Towards a Causal Framework for Intelligent Agents Development

No Author Given

No Institute Given

Abstract. In this paper we present a *Causal Artificial Intelligence Design* (CAID) theory that borrows notions from Classical philosophy for modeling intelligent agents. Principles introduced by this theory are used for extending a goal-driven BDI architecture and implementing what we call *Causal Agent*. This architecture incorporates causal formalisms like Pearl's Do calculus and C+ which are adapted to Semantic Web knowledge representations. Our approach includes an ontological agent description that enables and justifies the instantiation of agents as part of a plan.

Key words: Causality, Metaphysics, Intelligent Agent, Semantic Web, Bayesian Causal Networks

1 Introduction

Development of intelligent agents have originated several architectures and approaches ranging in complexity from the reactive to the rational agent. The use of mental states and explicit intentions have produced one of the most successful approaches [7]. Nevertheless, as long as the complexity of domains increases, general assumptions become insufficient to tackle every problem. SOAR [10] proposes incorporating semantic knowledge on its reasoning in order to formalize common sense. Approaches like LORA [14] and Jadex [11] have proposed to incorporate explicit definitions of agent capabilities in order to minimize the necessity of modeling other agents in open systems.

The assumption of self-interested agents with unknown purposes makes a really hard task to determine if a coalition with other agent will be profitable for both parts. For this purpose, cooperation between agents have been modeled through shared intentions [6] or social commitments [13]. Encoding utilitarian theories in agents' reasoning have proved to be an efficient mechanism for agent's decision but finally requires knowledge about all the participants when a global goal must be achieved [12]. All these approaches have an individual perspective where an agent is considered intelligent as long as can achieve its goals, and where cooperation is justified as long as report individual benefits.

We introduce an approach for modeling intelligent agents from a causal perspective borrowed from classical philosophy. The objective is to make explicit the purpose of agents and their constitution in order to facilitate their interaction and to find a middle point between the individual and the global perspective.

This paper is organized as follows: in section 2 are presented some philosophical foundations of our approach; section 3 presents rector principles of our proposal, meanwhile section 4 introduces an agent architecture based on these principles, including an actual implementation. In section 5 is compared our approach with other existing ones and in section 6 are given some closing remarks.

2 Philosophical Foundations

Aristotle unified classic realism and idealism in a middle position that proposed that real things are the material representation of forms called essences, abstract ideas which are materialized in concrete entities. This idea has evolved through the history of mankind, discussing mainly the precedence and existence of matter and ideas.

Aristotle conceived reality constituted by *entities* or *beings* which common characteristics are abstracted by human mind in concepts named *essences* [1]. These characteristics or accidents were classified by Aristotle as intrinsic, extrinsic and mixed. *Intrinsic accidents* include *quantitative* (age, size, etc.), *qualitative* (color, shape, etc.) and *relational* (fatherhood, nationality, etc.) accidents, that is, what internally identify an entity. *Extrinsic accidents* are relative to *time* (birth date, duration, etc.), *place* (position), *possession* (property) and *disposition* (sit, stand, etc.). *Mixed accidents* explain interaction among entities: *action* is present in an entity when originates movement or change in another, meanwhile *affection* is present in entities that receive passively the action of another.

Aristotle used the notion *potency* for representing the capacity of an entity for showing certain accident. *Act*, on the other hand, is the actual presence of the accident on the entity. Having certain accident in potency doesn't imply that the entity shows it actually, but only denotes possibility. For example, a given *statue* can be described by its actual accidents (size, color, antiquity, location, creator, etc.) and according to this we can identify the statue we are referring to. On the other hand, an *sculptor* is a person that in potency can sculpt an statue and receive a payment for it.

For explaining change on entities, Aristotle used the notion of Causality. *Causality* refers to the set of all particular "causal" or "cause-effect" relations. Most generally, causation is a relationship that holds between events, properties, variables, or states of affairs. Causality implies at least some relationship of dependency between the cause and the effect. *Cause* chronologically precedes the *effect*.

