is, $i \geq 0$ are timed data states of M such that $\forall T \geq 0$, S^T is an equilibrium, and S^{T+1} is obtained from S^T and Π as specified in Definition 10.*

The above-introduced notions of equilibrium and safe evolution trajectory generalize those defined in [6] for reactive mMCSs, where input from sensor data is considered, and is coped with by extending bridge rules applicability. In our case, the introduction of an explicit update operator allows for sensor data incorporation, as well as for any other knowledge base manipulation. Updates and bridge rules application are independent, where at each state either the update or the set of applicable bridge rules or both can be empty or not. Bridge rules application is however performed upon the contents of the updated knowledge bases.

[2] To be more precise, T is required to be the nearest approximation for $\hat{T}$ or respectively $\hat{F} * N$.

4.1 Properties of DACMACSs

In the terminology of [5], we require all management functions (both those related to agents and those related to contexts) to be *local consistency (lc-)preserving*. We also require the update operators to preserve consistency of agents' and contexts' knowledge bases. We thus obtain the following, which is the analogous to Proposition 2 stated in [5] for mMCSs:

Proposition 1. *Let D be a DACMACS such that all update operators and management functions associated to the composing agents and contexts are lc-preserving. Then D is locally consistent.*

Global consistency of all data sets in a DACMACS is not required here, as it is not required in DACMAS in the first place: in fact, incoherences may reflect agents' local views. By adopting the A-ILTL (Agent-Oriented Interval LTL) interval linear temporal logic [10, 11], several interesting properties of a DACMACS can be defined and verified. For instance, for proposition φ, it can be checked whether φ holds for agent A included in a DACMACS (i.e., in the above terminology, $A \in \{A_{j+1}, \ldots, A_n\}$) in some equilibrium reached at a certain time or within some time interval, given sequence Π of updates and a resulting safe evolution trajectory.

A simple property that might be interesting to check can be for instance $F_n\, \varphi_{A_v}$ which specifies that φ becomes true in agent A_v at some time less than or equal to n (F standing for "eventually", or "finally"). Given the above definitions, this accounts to check whether

$(\varphi \in S_v^0) \vee (\varphi \in S_v^n) \vee \exists m$ where $0 < m < n$ such that $F_m\, \varphi_{A_v}$

For finite updates this and many other properties are decidable, where the complexity of check depends upon the complexity of the update operators and of the management functions. However, to deal with complexity an approach is proposed in [11] for run-time rather than a-priori checking of such properties.

5 Medical Example

We will now introduce an example (necessarily simple due to lack of space) inspired to a real-life medical problem. Let us model the situation in DACMACS's terms, assuming to have an agent called *Paula* like the patient we imagine it is designed to care for. Let us assume the agent's ABox to be defined as follows (we basically adopt the datalog notation, because it is widely known). The patient, Paula, is a 78 years old lady. Being 78 implies being elderly, which is not a disease but nevertheless it is, from the sanitary point of view, a special condition.

$$person(paula). \quad age(paula, 78).$$
$$special\_condition(X, elderly) \text{ :- } person(X), age(X, A), A > 70.$$

Let us now assume that this elderly lady has certain health conditions, and takes certain medicaments in relation to these conditions. In particular, she takes an anticoagulant (and other medicaments not listed here) for coping with a cardiac insufficiency.

$has\_disease(paula, cardiac\_insufficiency)$.
$takes\_medicament(P, anticoagulant)$:- $has\_disease(P, cardiac\_insufficiency)$.

Unfortunately, Paula presents some disquieting symptoms, namely extreme weakness, that can be attributed to low hemoglobin level. Then, it is stated that hemoglobin level is obtained within a blood test, and that its value is worrying (cannot be explained in a trivial way) if it is less than 10.

$has\_symptom(paula, extreme\_weakness)$.
$symptom\_cause(extreme\_weakness, hemoglobin\_level)$.
$blood\_test(hemoglobin\_level)$.
$anomalous\_value(P, hemoglobin\_level)$:- $person(P),\ value(P, hemoglobin, V), V \leq 10$.

The following trigger rule states that, if a person P has a symptom S which has a possible cause T detectable as a blood test with value V, then: the agent has to obtain from the institutional agent $inst$ the specification name of the context corresponding to the blood test records via the query $Spec@inst(\textit{Rec}, blood\_tests\_records)$, which returns its result in the variable *Rec* . So, the bridge rule for testing Paula's hemoglobin level V by querying this context can be proactively activated.

