CILIOS: Connectionist Inductive Learning and Inter-Ontology Similarities for Recommending Information Agents

D. Rosaci

DIMET, Università “Mediterranea” di Reggio Calabria, Via Graziella Loc. Feo di Vito, 89060 Reggio Calabria (Italy)
E-mail: domenico.rosaci@unirc.it

Abstract

For a software information agent, operating on behalf of a human owner and belonging to a community of agents, the choice of communicating or not with another agent becomes a decision to take, since communication generally implies a cost. Since these agents often operate as recommender systems, on the basis of dynamic recognition of their human owners’ behaviour and by generally using hybrid machine learning techniques, three main necessities arise in their design, namely (i) providing the agent with an internal representation of both interests and behaviour of its owner, usually called ontology; (ii) detecting inter-ontology properties that can help an agent to choose the most promising agents to be contacted for knowledge-sharing purposes; (iii) semi-automatically constructing the agent ontology, by simply observing the behaviour of the user supported by the agent, leaving to the user only the task of defining concepts and categories of interest. We present a complete MAS architecture, called Connectionist Learning and Inter-Ontology Similarities (CILIOS), for supporting agent mutual monitoring, trying to cover all the issues above. CILIOS exploits an ontology model able to represent concepts, concept collections, functions and causal implications among events in a multi agent environment; moreover, it uses a mechanism able of inducing logical rules representing agent behaviour in the ontology by means of a connectionist ontology representation, based on neural-symbolic networks, i.e. networks whose input and output nodes are associated to logic variables.

Key words: Connectionist Learning, Ontology Similarities

NOTICE: this is the authors version of a work that was accepted for publication in Information Systems. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made
1 Introduction

The mutual monitoring among agents is a key issue in developing Multi-Agent Systems (MASs) for supporting Web applications. As pointed out in [43], in multi-agent interactions, agents often share a common goal, evaluated through a global utility function. However, agent cannot observe the global state of the environment in which it lives, therefore agents have to communicate with each other in order to share the information needed for deciding which actions to perform. Thus, for the agent, choosing whether to communicate or not with another agent becomes a decision to take, since communication generally implies a cost. In this sense, the mutual monitoring is a way for supporting cooperation among agents, although cooperation is a very broad issue, certainly not limited to mutual monitoring, and usually concerned not only with information sharing (or sharing mutual models including beliefs, desires, intentions) but particularly with goal sharing and task distribution in a joint effort of the agents. In this paper we focus on the particular issue of mutual monitoring and on its contribution in realizing an effective agent cooperation.

In our opinion, three main problems arise in the scenario described above:

1. An agent-based framework addressing the issue of agents mutual monitoring generally provides each agent with an internal representation of both interests and behaviour of the associated human user, usually called ontology [7,57].

2. The next step for implementing mutual monitoring is to detect inter-ontology properties that can support an agent to choose the most promising agents to be contacted for knowledge-sharing purposes. Some of these approaches use as inter-ontology properties the similarity between ontology concepts [51], also by determining synonymies and homonymies [7] between concepts, while other approaches exploit, in addition to similarity, also other properties involving information about the whole agent community, as the reputation that an agent has gained inside the community [6]. All these approaches use a symbolic representation of agent

---

2 In this paper, the expression “symbolic representation” is used in symbiosis with the expression “connectionist representation”: the former exploits symbols for expressing knowledge, the latter uses numerical values for the same purpose. In effect, this distinction is used in this work for highlighting our strategy of exploiting connectionist algorithms as neural networks for implementing learning. In this perspective, connectionism is our way to represent symbolic knowledge in a sub-symbolic way. However, it is worth to note that connectionism is not the only possible sub-symbolic representation (think to genetic algorithm, for instance) for which translation methods for transforming it in a symbolic form have been defined.
ontologies. It is worth pointing out that, in order to detect inter ontology properties, some approaches assume an individualistic view of agents societies, where each agent does not have access to the ontologies of the other agents. This is the viewpoint of most of the so called BDI approaches [47,21]. Differently, other approaches, as [24,52], adopt a “social” view of agents communities, where it is assumed that the ontology of each agent is, even partially, accessible for each other agent. The latter viewpoint is the one adopted in this paper where we deal with inter-ontology similarities.

(3) Another important issue, well known in the agent community, is the necessity that the agents automatically construct their ontologies, by observing the behaviour of the monitored users. While the interaction between an agent and its user is always necessary in defining user interests and representing them into an ontology, the agent should be able of automatically extracting logical rules representing user behaviour and/or causal implications among events. This latter issue needs the exploitation of some inductive methods, that can be either symbolic, as in the case of the Inductive Logic Programming, or connectionist, like neural networks and genetic algorithms. Note that, although the presence of an automatic learning mechanism is often a condition necessary to realize an effective agent autonomy, such a condition is not sufficient, as it is not the only property ensuring autonomy. For instance, an agent represented as an object, when receives a message cannot refuse to respond (it cannot autonomously choose whether to respond or not); in general, no client-server architecture may show autonomy of the server. In particular, autonomous “planning” or “scheduling of tasks” by the agent should also be guaranteed, based on the run time (learning) knowledge the agent has at its disposal.

At the best of our knowledge, all the approaches existing in the literature deal only with one or at most two of the three issues above. In this work, we present a mechanism dealing with all the three issues described above, able of inducing logical rules representing agent behaviour in the ontology by means of a connectionist ontology representation, based on neural-symbolic networks. This mechanism exploits a new ontology representation, called Information Agent Connectionist Ontology Model (IACOM). The idea underlying IACOM is that of combining some recent proposals for representing symbolic knowledge by a connectionist system, as those presented in [12,44], with the classical use of neural networks as function approximators. IACOM derives from [12] the idea of using a neural symbolic network for representing a logic program. Moreover, IACOM introduces the new notion of action for combining this neural symbolic network with classical neural networks that represent the functions contained in the ontology. In our framework, an action is defined as a mathematical relation between an event e and a function f, and represents the fact that f is activated (i.e., it is called by the system for generating an output)
only if \( e \) happens. This notion is implemented by means of special arcs linking the network associated to the logic program and the networks associated to the functions. The resulting network is able to represent the overall behaviour of the agent. Figure 1 graphically describes this idea. The white nodes \( e_1, e_2 \) and \( c \) represent three events (i.e., boolean variables) composing a neural symbolic network, whose structure means that the event \( c \), being associated with the output node, is a consequence of \( e_1 \) and \( e_2 \). Therefore, \( c \) becomes true only if both \( e_1 \) and \( e_2 \) are true. The black nodes compose a traditional three layers neural network, that represents a real function. The input nodes are associated to the inputs of the function (i.e. real variables) while the output node represents the output of the function. The arc between the event \( c \) and the function output is an action arc. It represents the fact that the function output is computed only if the event \( c \) is true. Otherwise, if \( c \) is false, the function output has a undefined value, meaning that the function in this case does not yield any output. The use of the approach presented in [12] gives, on one hand, the possibility to represent an initial background knowledge by a neural-symbolic network. Such a network can be trained for refining the initial knowledge by means of a supervised learning phase that exploits, as training set, the actual user’s behaviour. On the other hand, the choice of this approach allows the obtained knowledge, represented by the network weights, to be re-translated into the symbolic form for making it understandable. Finally, the so obtained symbolic knowledge can be object of a reasoning phase that generates useful deductions.

Note that this approach of constructing an agent ontology can be considered semi-automatic, in the sense that the human user has already the necessity to define concepts and categories of interests, but the agent equipped with its learning algorithm has the capability of discovering logic rules, functions and actions involving these concepts and categories. We also present a complete architecture of a MAS, called Connectionist Learning and Inter-Ontology Similarities (CILIOS), for supporting agent mutual monitoring, attempting to integrate solutions for each of the above mentioned problems. This architecture is conceived as extension of the well-known JADE platform, and can be used for implementing an agent platform for supporting many important tasks as, for instance, generating recommendations in Web communities or supporting Web navigation and file sharing. Finally, we present a series of experimental tests we have conducted for evaluating the capability of our proposed archi-
tecture in supporting agent mutual monitoring, in the case of Collaborative Filtering Recommender Systems. The paper is structured as follows: in Section 2, some related work is discussed. In Section 3 the CILIOS architecture is described in details, while in Section 4 we give a formal description of the IAOM symbolic ontology model. In Section 5 the framework for extracting inter-ontology similarities is presented. In Section 6, we introduce the IACOM connectionist ontology model and propose a method for building agent ontologies by using neural-symbolic networks. Some experiments using CILIOS are described in Section 7 and, finally, in Section 8 we draw our conclusions, by discussing both advantages and limitations of our proposal.

2 Related Work

Many fascinating discussions have arisen in the Artificial Intelligence (AI) field on the use of the term “ontology”. This term has been widely exploited in philosophy, in which it refers to the subject of existence, but in AI it has been often used in the context of knowledge sharing for indicating “a specification of a conceptualization” [22]. That is, an ontology is intended as a description of the concepts and relationships that can exist for an agent or a community of agents. As an example, in [22], it is observed that, when the knowledge of a domain is expressed by a declarative formalism, the set of objects that can be represented, and the describable relationships among them, are reflected in the representational vocabulary with which a knowledge-based program represents knowledge. Thus, in the context of AI, we can describe the ontology of a program by defining a set of representational terms. Ontological definitions associate the names of entities appearing in the program (e.g., classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms. Therefore, from a formal viewpoint, an ontology is the statement of a logical theory. This definition has been extended in [23], where it is observed that, in AI, an ontology is intended to be an engineering artifact, constituted by a set of terms, called vocabulary, used to describe an agent reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words. This set of assumptions has usually the form of a first-order logical theory, where vocabulary words appear as unary or binary predicate names, respectively called concepts and relations. Therefore, in the simplest case, an ontology describes a hierarchy of concepts related by subsumption relationships; in more sophisticated cases, suitable axioms are added in order to express other relationships between concepts and to constrain their intended interpretation. So two ontologies can be different in the vocabulary used while sharing the same conceptualization.