According to Aristotle, all possible causes fall into four main groups. The *material cause* refers to the matter or the substratum, known too as part-whole causation. The *formal cause* identifies the essence, the pattern, the form, or the structure, also known as whole-part causation. The *efficient cause* is the primary moving change or the agent and its action (*agent causation*). Finally, the *final cause* or agent intention is the goal, the plan, the end, or the good. According to this theory causes that intervene in any change, including the creation of

entities, falls in one of these categories. The entity that receives the effect of this change is called *caused thing*.

Continuing with our example, we can identify the main causes of the Caesar's statue at Louvre: its formal cause is Julius Caesar, its efficient cause is the sculptor Nicolas Coustou, its material cause is the rock of which it was made, and its final cause could be to receive some payment for it.

3 Causal Artificial Intelligence Design

It is presented the *Causal Artificial Intelligence Design* (CAID) approach, constituted by a set of principles that guide the design of intellignet agents and Multi-Agent Systems:

1. *Total Causality* [TC]. Any software agent or system can be specified and instantiated through its four main causes: formal, material, efficient and final.
2. *Intentional Creation* [IC]. Every agent in a system is created with a purpose derived from its creators intention.
3. *Ontological Commitment* [OC]. Created agents will intend to goals consistent with its essence and will act in consequence.
4. *Causal Effect Accountability* [CEA]. Agent's decision is based on the accountable effects of its actions.

According to the *Total Causality* principle, an agent can be instantiated through its main causes. The *formal cause* answers to the question *how is it?* with an agent definition given in terms of accidents the agent can show, i.e. its accidents in potency. The *efficient cause* answers the question *who creates it?* with the identifier of the agent that instantiate it. The *material cause* answers the question *what does compose it?* with the software implementations for the agent itself and its components (sensors and actuators). The *final cause* answers the question *why is it created?* and it's given in terms of those goals that the agent must achieve in the instantiator's plan. Even when the final cause is not a parameter on the agent's instantiation, it is the justification of its instantiation.

Capabilities of an agent are given by its definition and include the type of actions it can perform and the kind of goals it can achieve. If the capabilities of agent classes are known and their instantiation is controllable, planning can consider actions and plans controlled by non-existing agents. The resulting plan must indicate the sequence of actions required for achieving the original goal, as well as the set of agents that will perform them, given by tuples $PlanOperation(operation_i, agent_i)$. For a tuple $PlanOperation(op_1, ag_1)$ where ag_1 doesn't currently exist, the instantiation of ag_1 is enabled by the *Total Causality* principle. The *Intentional Creation* principle proposes that the instantiation of ag_1 has op_1 as final cause. If the created agent has the capability of instantiating other agents, this instantiation can be done recursively. The original goal, owned by the former agent, is decomposed hierarchically through this process.

The kind of goals the agent must achieve according to its ontological definition (agent class) are called *essential goals*. The *Ontological Commitment* principle establishes that ontologically, i.e. in order to be consistent with its definition, an instance of this agent class must agree to achieve goals compatible with its essential goals. The unique condition that would avoid that an agent wouldn't accept a commitment for one of these goals is that its current goals would be in conflict with the new goal. Once that the agent commits to achieve a new goal it will execute all the available plans until achieving the goal (success) or exhauste them (fail). In both cases, the agent delegating the goal must be informed of the outcome.

The *Causal Effect Accountability* principle establishes that an agent needs to know the effects of its own actions in order to choose those that will allow him to achieve the goals it has committed to. If the agent can identify or is informed of indirect effects of its actions it can consider them to make better decisions. Additionally, the creator agent can inform to the created agent how desirable (or not) are such effects, enforcing (or inhibiting) agents actions. This last attribution will allow to indicate the agent "*how well or bad is it doing it*". Causal relations and preferences are delivered by the agent responsible for a plan to those agents that execute it.

4 Causal Agent

We introduce a design methodology and agent architecture that uses the principles of CAID for development of intelligent agents. The proposed architecture extends the goal-driven BDI architecture with a set of elements and operations. An implementation with existing formalisms is proposed at the end of this section.