$person(P)\ AND\ has\_symptom(P, S)\ AND$
$symptom\_cause(S, T)\ AND\ blood\_test(T)\ AND$
$Spec@inst(\textit{Rec}, blood\_tests\_records)$ **enables** $blood\_test\_result(Paula, T, V, \textit{Rec})$.

The bridge rule simply queries *Rec* for the required value V of parameter T for person P (assuming that the blood test record context by default returns the most recent value).

$blood\_test\_result(P, T, V, \textit{Rec})$ **determinedby** $value(P, T, V)\ :\ Rec.$

The result is then simply added to the agent's ABox by a bridge-update rule:

upon $value(P, T, V)$ **then** $add(value(P, T, V))$.

The following trigger rule states that if a person which has a certain disease and some special condition has an anomalous value of some factor V, then the medical diagnosis context *Diag* (whose name is obtained as before from the institutional agent) must be consulted in order to obtain the list C of possible causes.

$person(P)\ AND\ has\_disease(P, D)\ AND$
$special\_condition(P, S)\ AND\ anomalous\_value(P, V)\ AND$
$Spec@inst(\textit{Diag}, medical\_diagnosis)$ **enables** $poss\_causes(P, D, S, V, C, \textit{Diag})$.

A corresponding sample bridge rule can be the following (more realistically however, several diagnostic knowledge bases might be consulted via more involved queries).

$poss\_causes(P, D, S, V, C, \textit{Diag})$ **determinedby** $poss\_causes(P, D, S, V, C) : \textit{Diag}.$

The related bridge-update rule starts a reasoning process to infer from given parameters and possible causes C the most plausible cause R.

upon $poss\_causes(P, D, S, V, C, \textit{Diag})$ **then** $infer\_plausible(P, D, S, V, C, R)$.

Actually R was, for Paula, an internal hemorrhage due to the long-termed use of anticoagulants. So, her life was saved and her relatively good health and satisfactory quality of life have been restored.

6 Concluding Remarks

In this paper we proposed DACMACS, which is a flexible framework augmenting agents with controlled interaction and knowledge exchange with external heterogeneous data sources. We have defined notions of timed data states and equilibria. Similarly to what argued for DACMAS, instances of DACMACS are implementable via publicly available technologies, taking as basis any of the existing logical agent-oriented programming languages. The execution semantics of a DACMACS can be defined by extending the *transition system* defined for DACMAS in [1]. Thus, results provided in [1] for DACMASs about verification using the μ-calculus can be extended to DACMACSs, though we defer discussion to a future paper.

Among the future directions of this work we may include the possibility to equip also contexts with ontological descriptions, and cope with ontology conversions in bridge rules application. We might also consider the aspect, not tackled in mMCSs either, of actual applicability of bridge rules, which occurs only if needed contents are delivered within a deadline. The issue of verification and of the balance among a-priori and run-time verification deserves further exploration.

References

1. Montali, M., Calvanese, D., De Giacomo, G.: Verification of data-aware commitment-based multiagent system. In: Proc. of AAMAS 2014
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. 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
4. 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
5. 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
6. Brewka, G., Ellmauthaler, S., Pührer, J.: Multi-context systems for reactive reasoning in dynamic environments. In: ECAI 2014, Proc. of the 21st European Conf. on Art. Int., IJCAI/AAAI (2014)
7. 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
8. Apt, K.R., Bol, R.: Logic programming and negation: A survey. The Journal of Logic Programming **19-20** (1994) 9–71
9. Singh, M.P.: Commitments in multiagent systems: Some history, some confusions, some controversies, some prospects
10. Costantini, S.: Self-checking logical agents. In: Proc. of the Eighth Latin American Workshop on Logic, Languages, Algorithms and New Methods of Reasoning LA-NMR 2012. Volume 911 of CEUR Workshop Proc., CEUR-WS.org (2012) 3–30 Invited Paper, Extended Abstract in Proc. of AAMAS 2013.
11. Costantini, S., De Gasperis, G.: Runtime self-checking via temporal (meta-)axioms for assurance of logical agent systems. In: Proc. of LAMAS 2014, 7th Workshop on Logical Aspects of Multi-Agent Systems, held at AAMAS 2014. (2014) 241–255