However, as noticed in [3,53], in Cooperative Agent Systems, ontologies are
often considered as collection of conceptual schemas representing the different
terms that can compose an agent message. As an example, in the well-known
JAva DEvelopment Framework (JADE) [32], a middleware that is fully com-
pliant with the standard defined by FIPA [15], an ontology is an extension of
the basic class ontology, that can contain different categories of elements as
predicates, terms, concepts, actions etc. In this sense, the basic class ontology
defines an ontology model, and the user can define its personal ontology by
adding to it its own predicates, terms, concepts and so on. In this paper we
use the term “ontology” as in the JADE framework. However, it is worth to
point out that we do not claim that the term “ontology” can be considered, in
any cases, as a synonym of “schema”, but only that we adopt the particular
viewpoint of most of the cooperative information agent systems in using this
term. Another point to highlight is that, in the User Modelling, the internal
representation of both the interest and the behaviour of the human owner of
an agent is usually called the (user) model or more simply the model. In our
paper we deal with agent representations that can be viewed as user models,
and on which we apply the same processes applied to ontologies (e.g. they are
checked for similarities and exploited accordingly). However, it is important to
distinguish between the ontology (that represents the kernel of a knowledge:
key concepts, properties and relations) and the framework (application of an
ontology to a domain of knowledge or expertise in order to represent it for
some machine readable purpose). In this paper we use the term ontology in
the sense of the knowledge kernel, representing the personal “viewpoint” of
the agent on the entire world it explores. In this sense an ontology can be
applied to different frameworks in which the agent occasionally operates.

In this context, we consider as ontology models those languages that have been
developed for representing the semantic of Web resources, and that show the
capability of modeling semi-structured data. This is the case of XML (eXten-
sible Markup Language)[56], that is a markup language standard developed
by the W3C (World-Wide Web Consortium) for facilitating information ex-
change. Another well-known standard is OML (Ontology Markup Language)
[41], that is an ontology specification language based on Conceptual Graphs
[8]. It allows the representation of concepts organized in taxonomies, rela-
tions and axioms in first order logic. Another language is DAML+OIL [9],
that provides modelling primitives commonly found in frame-based languages
(such as an asserted subsumption hierarchy and the description or definition of
classes). Instead, the Ontology Web Language (OWL) [42] has been conceived
for defining and instantiating Web ontologies. An OWL ontology may include
descriptions of classes, properties and their instances, and the OWL semantics
specifies how to derive logical consequences from a given ontology. Finally, the
Semantic Web Rule Language (SWRL) [54] is a combination of two OWL sub-
languages with the Unary/Binary Datalog RuleML sub-languages of the Rule
Markup Language. This approach extends the set of OWL axioms to include
Horn-like rules, that model causal implications. Thus, in SWRL it is possible to
combine Horn-like rules with an OWL knowledge base. Ontologies, that from an Information Agent viewpoint concerns intensional descriptions of interests and behaviour of the user, have been already exploited in Cooperative Information Agent Systems [39,40,7,45]. Moreover, many logic-based approaches for designing information agents have been proposed in the literature (see the survey [27]), demonstrating that Logic Programming (LP) and non-monotonic reasoning are appropriate for rational agents. As an example, the well known framework of the BDI logic [47] is used for modeling information about the state of the agent environment, objectives to be accomplished and currently chosen course of action, as in the LORA system [57]. Multi-Dimensional Dynamic Logic Programming (MDLP) is used in the system MINERVA [2], an architecture to represent the epistemic states of agents.

Our approach, in order to represent agent ontologies, uses the model presented in [48,49], that is here called Information Agent Ontology Model (IAOM). This model explicitly represents each object and each group of objects contained in the environment of an agent, using a unique “name” of a common vocabulary. We follow an Object-Oriented approach similar to that used in the well-known JADE Reference Model [32]. In the JADE framework, the “ontology model” is a class containing some basic schemas common to all the ontologies, and an “ontology” is an extension of this basic class, to which the schemas defining the structure of the types of predicates, agent actions and concepts relevant to the addressed domain have been added. Following this approach, we denote as “object” an actual object of the agent’s world and we define the “object-schema” of an object \( o \) as the set of properties of \( o \), where a property of \( o \) is another object that represents a particular “characteristic” of \( o \). Our concept of “object-schema” is thus similar to that of “class” in the Object Oriented Programming (OOP), but we introduce a difference with OOP relative to the completeness of the objects’ description. The difference is that, in our approach, an object-schema is not a “fixed” schema but, similarly to semi-structured formalisms as XML, it describes only partially the structure of its objects. However, although derived from XML, the idea of using an object-oriented approach for handling semi-structured data is not novel in the OOP and in Object-Oriented Databases communities, since it has been already proposed, as an example, in the XOM language [59]. Besides objects, we introduce the notion of “collection” for representing a group of objects, each possibly having a different schema, and organized in sub-collections. Furthermore, in order to model the events that can happen in the agent reality, as well as causal implications among events, we introduce in the ontology a set of propositional clauses, that form a logic program. Since the situations that may happen in agent realities can contain both classical and default negation, we choose to deal with the framework of extended logic programming. Our model also represents functions capable of performing operations on objects and collections. Finally, the proposed ontology is capable of modeling the behaviour of an agent, by means of relationships between events and functions,
that we call “actions”. IAOM, similarly to XML, allows to deal with semi-structured data but, differently from this standard, that model only objects’ structure, IAOM also provides the possibility to model causal implications by means of logic formalisms. Like OML, OML+OIL, OWL and SWRL, IAOM includes logic axioms: however, IAOM introduces the additional possibility to model agent actions, distinguishing actions from causal implications. In the IAOM framework, a causal implication is a logical relationship between two events, whereas an action is a relationship between an event and a function (i.e., a computer program), that has some effects on the state of other objects. The example depicted in Figure 1 shows that in the IAOM model causal implications are represented by the arcs of a neural-symbolic network, while an action is represented by an action arc. In the aforementioned classical frameworks there is not a direct way for expressing an action, but it is necessary to combine different constructs for simulating its effects. Moreover, the IAOM model is directly implementable by a neural-symbolic network, that is a machine learning mechanism suitable to realize an inductive process: none of the classical ontology models has been designed for directly supporting such a task. Another important difference is relative to BDI logics. BDI is individualistic, where the mutual monitoring that we propose is social and therefore we may centralize the computation for similarities and avoid the complexity explosion. The price we pay is that we have to be sure that our built ontologies indeed represent their users, what is the most difficult issue to prove. Finally IAOM, differently from the ontology models described in Section 2 that are implemented languages, is only a mathematical model: its implementation could be realized by using any of the above standards, and we have actually implemented it in the JADE framework.

If, on one hand, logic programming formalisms are necessary for representing agent behaviour, on the other hand logic programs have to be learnt from examples in order to realize autonomous agents. Three main categories of methods exist in the literature dealing with this issue, namely symbolic methods, statistical methods and connectionist methods. Symbolic methods aim at performing learning in a symbolic framework: this is the case of Inductive Logic Programming (ILP). The goal of an ILP method is the reverse of logical deduction, that is, it consists in finding, given some background knowledge \( B \), and a set of examples \( E \), a set of hypothesis \( H \), such that \( B \cup H \) entails \( E \). Most ILP methods can be classified into one of two categories: top-down or bottom-up. Top-down methods, exemplified by FOIL [46], begin with an initially empty hypothesis, which is grown by adding clauses that cover positive examples. The clauses are specialized by adding antecedents such that they cover the maximum number of positive examples while misclassifying a minimum number of negative ones. This process is repeated until the hypothesis completely covers the training set. Instead, bottom-up approaches try to form hypothesis by repeating generalization of the examples. Well known examples of statistical methods are the Probabilistic Relational Models (PRMs) [34],
that are an extension of Bayesian networks to relational domains. A PRM describes a template for a probability distribution over a database, including a relational component, that describes the relational schema for the domain, and a probabilistic component, that describes the probabilistic dependencies that hold in the domain. Among connectionist methods, Neural Networks [25] are a collection of processing units and adaptive connections that are designed to perform a specific processing function. The main features include a set of many simple processors linked by adaptive connections. These simple processors sum up their inputs and calculate an output value, or activation. This output is then sent to other processing units in the neural network. The connections are adaptive since they are adjusted during a training phase. The IACOM model, that we use as ontology learning model in CILIOS architecture, introduces the advantage, with respect to the ontology representation described in [45], to give the possibility of learning an ontology by directly observing the user behaviour. With respect to approaches that present learning capabilities, IACOM introduces the important feature of learning causal implications that allow both classical and default negation, very important in modelling agent realities. This latter feature is not new in the Knowledge Representation domain, being introduced in [11], however it is not present in the agent systems previously proposed in the literature that also support learning as, for instance, those described in [7,50].

3 The CILIOS Architecture

In this Section, we describe the CILIOS Multi-Agent Architecture. This architecture fully complies with FIPA specifications [15], and it is conceived as an extension of the JADE platform. JADE is a software framework aiming at supporting the implementation of agent-based applications according to FIPA specifications. As shown in Figure 2, such an architecture is composed by four levels of agents, each of them involving a different agent typology. The first level is called Main, and contains only three agents, namely the Agent Management System (AMS), the Directory Facilitator (DF) and the Agent User Interface (GUI). The AMS manages the whole platform, by controlling the agents’ accesses and providing a white page service. The DF provides a yellow pages service, i.e. it is able to give information about the various services provided by each agent of the platform. The GUI allows the human user to interact with the agent platform for requiring services. Agents belonging to the second level, called IACOM agents, are inductive agents, and each of them is associated with a human user. The goal of an IACOM agent is to construct an IACOM (connectionist) representation of the associated user’s interests and behaviour. As it is described in Section 6, such an IACOM representation is based on a neural-symbolic network, able to model
the agents’ functions (i.e., the programs that the agent performs) and the logic program that represents causal implications among events happening in the agent’s environment. This neural-symbolic network is constructed by each IACOM agent on the basis of a background knowledge provided by the associated human user and by observing the actual behaviour of the user (thus, the two arrows “background knowledge” and “observed behaviour” comes from the human user). Figure 2 shows that a refined knowledge is constructed by modifying the background knowledge on the basis of the observed behaviour. Agents in the third level are called IAOM agents, since each of them, associated with an agent of the IACOM level, contains an IAOM (symbolic) representation equivalent to the connectionist one of the associated IACOM agent. Section 4 describes the IAOM ontology model into detail. An IAOM agent contains an ontology translator that acts as a function receiving a IACOM ontology representation as input and yields as output the equivalent IAOM ontology representation.

Agents in the fourth level are called Ontology Similarities Managers (OSMs). Each OSM is associated with an agent of the IAOM level, and computes the similarity between the IAOM ontology of this agent and the IAOM ontology of each other agent in the platform.