Definition 1. A *Causal Agent* is an agent modeled [TC] and instantiated [IC] through the four main causes that is considered intelligent as long as it achieves goals consistent with its definition [OC] and makes decisions based on the causal effects of its actions [CEA].

4.1 CAID Elements

The main elements added to a goal-driven BDI by CAID are shown in Figure 1. The agent definition establishes through constraints those accidents an agent can show, among which we can find the set of essential goals, the rules used for revising its beliefs, the plans it controls, and the implementation of sensors and actuators it uses.

Agent definition. Agent definition is made using Description Logics formalisms [2] that support current Semantic Web Ontology language OWL [1]. The

[1] World Wide Web Consortium (W3C). Web Ontology Language (OWL). http://www.w3.org/2004/OWL/

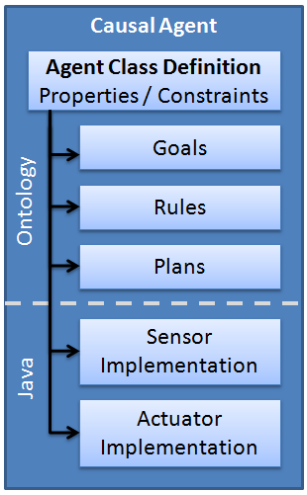

Fig. 1. Main elements of a Causal Agent.

agent class is represented by a concept name and its definition is given by a set of constraints that identify the roles (accidents) that can be associated to the agent. The *agent definition* has the general form:

$$AgClass \sqsubseteq SupCls \sqcap \geqslant 0\ R_i.C_i \sqcap = n\ R_j.C_j \sqcap R_k = V_k\ for\ 0 \leq \{i,j,k\} \leq m \tag{1}$$

where $AgClass$ is the name of the agent class, $SupCls$ is the agent class from which inherits properties and constraints, R_i are *potential roles* with range on objects of the class C_i, R_j *requires* n instances of type C_j, and R_k is set to V_k by default (V_k can be an known instance or a literal value).

This definition of agents allows to express a specialized hierarchy of agents. Concepts $SoftAg$ and $HumanAg$, descendants of $Agent$ are used for representing the class of software agents and human agents. $HumanAg$ subclasses denote roles played by humans in the system. The role type (accident category) can be indicated through a role hierarchy and the use of constant roles like R_{action}. If the agent can perform action α then role R_α is included as potential role on the agent's definition and it is asserted the relation $R_\alpha \sqsubseteq R_{action}$ in the schemas definition (TBox).

Essential goals. Essential goals are represented in the definition with the constructor $AgClass \sqsubseteq hasEssentialGoal = G$ where G is of type $Goal$. A goal has the general form $Goal(pre, final, cancel, until)$ where pre is the precondition that identifies the objects used and that will be modified, $final$ is another condition indicating the desired state, $cancel$ is the cancelation condition (ending with failure), $until$ is the termination condition (ending with success). All the conditions are represented by conjunctive queries, over which query containment

can be determined ($Q_1 \sqsubseteq Q_2$) [9]. The use of $until = \bot$ indicates an *indefinite goal*, on which achieving $final$ doesn't imply the termination of the goal.

Rules. Rules are associated with the constructor $knowsRule = R$ where R is of type *Rule*. Rules are used for transforming perceptions into beliefs and for revising beliefs. Rules can be used too for describing the possible outcomes of an action execution. Rules' predicates include concepts and roles names used for defining the agent and other objects in the domain.

Plans. Plans are described semantically as individuals in a public repository. Plan definition must serve to the following purposes: 1) determining which actions are contained in the plan, 2) describing the execution order, and 3) identifying goals that can be achieved through its execution. Depending on the actions contained in the plan and the capabilities of an agent it is possible to determine if the agent can execute the entire plan (*total control*) or part of it (*partial control*). This condition can be codified through role assertions of the type $controlsPlanT(AgClass, Plan)$ or $controlsPlanP(AgClass, Plan)$, respectively. Even when this calculation can be expensive computationally, it can be done off line, previous to system's startup. On the other hand, satisfiability of goal G by plan P can be calculated evaluating $G.pre \sqsubseteq P.pre \wedge G.final \sqsubseteq P.final$, where $P.pre$ and $P.final$ are conjunctive queries that represent the initial and final state associated to the plan execution. Goal satisfiability can be persisted through the role assertion $satisfyGoal(P, G)$.