Each agent typology is implemented as an extension of the JADE class Agent. Each agent level is implemented by an Agent Container, a data structure provided by JADE for collecting agent units having homogeneous behaviour. As described previously, the human user exploits a GUI, that belongs to the main level, that allows him to interact with AMS and DF agents, for requiring services. Generally, these services involve cooperation that in our case is provided by the OSM agents, as it refers only to similarities. The OSM level exploits, in its turn, the symbolic ontologies stored in the IAOM level, that have been constructed by the inductive agents belonging to the IACOM level. In Sections 4 and 5 we describe in details the main concepts involving IAOM and OSM level, respectively, while in Section 6 we deal with the concepts relative to the IACOM level.

4 Agent Ontology Model: IAOM level

In this section, we introduce a formal definition of agent ontology, that will allow us to face, in the next sections, all the issues relative to ontologies described in Section 1 by using a unique formalism. The necessity to introduce a new formalism, instead of exploiting an existing standard as XML, OWL etc., is due to the fact that none of these standards covers alone all of semi-structured data, classes’ description, collections, causal implications and actions. In order to better explain the various concepts we introduce below, consider the simple situation of a customer, called John, performing e-commerce activities. We
can observe that different kinds of knowledge need to be represented.

**Objects, Collections and Functions.** First of all, the ontology has to represent some products relative to the agent’s owner. Each product must have a set of associated properties that gives some information about it, as the price and the delivery time. Note that all AI representations including ontologies, have the purpose of representing intensions (instead of extensions, such as in Information Systems). Also in our approach, the representations of objects are intensional information that we call schemas. A schema may thus be a primitive data type (e.g. an integer), or it may have a more complex structure. Moreover, schemas are semi-structured, in the sense that each property can appear in actual objects with an arbitrary multiplicity. Each actual object (e.g. a book) can be viewed as an instance of a schema and assumes a state by assigning legal values to its properties. We admit that, for each property of an object schema, a set of instances of that property may appear into an object of the schema, and we associate two integer variables with the property for representing the maximum and the minimum cardinality of the property instances set. We introduce the constant $MUL$ for representing the maximum value admissible for the maximum cardinality. We call required a property that has both minimum and maximum cardinality equal to 1 and optional a property having minimum (resp. maximum) cardinality equal to 0 (resp. 1). We use a set $BS$ of primitive data types for representing integer, real, boolean and string values, as well as the data types void that represents a null value,
propositional and clause that represent a propositional variable and a logic clause, respectively. The variable price, that defines the price of a book, is an example of object of the basic object schema real.

**Definition 1** An object schema \( s \) is either: (i) a basic object schema or (ii) a set \( \{p_1, p_2, \ldots, p_n\} \), where each \( p_i \), \( i = 1..n \), that we call property of \( s \), is a triplet \((p, \text{minCard}, \text{maxCard})\) such that \( p \) is an object schema and both \( \text{minCard} \) and \( \text{maxCard} \) are integer objects with \( 0 \leq \text{minCard} \leq \text{maxCard} \).

**Definition 2** Let \( s = \{(p_1, m_1, M_1), (p_2, m_2, M_2), \ldots, (p_n, m_n, M_n)\} \) be an object schema. An object of \( s \) is a set \( o_s = \{(p_1^1, p_1^2, \ldots, p_1^n), (p_2^1, \ldots, p_2^n), \ldots, (p_n^1, \ldots, p_n^n)\} \), where \( p_i^1, p_i^2, \ldots, p_i^n \) are objects of \( p_i \), \( i = 1, \ldots, n \), and \( m_i \leq c_i \leq M_i \).

As an example of object schema that contains only one required basic property, consider \( \text{title} = \{\text{string}, 1, 1\} \). Instead, the schema \( \text{author} = \{\text{firstName}, 1, 1\} \) has two required, non basic, properties. Finally, the schema \( \text{book} = \{\text{author}, 1, \text{MUL}, \text{title}, 1, 1\}, \text{editor}, 0, \text{MUL}, \text{price}, 0, 1) \) has both required, optional, zero-or-more and one-or-more properties.

Furthermore, an ontology should be capable of representing a collection of objects, as the products purchased by John, that may be composed, in its turn, by other sub-collections as a collection of books and a collection of CDs. For this purpose, we define the notions of collection schema and collection.

**Definition 3** A collection schema \( cs \) is either an empty set or a set \( \{c_1, c_2, \ldots, c_n\} \), where \( c_1, c_2, \ldots, c_n \) are collection schemas.

The recursive definition shows that a collection schema can be viewed as a hierarchical organization of sub-collection names, like the directory organization of a file system.

A collection having a collection schema \( cs \) is an instance of \( cs \). More formally:

**Definition 4** A collection \( C \) having the collection schema \( cs = \{c_1, c_2, \ldots, c_n\} \) is a set \( \{C_1, C_2, \ldots, C_n, o_1, o_2, \ldots, o_m\} \), where \( C_1, C_2, \ldots, C_n \) are collections having collection schema \( c_1, c_2, \ldots, c_n \), respectively, and \( o_1, o_2, \ldots, o_m \) are objects.

In the context of an agent ontology, we have the necessity to use functions that operate on objects and collections. To this purpose, we define the notion of function schema, as a triplet that contains a set of object schemas and collection schemas of input, a set of object schemas and collection schemas of output, and a returned schema. Each actual function can be viewed as an instance of a function schema. More formally:

**Definition 5** A function schema \( fs \) is a triplet \((\text{input}, \text{output}, \text{returned})\), where (i) input (resp. output) is a set \( \{oi_1, oi_2, \ldots, oi_m, ci_1, ci_2, \ldots, ci_n\} \) (resp.
\{oo_1, oo_2, ..., oo_h, co_1, co_2, ..., co_k\} \) such that \( o_{i_l}, l = 1, ..., m \) (resp. \( oo_l, l = 1, ..., h \)) is an object schema and \( ci_r, r = 1, ..., n \) (resp. \( co_r, r = 1, ..., k \)) is a collection schema, and (iii) \( returned \) is an object schema or a collection schema. \( \square \)

**Definition 6** A function with schema \( fs = (input, output, returned) \) where \( input = \{osi_1, osi_2, ..., osi_m, csi_1, csi_2, ..., csi_n\} \), \( output = \{oso_1, oso_2, ..., oso_h, cso_1, cso_2, ..., cso_k\} \) and \( returned \) is an object (resp. a collection) schema, is a 4-tuple \( f = (i, o, M, r) \), where \( i = \{o_{i_1}, o_{i_2}, ..., o_{i_m}, c_{i_1}, c_{i_2}, ..., c_{i_n}\} \), \( o = \{oo_{o_1}, oo_{o_2}, ..., oo_{o_h}, co_{1}, co_{2}, ..., co_{k}\} \) such that \( o_{i_l}, l = 1, m \) (resp. \( oo_1, l = 1, h \)) is an object having schema \( osi_l \) (resp. \( oso_l \)), \( c_{i_l}, l = 1, n \) (resp. \( co_l, l = 1, k \)) is a collection having schema \( csi_l \) (resp. \( cso_l \)), \( M \) is an algorithm operating on the set of input objects and collections, the set of output objects and collections and the output object (or collection), producing the output values in correspondence of each input values’ configuration, as well as the value of the result \( r \). \( \square \)

As an example, in order to represent the action performed by the customer \( John \) when, in the context of a negotiation with a merchant, he reacts to a new \( proposed\_price \) of the merchant by making a new \( offer \), we may use the function schema \( new\_off = (\{real, real\}, \{real\}, void) \), and implement it by a function \( John\_new\_off = (\{proposed\_price, offer\}, \{offer\}, M, void) \), where \( proposed\_price \) is the input and \( offer \) is the output object, \( M \) represents the negotiation behaviour of \( John \), and the returned object is equal to \( void \) because this function only modifies the state of the object \( offer \) without returning any value. \( M \), in this case, is a simple algorithm that operates as follows: it sets the value of \( offer \) to the mean between the previous value of \( offer \) and the value of \( proposed\_price \).

**Events and Causal Implications.** In the reality of an agent, there exist some situations that may happen or not. In order to represent such situations, we use propositional objects, that we call events, following [35]. For instance, suppose \( John \) has the possibility to decide to make or not an offer for the product that is currently evaluating. We represent such a situation with the event \( makeO \). Often, we want to represent an event that is the **negation** of another event. We represent this kind of situation by an event \( \neg e \) that is the **classical negation** of \( e \). If \( e \) is an event, the classical negation of \( e \), denoted by \( \neg e \), means that the negation of \( e \) happens. However, in other cases, we want to represent the fact that there is no evidence that an event happens. To this purpose, we use the **default negation**. If \( e \) is an event, the default negation of \( e \), denoted by \( \sim e \), means that \( e \) is assumed to be \( false \) by default. Extended Logic Programming [20] is a formalism to represent logical rules that deal with both classical and default negation. Moreover, there exist some relationships between events that can be represented by logical rules. For instance, suppose to represent the following situation: \( John \) makes an offer for the current book if both he considers interesting the book and the price proposed by the seller is not too high. \( John \) also considers not too high the
price if this latter is smaller than all the other prices proposed for the same book by the other merchants that he knows. We can model such a situation by the rule: \( \text{makeOff} \leftarrow \text{intBook}, \sim \text{tooHigh} \); If any other merchants exist that sell the book, following this rule \text{John} would consider \text{false} by default \text{tooHigh}, then he would make the offer. However, we could have the necessity to model a situation in which \text{John} makes an offer only if it is sure that the price is not too high, in the sense that he has explicitly verified that other merchants exist and none of them has made a smaller price. In this case we have to use the rule \( \text{makeOff} \leftarrow \text{intBook}, \neg \text{tooHigh} \), where the strong negation \( \neg \text{tooHigh} \) is necessary for avoiding that \text{John} makes an offer if he is not able to explicitly prove that the price is not too high. It is worth pointing out that the rules representing causal implications are intrinsically dynamic, in the sense that they change with time. It is thus important to have a mechanism for inducing these rules by observing the user behaviour, as we will see in the following.