Components. *Sensors* and *actuators* are implemented in Java through given interfaces. These interfaces allow sensors to gather information and store it on perceptions, and actuators to execute actions indicated in the beliefs set. Both of them are considered *agent components* and are described semantically as concepts (classes) that are associated to the agent class through the constraint $= n\ hasSensor.S_1$ (or $= n\ hasActuator.A_1$), indicating the number of sensors (or actuators) of type S_1 (or A_1) the agent must posses. Component description must include references to its software implementation and a list of affection or action roles that implement. For example, constraint $A_1 \sqsubseteq implements = R_\alpha$, indicates that actuator A_1 implements the action role R_α. On a similar way, the Java implementation of the agent is indicated through a similar constraint, $AgClass \sqsubseteq hasImplementation = Impl$, where $Impl$ describes the Java class and the code location.

4.2 CAID Functions

Additionally to functions performed in a goal-driven DBI agent, CAID principles introduce new functions.

Agent instantiation. Denoted by an action role, described by an action rule and implemented through an actuator, agent instantiation is achieved indicating: the agent class, the instantiator agent (ID and type), and the identifier for the new agent. The agents platform must provide the appropriate functionality. During set up, the agent implementation load instances of sensors and actuators implementations, set default values and loads essential goals, controlled plans and known rules, all of them indicated on its definition. Finally, it associates the constant $Self$ used on rules and plans, to its assigned identifier. If some exception is raised during this process, it notifies to its efficient cause; otherwise notifies the successful instantiation.

Intention transmission. The operation Op_1 that motivated the instantiation of agent Ag_1 is transmitted to the instantiated agent as a goal through a commitment. The commitment has the form $XCC(db, cr, G, w)$, where db is the debtor agent, cr is the creditor agent, G is the goal to achieve, and w is the importance factor assigned by cr in relation to the upper level goal from which G was derived. If G is contained in some agent's essential goal G_0, denoted $G \sqsubseteq G_0$, then the commitment is accepted; otherwise, it is rejected.

Goal containment is done by evaluating $G.pre \sqsubseteq G_0.pre \wedge G.final \sqsubseteq G_0.final$. Once that the agent accepts a commitment for this goal, it activates those plans that satisfy the respective G_0. Due to query containment, $satisfyGoal(P, G_0)$ and $G \sqsubseteq G_0$ implies that $satisfyGoal(P, G)$.

Planning. Planning is an expensive task that only certain agents must be enabled to do. Implemented through an actuator, planning receives a goal as input and can be done on two ways: considering agent instantiation as an operator in a logic program, or using an algorithm that apply essential goals as operators and latter determines which agent would be the best candidate for it (instantiating it if there is none available).

In both cases, the plan produced by this algorithm must throw: a set of agents to instantiate and a set of commitments to delegate. Planning can produce more than one feasible plan due to: multiple paths for reaching the goal, goals in common among different agent classes, and the possibility of delegating the commitment to an existing or a new agent. From the set of feasible plans, one must be selected. Plan selection can use heuristics like minimizing the number of commitments or maximizing the probability of success of the sequence of actions.

Plan deployment. Plan deployment consists on instantiating required agents and delegating commitments in the plan. If some agent doesn't accept one of the commitments in the plan, the commitment can be assigned to another of the same class or to a new agent. Once that the goal is achieved, commitments derived from the plan are released and agents without active commitments are disposed.

Commitment operations. The following functionality of Singh's commitment formalism is incorporated: 1) definite commitments are discharged by the agent once that the *until* condition is met, 2) *final*'s condition achievement of indefinite commitments is notified to the creditor, 3) cancelation of a commitment is notified once that the *cancel* condition holds, 4) a commitment can be discharged by the creditor. Discharging, releasing or cancelation of commitments motivates the revision of goals and active plans.