**Definition 7** A clause \( k \) is a triple \((\text{head}, \text{body}, \sim \text{body})\), where \text{head} is an event and \text{body} = \{a_1, a_2, ..., a_n\} and \( \sim \text{body} = \{a_{n+1}, \sim a_{n+2}, ..., \sim a_m\} \) are two sets of events. Such a clause represents a propositional rule of the form: \( \text{head} \leftarrow a_1, a_2, ..., a_n, \sim a_{n+1}, \sim a_{n+2}, ..., \sim a_m \). This notation means that, if both \( a_1, a_2, ..., a_n \) are proved to be \text{true}, and \( a_{n+1}, a_{n+2}, ..., a_m \) are assumed to be \text{false} by default, then \text{head} is to be proved. 

In our approach, we consider a clause as an object belonging to the schema \text{clause}. Generally, in representing an agent ontology, we have to consider a (finite) set of clauses, that form an \text{extended logic program}. The semantics of extended programs is an extension of the stable model semantics [5] and it is represented by the \text{answer sets} [20]. Generally, in representing an agent ontology, we have to consider a (finite) set of clauses. These clauses form an \text{extended logic program}. For understanding the semantics of such a kind of program, firstly consider the case of a program formed by clauses that do not have any classical negative event, i.e., a program with only default negation. This program is called a \text{general logic program}.

**Definition 8** Let \( \mathcal{P} \) a general logic program. We call \text{Herbrand base} of \( \mathcal{P} \), denoted by \( \mathcal{B}_\mathcal{P} \), the set of events occurring in \( \mathcal{P} \). An \text{interpretation} of \( \mathcal{P} \) is a function mapping each event in \( \mathcal{B}_\mathcal{P} \) to \{true, false\}. A \text{Herbrand Model} for \( \mathcal{P} \) is an interpretation making true all the clauses of the program. A Herbrand model \( \mathcal{M} \) of \( \mathcal{P} \) is minimal if no proper subset of \( \mathcal{M} \) is a Herbrand model of \( \mathcal{P} \).

**Definition 9** Let \( \mathcal{P} \) be a general logic program. The \text{Immediate Consequence Operator} is the mapping \( T_\mathcal{P}:2^\mathcal{B}_\mathcal{P} \leftarrow 2^\mathcal{B}_\mathcal{P} \) defined as follows. Let \( \mathcal{I} \) an interpretation of \( \mathcal{P} \), then \( T_\mathcal{P}(\mathcal{I}) = \{ e \in \mathcal{B}_\mathcal{P} | e \leftarrow e_1, ..., e_n \ \text{is a clause of} \ \mathcal{P} \ \text{and} \ \{e_1, ..., e_n\} \subseteq \mathcal{I} \} \), where \( \sim e_j \) is mapped to false (resp. true) by \( \mathcal{I} \) iff \( e_j \) is
Definition 10 Let $\mathcal{T}$ be a mapping and let $\alpha$ be an ordinal successor. We define $\mathcal{T} \uparrow \alpha = \mathcal{T}(\mathcal{T} \uparrow (\alpha - 1))$. Let $\mathcal{P}$ be a general logic program. Assuming that $\mathcal{B}_\mathcal{P}$ is finite, there is some $n \in \omega$ ($\omega = \{1, 2, \ldots\}$), such that $\mathcal{T}_\mathcal{P} \uparrow n = \mathcal{T}_\mathcal{P} \uparrow (n + 1)$. We define the Least Fixpoint $\mathcal{T}_\mathcal{P} \uparrow \omega$ of $\mathcal{P}$ as $\mathcal{T}_\mathcal{P} \uparrow n$.

A general program that does not contain negative events $\sim e$ has exactly one minimal Herbrand model, that is identical to the least fixpoint $\mathcal{T}_\mathcal{P} \uparrow \omega$. Programs with negative events may have several Herbrand models. In [19], the definition of stable model is introduced for providing a semantic for a general logic program. The intuition behind this definition is as follows: consider a rational agent with a set of premises $\mathcal{P}$. We search for sets of interpretation of $\mathcal{P}$ that can be considered as sets of belief that the rational agent might hold. To this purpose, consider an interpretation $\mathcal{I}$. Then, any clause of $\mathcal{P}$ that has an event $\sim e$ in its body, when $e \in \mathcal{I}$, is useless, and may be removed from $\mathcal{P}$. Moreover, any event $\sim e$, when $e \notin \mathcal{I}$, is trivial, and may be deleted from the clauses in which it appears in $\mathcal{P}$. This yields a simplified (positive) program $\mathcal{P}_\mathcal{I}$ and if $\mathcal{I}$ happens to be precisely the set of events that follows logically from $\mathcal{P}_\mathcal{I}$, then the set $\mathcal{I}$ is stable and represents a set of belief for the rational agent. More formally:

Definition 11 Let $\mathcal{P}$ be a general logic program. Given a set $\mathcal{I}$ of events from $\mathcal{P}$, let $\mathcal{P}_\mathcal{I}$ be the (positive) program obtained from $\mathcal{P}$ by deleting (i) each clause that has a negative event $\sim e$ in its body, with $e \in \mathcal{I}$, and (ii) the negative events in the bodies of the remaining clauses.

The program $\mathcal{P}_\mathcal{I}$ is positive (i.e., does not contain any negative event) and thus it has a unique minimal Herbrand Model $\mathcal{M}$ [14]. We say that $\mathcal{M}$ is a stable model if it coincides with $\mathcal{I}$.

Definition 12 Let $\mathcal{P}$ be a general logic program. A Herbrand interpretation $\mathcal{I}$ of $\mathcal{P}$ is called stable iff $\mathcal{T}_{\mathcal{P}_\mathcal{I}}(\mathcal{I}) = \mathcal{I}$.

A general program is well behaved if it has exactly one stable model. In a well behaved general program, an event $e$ is true or false, depending on whether $e$ belongs or not to the unique stable model of the program. Now, we characterize some general logic programs that are well-behaved, and that are called acceptable programs.

Definition 13 A level mapping for a program $\mathcal{P}$ is a mapping $||: \mathcal{B}_\mathcal{P} \rightarrow \mathcal{N}$ of events to natural numbers. We denote by $| e |$ the level of $e$, for $e \in \mathcal{B}_\mathcal{P}$, and $| \sim e | = | e |$.

Definition 14 Let $\mathcal{P}$ be a general logic program, $||$ a mapping for $\mathcal{P}$ and $\mathcal{I}$ a model for $\mathcal{P}$. $\mathcal{P}$ is called acceptable w.r.t. $||$ and $\mathcal{I}$ if, for every clause
\( e \leftarrow a_1, a_2, \ldots a_n \) in \( \mathcal{P} \), the following implication holds, for \( 1 \leq i \leq n \):
\[
if I \models \land_{j=1}^{i-1}a_j \implies | e \rangle | a_i |
\]

\( \mathcal{P} \) is called \textit{acceptable} if it is acceptable w.r.t. some level mapping and a model of \( \mathcal{P} \).

**Definition 15** For each acceptable general program \( \mathcal{P} \), the function \( T_\mathcal{P} \) has a unique fixpoint. The sequence of all \( T_\mathcal{P} \uparrow m(i), m \in \mathcal{N} \), converges to this fixpoint \( T_\mathcal{P} \uparrow \omega(i) \) (which is identical to the stable model of \( \mathcal{P} \)), for each \( i \in \mathcal{B}_\mathcal{P} \).

The semantics of extended programs is an extension of the stable model semantics [5] and is represented by the \textit{answer sets}. A \textit{well behaved} extended program has exactly one answer set, and an event \( e \) is \textit{true}, \textit{false} or \textit{unknown} depending on whether its answer set contain \( e \), \( \neg e \) or neither. If a program does not contain classical negation, then its answer sets are exactly the same as its stable models [20]. Formally:

**Definition 16** Let \( \mathcal{P} \) be an extended logic program. Given a set \( I \) of events from \( \mathcal{P} \), let \( \mathcal{P}_I \) the extended program obtained from \( \mathcal{P} \) by deleting (i) each clause that has a negative event \( \sim e \) in its body, with \( e \in I \), and (ii) the negative events in the bodies of the remaining clauses.

\( \mathcal{P}_I \) is called, after its inventors, the \textit{Gelfond-Lifschitz Reduction} of \( \mathcal{P} \) w.r.t. \( I \).

The answer set of \( \mathcal{P}_I \) is unique and it is defined as follows

**Definition 17** Let \( \mathcal{P} \) be an extended logic program and let \( \mathcal{P}_I \) be the Gelfond-Lifschitz Reduction of \( \mathcal{P} \) w.r.t. the event set \( I \). The \textit{answer set} of \( \mathcal{P}_I \) is the smallest subset of \( \mathcal{B}_\mathcal{P} \) such that: (i) for any clause \( b \leftarrow a_1, a_2, \ldots, a_n \) of \( \mathcal{P}_I \), if \( a_1, \ldots, a_n \in \mathcal{P}_I \) then \( b \in \mathcal{P}_I \), and (ii) if \( \mathcal{P}_I \) contains a pair of complementary events then \( \mathcal{P}_I = \mathcal{B}_\mathcal{P} \).

Now, we define the answer set of an extended logic program \( \mathcal{P} \) as follows.

**Definition 18** Let \( \mathcal{P} \) be an extended logic program and let \( \mathcal{P}_I \) be the Gelfond-Lifschitz Reduction of \( \mathcal{P} \) w.r.t. the event set \( I \). Let \( I^+ \) the answer set of \( \mathcal{P}_I \). \( I \) is an answer set of \( \mathcal{P} \) iff \( I = I^+ \).

If an extended logic program does not contain classical negation, it is a general logic program. In this case, its answer sets are identical to its stable models. However, it is worth pointing out that the absence of an event \( e \) in the stable model of a general program means that \( e \) is \textit{false} (by default), while the absence of both \( e \) and \( \neg e \) in the answer set of an extended program means that we do not know nothing about \( e \). Some extended logic programs can be easily reduced to general logic programs. Consider an extended logic program
$\mathcal{P}$, and an event set $\mathcal{I}$ of this program. For each negative event $\neg e \in \mathcal{I}$, consider a positive event $e^p$ that does not occur in $\mathcal{P}$. $e^p$ is called the positive form of $e$. We call positive form of $\mathcal{I}$, denoted by $\mathcal{I}^*$, the set of the positive forms of all the events of $\mathcal{I}$, and we call the positive form of the program $\mathcal{P}$ the program obtained from $\mathcal{P}$ by replacing all the negative events $\neg e$ of each clause of $\mathcal{I}$ by its positive form. Then we define the notion of consistent answer set.

**Definition 19** Let $\mathcal{P}$ be an extended logic program and let $\mathcal{I}$ a consistent event set of $\mathcal{P}$, i.e., an event set that does not contain any contradiction. $\mathcal{I}$ is an answer set of $\mathcal{P}$ iff $\mathcal{I}^*$ is a stable model of $\mathcal{P}^*$. □

**Definition 20** An extended logic program $\mathcal{P}$ is called acceptable if its positive form $\mathcal{P}^*$ is acceptable. □

Acceptable extended programs[16] have a unique answer set.

**Example 1** As an example, consider the program in the John’s ontology:

- $k_1 : makeO \leftarrow \text{interestingBook}, \neg \text{tooHigh}$
- $k_2 : \neg \text{tooHigh} \leftarrow \text{interestingBook}, \sim \sim \text{smallPrice}$
- $k_3 : \text{interestingBook} \leftarrow \sim \sim \text{loveStory}$

This program is acceptable, and its unique answer set is \{makeO, interestingBook, $\neg$tooHigh\}. □

**Actions.** Often, when an event happens in the world of a customer or a seller, an action is consequently produced. For instance suppose that, when John decides to make an offer in a negotiation, the value of the offer is equal to the mean between his previous offer and the price proposed at the present by the seller. We can thus say that, when the event makeO is true, the function John\_new\_offer described above is called. We call action a 5-tuple composed by an event, as makeO, a function, as John\_new\_offer, that is activated by the event, a set of objects, as $\{\text{proposed\_price}\}$, that are the arguments of the function, and another two sets of objects and events, as $\{\text{offer}\}$ and $\{\text{end}\}$, respectively, whose state is modified by the function.

**Definition 21** An action is a set $\{e, f, os_1, es, os_2\}$ where $e$ is an event, $f$ is a function, $os_1, os_2$ are object sets, $es$ is an events set, such that the function $f$ is activated if $e = \text{true}$ by passing it as input arguments the objects belonging to $os_1$ and $f$ modifies both the value of the objects belonging to $os_2$ and the events belonging to the $es$ and returns a value $f(os_1, es, os_2)$. □

**Ontologies.** The definitions presented above allow us to define a set of schemas that describes an agent reality. This set, together with the logic program and
the action set, defining the behaviour of the agent, represent the ontology $O$ of the agent.

**Definition 22** An ontology is a 6-tuple $\langle OS, CS, FS, P, A \rangle$, where: (i) $OS$ is a set of object schemas; (ii) $CS$ is a set of collection schemas; (iii) $FS$ is a set of function schemas; (iv) $\varepsilon$ is a set of events; (v) $P$ is an extended logic program; (vi) $A$ is a set of actions. \hfill $\Box$

5 Inter-Ontology Similarities: OSM level

5.1 Content-based Similarities

We define the terminological similarity $T_{s_1s_2}$ between two schemas as a real coefficient, belonging to $[0,1]$, that gives a measure of how much the names of $s_1$ and $s_2$ are synonyms. This coefficient can be derived by a standard thesaurus as, for instance, Wordnet. We assume that, for each property of $s_1$, there is at most only a property of $s_2$ with a non-zero terminological similarity.

We compute the similarity between two object schemas $s_1$ and $s_2$ recursively. If both $s_1$ and $s_2$ are basic schemas, we assign a similarity equal to 1 if the schemas are the same, 0 otherwise. If one of the schemas is basic and the other is non-basic, we assign 0 to the similarity. In the general case of two non-basic schemas, we compute the similarity as the product of the terminological similarity and the mean value of some terms, each term associated with a pair $(x, y)$ of properties, where $x$ (resp. $y$) is a property of $s_1$ (resp. $s_2$). Each term is computed as the product of three contributions, namely (i) the similarity between $x$ and $y$, (ii) the terminological similarity between $x$ and $y$, and (iii) a factor that takes into account both maximum and minimum cardinalities.

**Definition 23** Let $x = \{ (p_1^x, m_1^x, M_1^x), (p_2^x, m_2^x, M_2^x), \ldots, (p_n^x, m_n^x, M_n^x) \}$ and $y = \{ (p_1^y, m_1^y, M_1^y), (p_2^y, m_2^y, M_2^y), \ldots, (p_k^y, m_k^y, M_k^y) \}$ be two object schemas and let $l = \max(n, k)$. The similarity between $x$ and $y$ is defined as

$$SS_{xy} = \begin{cases} 1, & \text{if } x, y \in BS \text{ and } x = y \\ 0, & \text{if } (x, y \in BS \text{ and } x \neq y) \\ \text{or } (x \in BS \text{ and } y \notin BS) \\ \text{or } (y \in BS \text{ and } x \notin BS) \end{cases}$$

and

$$TS_{xy} \cdot \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} SS_{p_i^x p_j^y} \cdot \alpha_{p_i^x p_j^y} \cdot TS_{p_i^x p_j^y}, \text{ if } x, y \notin BS.$$
Let \( l \) be a subcollection of \( \alpha \). Finally, we define the similarity between two function schemas \( f_1 \) and \( f_2 \), as an example, consider the collections \( \text{books1} = \{ \text{author}, \text{title}, \text{price} \} \) and \( \text{books2} = \{ \text{author}, \text{title}, \text{price} \} \), that have similarity equal to 1, since they are composed by two basic properties with terminological similarity equal to 1 and with the same maximum and minimum cardinality. Moreover, \( \alpha_{\text{books1},\text{books2}} = 0.5 \) and \( \alpha_{\text{author},\text{author}} = 1.0 \). Also suppose that \( T_{\text{books1},\text{books2}} = 0.95 \). Thus, the similarity between these two schemas is \( 0.95 \cdot \frac{3}{5} \cdot (0.5 + 1) = 0.475 \).

We also define the similarity between two collections \( X \) and \( Y \). If both \( X \) and \( Y \) are collections that do not contain any subcollection, the similarity is equal to the terminological similarity. Otherwise, it is computed as the mean value of the similarities between each pair \((a, b)\) of subcollections, where \( a \) (resp. \( b \)) is a subcollection of \( X \) (resp. \( Y \)). We assume that for each subcollection of \( X \) there is at most only a subcollection of \( Y \) with a non-zero terminological similarity.

**Definition 24** Let \( X \) and \( Y \) be two collection schemas. The similarity between \( X \) and \( Y \) is defined as:

\[
SS_{XY} = \begin{cases} 
TS_{XY}, & \text{if } X = Y = \emptyset; \\
TS_{XY} \cdot \frac{1}{l} \sum_{i=1}^{n} \sum_{j=1}^{k} SS_{c_i^X, c_j^Y} \cdot TS_{c_i^X, c_j^Y}, & \text{if } X = \{c_1^X, c_2^X, ..., c_n^X\} \text{ and } Y = \{c_1^Y, c_2^Y, ..., c_k^Y\}.
\end{cases}
\]

where \( l = \max(n, k) \) \( \Box \)

For instance, consider the collections \( \text{books1} = \{ \text{narrative}, \text{poetry} \} \) and \( \text{books2} = \{ \text{romances}, \text{essays} \} \). We suppose that all the involved subcollections are empty schemas. Also suppose that \( T_{\text{books1},\text{books2}} = 0.95 \) and \( T_{\text{narrative},\text{romances}} = 0.66 \) is the only subcollection pair with non-zero terminological similarity. Thus, the similarity between \( \text{books1} \) and \( \text{books2} \) is \( 0.95 \cdot 0.66/2 = 0.32 \).

Finally, we define the similarity between two function schemas \( f_1 \) and \( f_2 \). In this case, if the cardinalities of the respective input and output sets are different, the similarity between \( f_1 \) and \( f_2 \) is assumed equal to 0, otherwise, it is computed as the mean value of all the similarities between each corresponding parameter.

**Definition 25** Let \( f_1 = (\text{input}_1, \text{output}_1, \text{returned}_1) \) and \( f_2 = (\text{input}_2, \text{output}_2, \text{returned}_2) \). 

\[
\alpha_{p_i^X, p_j^Y} = \begin{cases} 
1, & \text{if } m_i^x = m_j^y \text{ and } M_i^x = M_j^y; \\
0, & \text{if } m_i^x \neq m_j^y \text{ and } M_i^x \neq M_j^y; \\
0.5, & \text{if } (m_i^x = m_j^y \text{ and } M_i^x \neq M_j^y) \text{ or } (m_i^x \neq m_j^y \text{ and } M_i^x = M_j^y). 
\end{cases}
\]
returned\(_2\)) be two function schemas, where (i) \(\text{input}_1\) is a set \(\{oi^1_1, oi^1_2, \ldots, oi^1_m, ci^1_1, ci^1_2, \ldots, ci^1_n\}\) such that \(oi^1_1, oi^1_2, \ldots, oi^1_m\) are object schemas and \(ci^1_1, ci^1_2, \ldots, ci^1_n\) are collection schemas, (ii) \(\text{input}_2\) is a set \(\{oi^2_1, oi^2_2, \ldots, oi^2_m, ci^2_1, ci^2_2, \ldots, ci^2_n\}\) such that \(oi^2_1, oi^2_2, \ldots, oi^2_m\) are object schemas and \(ci^2_1, ci^2_2, \ldots, ci^2_n\) are collection schemas, (iii) \(\text{output}_1\) is a set \(\{oo^1_1, oo^1_2, \ldots, oo^1_s, co^1_1, co^1_2, \ldots, co^1_h\}\) such that \(oo^1_1, oo^1_2, \ldots, oo^1_s\) are object schemas and \(co^1_1, co^1_2, \ldots, co^1_h\) are collection schemas, (iv) \(\text{output}_2\) is a set \(\{oo^2_1, oo^2_2, \ldots, oo^2_s, co^2_1, co^2_2, \ldots, co^2_h\}\) such that \(oo^2_1, oo^2_2, \ldots, oo^2_s\) are object schemas and \(co^2_1, co^2_2, \ldots, co^2_h\) are collection schemas, (v) \(\text{returned}_1\) (resp. \(\text{returned}_2\)) is an object schema or a collection schema.

The similarity between \(f_1\) and \(f_2\) is computed as the mean of all the similarities between each pair of corresponding function parameter, that is: \(SS_{f_1f_2} = \frac{1}{z}(\sum_{i=1}^m SS_{oi^1_i, oo^2_i} + \sum_{i=1}^n SS_{ci^1_i, co^2_i})\), where \(z = m + n + h + k + 1\).