Causal relations exchange. Causal relations are exchanged using standard communication protocols in the format $CR(cause, effect, likelihood)$ where $cause$ and $effect$ are conditions represented by conjunctive queries, and likelihood represents the conditional probability $P(effect|cause)$ that the sender beliefs. This information can be used for extending the causal network of an agent or can be considered for adjusting the probability of an existing causal relation.

Additionally, *preferences* can be modeled associating a cost or utility C to certain condition P, denoted $Preference(P, C)$ on which C is positive for indicating preference and negative for indicating an undesired condition. The absolute value of C indicates how preferred or not is P. Preferences are searched over the nodes of a plan in order to include the cost C in the action selection procedure.

4.3 Implementation

The implementation of a Causal Agent extends a goal-driven BDI architecture adding the elements illustrated in Figure 2 and redefining some of the actual functions of the original approach. Our current implementation is made using the JADE framework [3]. The domain ontology, that includes agents and objects, as well as the core ontology containing the elements described above are defined using OWL. The Jena Framework[2] is used for loading the domain ontology, maintaining sets of assertions (ABoxes) in memory, and applying semantic reasoning derived from the definitions and the given entailment rules. Currently, it is applied only RDFS reasoning, i.e. object oriented reasoning, which is computed in polynomial time.

Java Interfaces. The Java `Sensor` interface requires the implementation of the function `boolean sense(ABox perceptions)` which stores the information gathered in the ABox `perceptions` using the schemas defined in the domain ontology, and returns `True` if something is perceived. The perception must be consistent with the sensor description S that establishes $implements(S, R_\rho)$, where the affection role R_ρ has for range the type of objects perceived; for instance, an ACL message. Similarly, the Java `Actuator` interface implements the function `boolean act(ABox beliefs, ABox nextBeliefs)` which looks in the `beliefs` Abox for instances of the action roles it implements in order to

[2] Jena - A Semantic Web Framework for Java. http://jena.sourceforge.net/

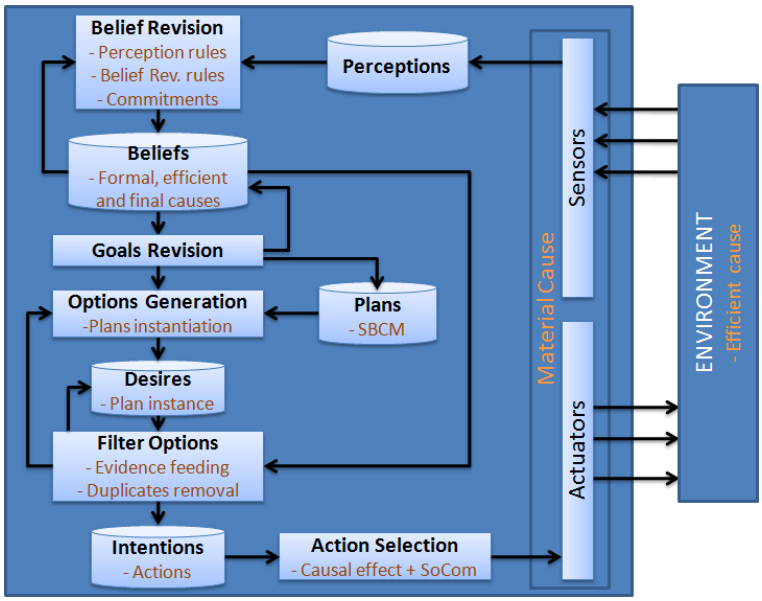

Fig. 2. CAID BDI architecture.

trigger their execution. The result of its execution is updated in the `nextBeliefs` ABox and the function returns `True` if at least one action was executed.

ABoxes. Beliefs and perceptions are represented by independent ABoxes. Along the inference process there are two versions of the beliefs set: the previous beliefs set and the current beliefs set. Both beliefs set are almost the same, except that the former stores the actions executed in the last step and the latter stores the effects of such actions. In the beliefs set can be found the agent definition, its efficient cause and its active goals.