The definitions above allow us to compute the overall content-based similarity between two ontologies \(O_1\) and \(O_2\), as the mean of all the semantic similarities between each pair of object, collection and function schemas \((s_1, s_2)\), where \(s_1\) belongs to \(O_1\) and \(s_2\) belongs to \(O_2\).

**Definition 26** Let \(O_1\) and \(O_2\) be two ontologies. The content-based similarity between \(O_1\) and \(O_2\) is computed as follows:

\[
\text{CBS}_{O_1O_2} = \frac{\sum_{i=1}^{\lvert OS_1 \rvert} \sum_{j=1}^{\lvert OS_2 \rvert} SS_{oi^1_i, oo^2_j} + \sum_{i=1}^{\lvert CS_1 \rvert} \sum_{j=1}^{\lvert CS_2 \rvert} SS_{ci^1_i, co^2_j} + \sum_{i=1}^{\lvert FS_1 \rvert} \sum_{j=1}^{\lvert FS_2 \rvert} SS_{fs^1_i, fs^2_j}}{\lvert OS_1 \rvert \cdot \lvert OS_2 \rvert + \lvert CS_1 \rvert \cdot \lvert CS_2 \rvert + \lvert FS_1 \rvert \cdot \lvert FS_2 \rvert}
\]

where \(OS_1, CS_1\) and \(FS_1\) (resp. \(OS_2, CS_2\) and \(FS_2\)) are the object schema set, the collection schema set and the function schema set of \(O_1\) (resp. \(O_2\)).

### 5.2 Behaviour-based Similarities

Here we define the similarity between two ontologies \(O_1\) and \(O_2\) with respect to the behavioural component, represented by logic programs and actions. Firstly, we assume that there exists the possibility to determine if two events \(e_1 \in O_1\) and \(e_2 \in O_2\) are synonyms and, in the affirmative case, we choose to indicate with a unique common event \(e\) both \(e_1\) and \(e_2\). Then, if \(A_1\) and \(A_2\) are two event sets, we indicate with \(CE_{A_1, A_2}\) the set of events common to \(A_1\) and \(A_2\). Let \(P_1\) (resp. \(P_2\)) be the logic program of \(O_1\) (resp. \(O_2\)) and let \(AS^1\) (resp. \(AS^2\)) be the answer sets collection of \(P_1\) (resp. \(P_2\)). In order to compute the similarity \(S_{P_1P_2}\) between the two programs, first we determine, for each answer set \(AS^1_i\) of \(AS^1\), the similarity \(S_{AS^1_i AS^2_j}\) with each answer set \(AS^2_j\) of \(AS^2\). This similarity is computed as the ratio between (i) the number of events common to both \(AS^1_i\) and \(AS^2_j\), and (ii) the number of events present in the union \(AS^1_i \cup AS^2_j\) of the two involved answer sets. Finally, we compute
the program similarity \( S_{P_1,P_2} \) as the mean of all the \( S_{AS_i^1,AS_j^2} \), for each \( AS_i^1 \) (resp. \( AS_j^2 \)) belonging to \( AS_1^1 \) (resp. \( AS_2^2 \)).

**Definition 27** Let \( P_1 \) (resp. \( P_2 \)) be the logic program of \( O_1 \) (resp. \( O_2 \)). Let \( ASC_1^1 \) (resp. \( ASC_2^2 \)) be the answer sets collection of \( P_1 \) (resp. \( P_2 \)). Let \( S_{ASC_i^1,ASC_j^2} = |ASC_i^1 \cap ASC_j^2|/|ASC_i^1 \cup ASC_j^2| \) be the similarity between \( ASC_i^1 \in ASC_1^1 \) and \( ASC_j^2 \in ASC_2^2 \). We compute \( S_{P_1,P_2} \) as the mean of all the \( S_{ASC_i^1,ASC_j^2} \), for each \( ASC_i^1 \) (resp. \( ASC_j^2 \)) \( \in ASC_1^1 \) (resp. \( ASC_2^2 \)). □

As an example, consider these two programs:

\[
\begin{align*}
makeOff & \leftarrow intBook, \sim end & makeOff & \leftarrow intBook, \sim highPrice \sim end \\
P_1 : & \quad end \leftarrow makeOff & P_2 : & \quad end \leftarrow \sim makeOff \\
& \quad intBook \leftarrow & & \quad intBook \leftarrow \\
& \quad \sim highPrice \leftarrow & & \quad \sim highPrice \leftarrow
\end{align*}
\]

where \( P_1 \) means that (i) a customer makes an offer for a book if this is interesting and there is no evidence that the negotiation phase is ended and (ii) the negotiation ends if there is no evidence the customer makes an offer. \( P_2 \) corresponds instead to a customer that makes an offer for a book if this is interesting, the price is not high and the negotiation is not ended; moreover, the negotiation ends if the customer makes no longer an offer. The answer sets of \( P_1 \) are \( ASC_1^1 = \{intBook, makeOff, \sim highPrice\} \) and \( ASC_2^2 = \{intBook, end \sim highPrice\} \), while the unique answer set of \( P_2 \) are \( ASC_2^2 = \{intBook, \sim highPrice, makeOff\} \). \( ASC_1^1 \) has a similarity with \( ASC_2^2 \) equal to 1, while \( ASC_2^2 \) has a similarity with \( ASC_1^1 \) equal to \( 2/4 = 0.50 \), thus the overall similarity is \((1+0.50)/2 = 0.75\).

Now, we define the action similarity between two actions \( A_1 \) and \( A_2 \). It is computed as the mean of all the similarities between each element of \( A_1 \) and the corresponding element of \( A_2 \).

**Definition 28** Let \( A^1 = \{e^1, f^1, o^1_1, e^1, o^2_1\} \) and \( A^2 = \{e^2, f^2, o^1_2, e^2, o^2_2\} \) be two actions. We define \( S_{e^1,e^2} = \begin{cases} 1, & \text{if } e^1 = e^2; \\ 0, & \text{otherwise.} \end{cases} \)
\( S_{f^1,f^2} = \begin{cases} 1, & \text{if } f^1 = f^2; \\ 0, & \text{otherwise.} \end{cases} \)
\[ S_{o^1_1,o^2_2} = \begin{cases} 1, & \text{if } o^1_1 = o^2_2; \\ 0, & \text{otherwise.} \end{cases} \]
\[ S_{o^1_2,o^2_1} = \begin{cases} 1, & \text{if } o^2_1 = o^2_1; \\ 0, & \text{otherwise.} \end{cases} \]
\[ S_{e^1,e^2} = \begin{cases} 1, & \text{if } e^1 = e^2; \\ 0, & \text{otherwise.} \end{cases} \]

The overall similarity between \( A_1 \) and \( A_2 \) is computed as \( S_{A_1,A_2} = \frac{S_{e^1,e^2} + S_{f^1,f^2} + \sum_{i=1}^{\max(|o^1_1|,|o^1_2|)} (S_{o^1_i,o^2_i}) + \sum_{i=1}^{\max(|o^2_1|,|o^2_2|)} (S_{o^2_i,o^2_i}) + \sum_{i=1}^{\max(|e^1|,|e^2|)} (S_{e^1_i,e^2_i})}{2+\max(|o^1_1|,|o^1_2|)+\max(|o^2_1|,|o^2_2|)+\max(|e^1|,|e^2|)} \). □
The rationale underlying the formula above is that we consider the following contributes for computing the overall similarity:

1. The two similarities $S_{e_1,e_2}$ and $S_{f_1,f_2}$ between the corresponding events and the corresponding functions, respectively, of the two actions.

2. We choose to compute the similarity between the two set $o_1^1$ (resp. $o_2^1$, $e_1$) and $o_1^2$ (resp. $o_2^2$, $e_2$) by comparing each element $o_1^1_i$ (resp. $o_2^1_i$, $e_1_i$) of the set $o_1^1$ (resp. $o_2^1$, $e_1$) with all the elements $o_1^2_j$ (resp. $o_2^2_j$, $e_2_j$) of the set $o_1^2$ (resp. $o_2^2$, $e_2$), and thus selecting as the “most corresponding element” of $o_1^1_i$ (resp. $o_2^1_i$, $e_1_i$) that element $o_1^2_j$ (resp. $o_2^2_j$, $e_2_j$) having the maximum similarity with $o_1^1_i$ (resp. $o_2^1_i$, $e_1_i$).

The denominator of the formula contains the number of all the contributions.

If $AS^1$ and $AS^2$ are two sets of actions, we compute the similarity between $AS_1$ and $AS_2$ as the mean of all $S_{AS^1_i,AS^2_j}$, where $AS^1_i \in AS^1$ and $AS^2_j \in AS^2$.

**Definition 29** Let $AS^1$ and $AS^2$ be two sets of actions. The similarity between $AS_1$ and $AS_2$ is computed as $S_{AS^1_i,AS^2_j} = \sum_{i=1}^{\|AS^1\|} \sum_{j=1}^{\|AS^2\|} (S_{AS^1_i,AS^2_j}) / \|AS^1\| \times \|AS^2\|$.

Finally, we can define the overall behaviour-based similarity between two ontologies as the mean of the program similarity and the action similarity.

**Definition 30** Let $O_1$ and $O_2$ be two ontologies. The behaviour-based similarity between $O_1$ and $O_2$ is computed as follows: $BBS_{O_1,O_2} = \frac{SS_{P_1,P_2} + SA_{A_1,A_2}}{2}$ where $P_1$ and $A_1$ (resp. $P_2$ and $A_2$) are the logic program and the action set of $O_1$ (resp. $O_2$), respectively.

5.3 Complexity of Computing Content-Based Similarity

Figure 3-(A) represents an object schema $o$ by a rooted tree having the schema name as root and having an arc for each relationship between a property and a sub-property (recall that properties are schemas in its turn). We call structural depth $d_o$ of the schema $o$ the maximum length of a path in the associated tree. In other words, $d_o$ is the maximum number of nested sub-schemas existing between the schema $o$ and a basic schema component. We also call structural amplitude $a_o$ of the schema $o$ the maximum number of subschemas contained in a subschema of $o$ (including $o$). In the example of Figure 3-(A), $d_{Book} = 2$ and $a_{Book} = 3$. We can observe that the time cost for computing the semantic similarity between two object schemas $o_1$ and $o_2$ is strongly related to both (i) the minimum value between the structural depths $d_{o_1}$ and $d_{o_2}$ and (ii) the maximum value between the structural amplitudes $a_{o_1}$ and $a_{o_2}$. More formally:
Lemma 1 Let \( o_1 \) and \( o_2 \) be two object schemas and let \( d \) be the minimum value between \( d_{o_1} \) and \( d_{o_2} \), where \( d_{o_1} \) (resp. \( d_{o_2} \)) is the structural depth of \( o_1 \) (resp. \( o_2 \)). Moreover, let \( A \) be the maximum value between \( a_{o_1} \) and \( a_{o_2} \), where \( a_{o_1} \) (resp. \( a_{o_2} \)) is the structural amplitude of \( o_1 \) (resp. \( o_2 \)). Then, the cost of computing the similarity \( SS_{o_1o_2} \) is \( \mathcal{O}(A^d+1) \).

**Proof.** It directly derives from the Definition 23, by considering that each subschema \( s_{1i} \) of \( o_1 \), at a fixed depth level \( i (i = 0, 1, \ldots, h) \), has to be compared with each subschema \( s_{2i} \) of \( o_2 \) at the same level, noting that \( d \) is the maximum level for which the comparison can be performed. Then, since the number of schemas at the level \( i \) is \( \mathcal{O}(A) \) and that for each comparison between \( s_{1i} \) and \( s_{2i} \) we have to perform all the comparison of each nested subschema of \( s_{1i} \) with all the nested subschemas of \( s_{2i} \), until the maximum nested level that is \( \mathcal{O}(d) \), we derive that the overall cost is \( \mathcal{O}(A^d+1) \). □

An analogous result can be derived for the semantic similarity between two collection schemas. Figure 3-(B) represents a collection schema \( C \) by a rooted tree having the schema name as root and having an arc for each relationship between a collection schema and a sub-collection schema. We call structural depth \( d_C \) of the schema \( C \) the maximum length of a path in the associated tree. In brief, \( d_C \) is the maximum number of nested sub-schemas existing between the collection schema \( C \) and an empty collection schema component. We call structural amplitude of \( C \) the maximum number of subschemas contained in a subschema of \( C \) (including \( C \)). We can observe that the time cost for computing the semantic similarity between two collection schemas \( C_1 \) and \( C_2 \) is strongly related to both (i) the minimum value between the structural depths \( d_{C_1} \) and \( d_{C_2} \) and (ii) the maximum value between the structural amplitudes \( a_{C_1} \) and \( a_{C_2} \).

Lemma 2 Let \( C_1 \) and \( C_2 \) be two collection schemas and let \( d \) be the minimum value between \( d_{C_1} \) and \( d_{C_2} \), where \( d_{C_1} \) (resp. \( d_{C_2} \)) is the structural depth of \( C_1 \) (resp. \( C_2 \)). Moreover, let \( A \) be the maximum value between \( a_{C_1} \) and \( a_{C_2} \), where \( a_{C_1} \) (resp. \( a_{C_2} \)) is the structural amplitude of \( C_1 \) (resp. \( C_2 \)). Then, the computational complexity of computing the similarity \( SS_{C_1C_2} \) is \( \mathcal{O}(A^d+1) \).

**Proof.** It directly derives from the Definition 24, by means of considerations analogous to those of Lemma 1. □
The time cost for computing the similarity between two functions is formally derived below:

**Lemma 3** Let \( f_1 = (\text{input}_1, \text{output}_1, \text{returned}_1) \) and \( f_2 = (\text{input}_2, \text{output}_2, \text{returned}_2) \) be two function schemas, where (i) \( \text{input}_1 \) is a set \( \{o_1^1, o_2^1, ..., o_m^1, c_1^1, c_2^1, ..., c_n^1\} \) such that \( o_1^1, o_2^1, ..., o_m^1 \) are object schemas and \( c_1^1, c_2^1, ..., c_n^1 \) are collection schemas, (ii) \( \text{input}_2 \) is a set \( \{o_1^2, o_2^2, ..., o_m^2, c_1^2, c_2^2, ..., c_n^2\} \) such that \( o_1^2, o_2^2, ..., o_m^2 \) are object schemas and \( c_1^2, c_2^2, ..., c_n^2 \) are collection schemas, (iii) \( \text{output}_1 \) is a set \( \{oo_1^1, oo_2^1, ..., oo_1^k, co_1^1, co_2^1, ..., co_k^1\} \) such that \( oo_1^1, oo_2^1, ..., oo_1^k \) are object schemas and \( co_1^1, co_2^1, ..., co_k^1 \) are collection schemas, (iv) \( \text{output}_2 \) is a set \( \{oo_1^2, oo_2^2, ..., oo_1^h, co_1^2, co_2^2, ..., co_h^2\} \) such that \( oo_1^2, oo_2^2, ..., oo_1^h \) are object schemas and \( co_1^2, co_2^2, ..., co_h^2 \) are collection schemas, (v) \( \text{returned}_1 \) (resp. \( \text{returned}_2 \)) is an object schema or a collection schema.

The computational complexity of \( SS_{f_1f_2} \) is \( \mathcal{O}(z \cdot A^{D+1}) \), where \( z = m+n+h+k+1 \), \( A \) (resp. \( D \)) is the maximum structural amplitude (resp. the maximum structural depth) of the schemas contained in \( \text{input}_1 \cup \text{output}_1 \cup \text{returned}_1 \cup \text{input}_2 \cup \text{output}_2 \cup \text{returned}_2 \).

**Proof.** The result directly derives from the Definition 25, by considering that the cost for computing \( \sum_{i=1}^{m} SS_{o_i^1o_i^2} + \sum_{i=1}^{n} SS_{c_i^1c_i^2} + \sum_{i=1}^{h} SS_{oo_i^1oo_i^2} + \sum_{i=1}^{k} SS_{oo_i^1co_i^2} + SS_{\text{returned}_1\text{returned}_2} \) is upper-bounded by \( z \cdot A^{D+1} \), exploiting the results of both Lemma 1 and Lemma 2. \( \square \)

Now, we determine the time complexity for computing the overall content-based similarity between two ontologies, as follows:

**Theorem 1** Let \( O_1=(O_1, V, O_1, I) \) and \( O_2=(O_2, V, O_2, I) \) be two ontologies. The cost of computing the content-based similarity \( CBS_{O_1O_2} \) between \( O_1 \) and \( O_2 \) is \( \mathcal{O}(|OS_1| \cdot |OS_2| + |CS_1| \cdot |CS_2| + |FS_1| \cdot |FS_2| \cdot A^{D+1}) \), where:

(i) \( OS_1 \), \( CS_1 \) and \( FS_1 \) (resp. \( OS_2 \), \( CS_2 \) and \( FS_2 \)) are the object schema set, the collection schema set and the function schema set of \( O_1, V \) (resp. \( O_2, V \));

(ii) \( A \) (resp. \( D \)) is the maximum structural amplitude (resp. the maximum structural depth) of both the object schemas belonging either to \( OS_1 \) or \( OS_2 \) and collection schemas belonging either to \( CS_1 \) or \( CS_2 \).

**Proof.** For proving the theorem, simply consider that (i) the content-based similarity is
\[
CBS_{O_1O_2} = \frac{\sum_{i=1}^{OS_1} \sum_{j=1}^{OS_2} SS_{o_i^1o_j^2} + \sum_{i=1}^{CS_1} \sum_{j=1}^{CS_2} SS_{c_i^1c_j^2} + \sum_{i=1}^{FS_1} \sum_{j=1}^{FS_2} SS_{fo_i^1fo_j^2}}{|OS_1| \cdot |OS_2| + |CS_1| \cdot |CS_2| + |FS_1| \cdot |FS_2|}
\]
and (ii) the complexity of computing each object (resp. collection, function) semantic similarity in this formula is given by Lemma 1 (resp. Lemma 2, Lemma 3). \( \square \)
5.4 Complexity of Computing behaviour-based Similarity

In this section, we analyze the cost of computing the similarity between the programs of two ontologies. This cost is equal to the cost of comparing the answer sets of the associated programs, following the Definition 27. However, before performing this comparison, it is necessary to find the answer sets of the two programs. The cost required for finding the answer sets of a program $P$ having $m$ atoms and $n$ total occurrences of these atoms, is $O(m \cdot 2^n)$ (see [36]), thus it becomes very expensive in the case the programs do not have a very small size. Therefore, it is necessary to deal with programs having a particular structure, suitable to making more simple the answer sets computation than in the general case. For example, in the case the program is an acceptable program, the computation of its unique answer sets can be performed with a polynomial cost with respect to both the number of atoms and the number of clauses, as we will show in Section 6.4. This is the case of the logic programs learned by the IACOM level that we will introduce in the next section. However, the notion of inter-ontology similarity that we introduce here is independent from the assumption we have made on the logic program structure and therefore, in the following of this Section, we analyze the cost of comparing the answer sets of two extended logic programs without considering constraints on their structure, supposing that these answer sets have been computed in a previous phase. The time cost is determined as follows:

**Lemma 4** Let $P_1$ (resp. $P_2$) be the logic program of $O_1$ (resp. $O_2$). Let $ASC^1_1$ (resp. $ASC^2_2$) be the answer sets collection of $P_1$ (resp. $P_2$) and let $\varepsilon_1$ (resp. $\varepsilon_2$) be the set of the events of $O_1$ (resp. $O_2$). The cost of computing the semantic similarity $S_{P_1 P_2}$ is $O(|ASC^1_1| \cdot |ASC^2_2| \cdot |\varepsilon_1| \cdot |\varepsilon_2|)$

**Proof.** It directly derives from the Definition 24, by means of considerations analogous to those of Lemma 1. □

The complexity of computing the similarity between two actions sets is provided by the following lemma:

**Lemma 5** Let $AS^1$ and $AS^2$ be two sets of actions. The complexity of computing the similarity between $AS^1$ and $AS^2$ is $O(|AS^1| \cdot |AS^2| \cdot mo1 \cdot mo2 \cdot me)$ where $mo1$ (resp. $mo2$, $me$) is the maximum cardinality of the object input sets $o1$ (resp. object output sets $o2$, event sets $e$) of all the actions belonging either to $AS^1$ or to $AS^2$.