Rules. Belief revision rules are implemented as RDF rules and executed with the Jena inference support on the current set of beliefs. For the incorporation of perceptions into beliefs is used a rule representation with the form $PercRule(perc, curr, e^+, e^-)$, where *perc* and *curr* are conjunctive queries executed over the current set of perceptions and beliefs, respectively; results obtained from *perc* are replaced in *curr* and the last result is used for constructing statements from e^+ and e^- that are added or removed in the current set of beliefs. The execution of these rules is made through SPARQL select query executions. The rule engine has the characteristic of creating instances on runtime for representing new individuals added by e^+.

Goals Revision. After beliefs revision is performed a procedural checking of goals' satisfaction and commitments' discharging. *final* and *until* goals' con-

ditions are evaluated as SPARQL select queries in the updated set of beliefs. Commitments discharges and goals achievements are prepared as messages to the respective creditors. Active goals and plans are revised.

Plans. Plans are represented by a formalism that integrates Bayesian networks and semantic annotations [5]. Annotations, represented by DL conjunctive queries, allows to identify action roles in a node and based on the agent class capabilities, controllability of the plan can be determined. The execution order is given by the causal structure and goals achievable by the plan are given by the annotations of the final nodes (F). This formalism was complemented with a set of realizations ($Z_i = z_i$) identifying the context on which the plan can be triggered; annotations of these realizations are used for identifying the precondition in terms of queries that can be executed over the beliefs set.

Options. Options generation consists on identifying those active plans that can be instantiated from the current beliefs set. Annotations of SBCM contexts are evaluated as SPARQL select queries on the current set of beliefs. For every tuple returned by this evaluation a new SBCM instance is created. A SBCM instance is represented by $SBCMInst(B, S)$ where $B = \{(V_i = v_i)|i > 0\}$ is a set of realizations of the Bayesian network and $S = \{(S_i = s_i)|i > 0\}$ is a set of realizations for semantic variables produced by the evaluation of annotations on V.

During goals revision, options with pending nodes are evaluated in the current set of beliefs. Options completely revised are used for parametric learning of the model and removed from the set of current options. Options filtering is used for discarding duplicated options. A new option is considered duplicated if all the semantic variables of its context are contained in the set of semantic variables of a current option.

Intentions. Intentions are those actions that can be executed from the current set of options. An executable action is represented by a controllable node X not intervened and which preconditions are set, i.e. $P(X = True|E = e) \geq \tau$, where $E = e$ is the evidence stored in the option and τ is a threshold that indicates under how much uncertainty the agent can act, usually in the range $[0.8, 1.0]$.

Action selection considers the causal effect of all feasible actions over the final state of the plan and over the preferences identified on it. The causal effect of the action X over the final state of the plan $F = f$ is calculated by $ce(F) = P(F = f|do(X = True), E = e)$. A preference $Pref_1 = Pref(P_1, C_1)$ is identified in the plan if exist some observable variable Z_p in the plan such that $P_1 \sqsubseteq Ann(Z_p = z_p)$. The causal effect over such preferences is calculated on a similar way, $ce(Pref_1) = P(Z_p = z_p|do(X = True), E = e)$. On this way, the score of action X is calculated by:

$$score(X) = ce(F) + \sum_i ce(Pref_i) \cdot C_i \tag{2}$$

for all preferences $Pref_i = Pref(P_i, C_i)$ identified in the plan. If $score(X) < \tau$ the action is inhibited.

The action X with the highest score is selected and its annotation $Ann(X)$ is used for encoding the action to execute. *Action encoding* consist on replacing constants from the SBCM instance in $Ann(X)$ and insert the resulting statements in the beliefs ABox.

5 Discussion

Unlike methodologies like Gaia that require an additional processing of its specification, our proposal uses the ontology formalism for documenting the domain and designing agents on an expressive formal language.

Potential accidents included in the definition of an agent allows to identify those objects that can be found in the agent's beliefs set and that can be accessed by rules and actuators; such definition becomes cumbersome in Jadex. Agent's definitions and the formalization of the instantiation act allows that agents can consider the inclusion of a non-existing agent in planning. The framework allows to perform a hierarchical decomposition of a task allowing an agent to delegate subtasks or subgoals to other agents. As a consequence, it allows to establish protocols dynamically through commitments delegation. Hierarchical planning like the one performed by SOAR [10] requires a global perspective, which we achieve indicating in the essential goals only those aspects of the plan that affect external objects putting aside implementation details.

Inference is made at three levels: semantic, logic and probabilistic. The use of ABoxes for representing beliefs allows to introduce RDFS entailment. Belief revision and perceptions incorporation is controlled through rules compatible with ABoxes. Finally, the SBCM approach allows to close the bridge between the ABox and a probabilistic representation.

The probabilistic representation of plans in Bayesian Network allows to calculate indirect effects beyond a series of actions, something that C+ [8] can only do on a scope of a time step. Agents can calculate the tradeoff between individual intentions and global preferences, inhibiting or enforcing some actions. Similar work has been done introducing custom factors in rules used by agents [4].

Unlike Jadex [11], our approach provides a plan representation that allows synchronizing plan execution through action selection and transfers Java implementation to the layer of sensors and actuators. SBCM instances are used for performing Bayesian parametric learning on the model, which allows to modify the behavior of the agent or detect ineffective plans.

6 Conclusions

It was presented the Causal Artificial Intelligence Design, a theory for modeling intelligent agents based on philosophical causality theory. Aristotelian metaphysics and causality theory is used for defining intelligent agents. The Causal

Agent architecture, based on this theory, is introduced. It was documented an implementation of such architecture using formalisms like goal-driven BDI architectures, Pearl's Do calculus, the nonmonotonic causal logic C+, among others. Semantic web technology is used extensively in the adoption of such formalisms.

As future work is considered modeling intelligent organizations using CAID theory as well as interaction with human users. The incorporation of lite versions of OWL inference is considered too. Particular planning algorithms for this approach can be developed as well as negotiation protocols for causal relations exchange.

References

1. Tomas Alvira, Luis Clavell, and Tomas Melendo. *Metafisica*. EUNSA, 2001.
2. Franz Baader. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, September 2007.
3. Fabio L. Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. Wiley, April 2007.
4. Eva Bou, Maite Lopez-sanchez, and J. A. Rodriguez-aguilar. Towards self-configuration in autonomic electronic institutions. In *In: COIN 2006 Workshops. Number LNAI 4386*, pages 220–235. Springer, 2007.
5. Hector Ceballos and Francisco Cantu. Integrating semantic annotations in bayesian causal models. In *20th International Workshop on Description Logics (DL07)*, pages 527–528, June 2007.
6. P.R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, (42):213, 1990.
7. Mike Georgeff, Barney Pell, Martha Pollack, Milind Tambe, and Mike Wooldridge. The belief-desire-intention model of agency. In Jörg Müller, Munindar P. Singh, and Anand S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555, pages 1–10. Springer-Verlag: Heidelberg, Germany, 1999.
8. Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, Hudson Turner, and Joohyung Lee Vladimir Lifschitz. Nonmonotonic causal theories. *Artificial Intelligence*, 153:2004, 2004.
9. Ian Horrocks, Sergio Tessaris, Ulrike Sattler, and Stephan Tobies. How to decide query containment under constraints using a description logic. In *In Proc. of LPAR2000*, pages 326–343. Springer, 1999.
10. Allen Newell. *Unified Theories of Cognition*. Harvard University Press, 1994.
11. Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. *EXP - In Search of Innovation (Special Issue on JADE)*, volume 3, chapter Jadex: Implementing a BDI-Infrastructure for JADE Agents, pages 76–85. Telecom Italia Lab, 2003.
12. David Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 1-2(94):7–56, 1997.
13. Munindahr P. Singh. Commitments among autonomous agents in information-rich environments. In *Modelling Autonomous Agents in a Multi-Agent World*, pages 141–155, 1997.
14. M. Wooldridge. *Reasoning about Rational Agents*. MIT, 2000.