**Proof.** The result directly derives from the Definition 29, by considering that the cost for computing the similarity between two actions is $O(mo1 \cdot mo2 \cdot me)$. □

The results obtained in this subsection allow us to determine the time complexity of the overall behaviour-based similarity.
Theorem 2 Let $\mathcal{O}_1$ and $\mathcal{O}_2$ be two ontologies. The time cost of computing the behaviour-based similarity $BBS_{\mathcal{O}_1,\mathcal{O}_2}$ between $\mathcal{O}_1$ and $\mathcal{O}_2$ is $O(|ASC_1| \cdot |ASC_2| \cdot |\varepsilon_1| \cdot |\varepsilon_2| + |AS_1| \cdot |AS_2| \cdot mo_1 \cdot mo_2 \cdot me)$ where: (i) $P_1$ (resp. $P_2$) is the logic program of $\mathcal{O}_1$ (resp. $\mathcal{O}_2$); (ii) $ASC_1$ (resp. $ASC_2$) is the answer sets collection of $P_1$ (resp. $P_2$); (iii) $\varepsilon_1$ (resp. $\varepsilon_2$) is the set of the events of $\mathcal{O}_1$ (resp. $\mathcal{O}_2$); (iv) $mo_1$ (resp. $mo_2$, $me$) is the maximum cardinality of the input sets $o_1$ (resp. output sets $o_2$, event sets $e$) of all the actions belonging either to the event sets $AS_1$ of $\mathcal{O}_1$ or to $AS_2$ of $\mathcal{O}_2$.

Proof. The complexity is simply the sum of the results obtained in Lemma 4 and Lemma 5 □

The above analysis shows that the cost of computing the content-based similarity between two ontology linearly increases with respect to the size of each object schema set, collection schema set and function schema set and exponentially increases with respect to the maximum structural depth of both object and collection schemas, while this exponential growth has as base the maximum structural amplitude of both object and collection schemas. This leads to conclude that we have to avoid a too big structural complexity of schemas if we want to obtain an acceptable efficiency of the performances. Moreover, the cost of computing the behavior-based similarity linearly depends on the size of all the sets contained in the ontology (i.e., the answer set collections, the action sets and the event sets) as well as on the maximum number of parameters present in the functions belonging to the function sets. Besides of avoiding to use schemas having too big values of structural depth and amplitude, we argue that the linear dependence from all the remaining parameters could be too onerous in practical applications: the advantage that a multi-agent community can obtain from computing inter-ontology similarities could be annulled or even exceeded by the disadvantage of having too big cost for such a computation. Therefore, it is necessary to adopt a strategy of off-line similarities computation, also dedicating specialized agents in the community to perform this crucial task. These specialized agents in CILIOS platform form a computational grid, called OSM level, that periodically update the values of the similarities between agents of the community. As shown in Figure 2, an OSM agent is associated with each agent and computes all the similarities between such an agent and all the other agents. Therefore, when an agent in the community needs to know the similarity between itself and another agent, it has to simply query its OSM, with any additional cost with respect to those necessary for the current agent activity. However, this technique presents the disadvantage that an agent might query the OSM towards the end of the interval between two consecutive updates, i.e. when a new similarity update is to begin and the similarity values are the most obsolete as possible. The longer is this interval, the worse will be the performance of the similarity computing.
The idea of linking events and functions by means of actions, proposed in Section 4, allows us to use the ontology of an agent for simulating the agent behaviour. In order to explain our approach, consider the notion of ontology graph that we define as follow:

**Definition 31** Let \( \mathcal{O} \) be an agent ontology. The ontology graph associated to \( \mathcal{O} \) is the pair \((\mathcal{N}, \mathcal{E})\), where \( \mathcal{N} \) is a set of nodes and \( \mathcal{E} \) is a set of arcs. A node \( n \in \mathcal{N} \) is associated to (i) each event that appears either in the program \( \mathcal{P} \) or in a function \( f \) of \( \mathcal{O} \); (ii) each function \( f \) of \( \mathcal{O} \); (iii) each object that appears in a function \( f \) of \( \mathcal{O} \). An arc \( e = (a, b) \in \mathcal{E} \) is associated to (i) each pair \((a, b)\) of events such that there exists a clause of \( \mathcal{P} \) where \( a \) appears in the body and \( b \) appears in the head of the clause; (ii) each pair \((a, f)\) where \( f \) is a function of \( \mathcal{O} \) and \( a \) is an object such that there exists in \( \mathcal{O} \) an action \( a \) where \( a \) appears as an input object; (iii) each pair \((f, a)\) where \( f \) is a function of \( \mathcal{O} \) and \( a \) is an object such that there exists in \( \mathcal{O} \) an action \( a \) where \( a \) appears as an output object.

For instance, consider the ontology graph depicted in the example in Figure 4, relative to the John’s ontology. We introduced a (white) node for each event belonging to the set \( \varepsilon \) appearing in the program \( \mathcal{P} \) of the John’s ontology and a black node for the unique function of the set \( F \). When the agent starts, it firstly computes the answer set of the program \( \mathcal{P} \), that is \( \{\text{makeO, interestingBook, } \neg\text{tooHigh}\} \). The arc \((\text{makeO, John}_\text{new}_\text{offer})\), due to the action \( \text{a}_\text{John} \), activates the function \( \text{John}_\text{new}_\text{offer} \) since \( \text{makeO} \) is in the state true. Thus, the agent sets the state of the object offer on the base of the current state of the object proposed_price. The ontology graph of an agent shows that an ontology can be exploited for representing the behaviour of the agent’s owner. As discussed in the Section 1, the agent should be pro-active, thus it should be capable of learning the behaviour of its human owner by simply “watching over the shoulders” of the owner, i.e., by continuously monitoring owner’s actions. This means the agent should exhibit inductive capabilities, since it should be able of:

- (A) learning the mappings between objects as, for instance, the mapping between the objects proposed_price and offer in function \( \text{John}_\text{new}_\text{offer} \) of the Figure 4;
- (B) learning the causal implications between events as, for instance, the implications between the objects \( \text{makeO, interestingBook, } \neg\text{tooHigh, loveStory} \) and \( \neg\text{smallPrice} \) in the John’s ontology.

\(^3\) Recall that an event is an object, and thus it can appear as a parameter in a function.
Neural networks can be usefully exploited for solving both these tasks. In particular:

- neural networks are commonly used for learning mappings between concepts, so their application to solve problem (A) is straightforward.
- It has been proved ([11]) that neural networks can be used to learn clauses of extended general programs and that they are capable of computing the answer set of the program if this latter is acceptable (see Definition 20 for remembering the meanings of acceptable program): this allows one to apply neural networks to the problem (B).

In the next two sub-sections we give some technical details relative to two general methodologies covering the issues (A) and (B), respectively. Then, in the subsection 6.3 we explain our idea of defining a neural network model capable of learning and encoding an agent ontology, based on a combination of the aforementioned methodologies.

### 6.1 Neural networks that learn functions

A standard feedforward neural network is composed by a set of $N$ nodes $\mathcal{N}$ and a set of $M$ arcs $\mathcal{A}$. The nodes are partitioned into three groups, called layers: a set of $I$ nodes $\mathcal{N}_I$, called input layer, a set of $H$ nodes $\mathcal{N}_H$, called hidden layer and a set of $O$ nodes $\mathcal{N}_O$, called output layer. Each node of the output layer is connected with all the nodes of the hidden layer, and each node of the hidden layer is connected with all the nodes of the input layer. A real value $W_{ij}$, called weight, is associated with the arc from the node $j$ to the node $i$. The network is used for representing a real function. Each input layer node is associated with an input (real) value and each output layer node is associated with an output.
(real) value of the function. The output values are computed by the network, by using the input values. Hidden layer nodes are associated with intermediate results of the computation. The network computes the output values as follows: both each hidden and output layer node \( n \) is provided with the same function \( a \), that is called activation function, and with a parameter \( \theta_n \), that is called bias. Each hidden layer node \( j \) computes its associated hidden value \( h_j = a(\sum_{i=1}^{n} W_{ji} \cdot I_i - \theta_h) \), where \( i \) is an input layer node, i.e., by computing the weighted sum of the input values \( I_i \) using the weights associated to all the connections between each input layer node \( i \) and the hidden layer node \( j \). Analogously, each output layer node \( j \) computes its associated output value \( o_j = a(\sum_{h=1}^{m} W_{oh} \cdot H_h - \theta_o) \), where \( h \) is an hidden layer node, i.e. by computing the weighted sum of the hidden values \( H_h \) using the weights associated to all the connections between each hidden layer node \( h \) and the output node \( o \). Multilayer feedforward networks are commonly used for approximating real functions. In [26], Hetch-Nielsen states that any continuous function defined on \( I^m \) \( \text{4} \) could be implemented exactly by a three layered networks with \( 2n + 1 \) units in the hidden layer with transfer function \( \lambda^p \phi_q \ (p = 1, \ldots, n; q = 1, \ldots, 2n + 1) \) from the input to the hidden units and \( \psi \) from the hidden to the output units, where \( \lambda \) is a constant and \( \phi_q, \psi \) are functions highly non linear and very difficult to calculate with, thus they are not suitable to be used as activation function in a neural network. In [29], Hornik et al. proved that any continuous function \( f \in C(I^n) \) \( \text{5} \) can be approximated arbitrary well in the supremum norm \( \text{6} \) by a three-layered network with \( m \) semilinear hidden units using a threshold function and one linear output unit, formally: \( \vert f(x) - \sum_{i=1}^{n} w_i g(\sum_{j=1}^{n} a_{ij} x_j + c_i) \vert < \varepsilon \), where real numbers \( w_i \) and \( a_{ij} \) are the weights and \( c_i \) the thresholds. The semilinear activation functions have the form \( g(L(x) - b) \), where \( L(x) \) is linear in \( x \) and the function \( g \) is a monotone real function with the limits \( \lim_{x \to -\infty} g(x) = 0 \) and \( \lim_{x \to \infty} g(x) = 1 \).

Hence, \( g \) could be a sigmoidal function, e.g. \( \frac{1}{1+e^{-x}} \). In [28], Hornik generalized previous results, proving that whenever the activation function \( a \) is bounded and nonconstant, then the multilayer feedforward network is capable of approximate every \( f \in L^p \mu(I^n) \) \( \text{7} \), and if additionally \( g \) is continuous, then every \( f \in C(I^n) \), arbitrarily well with respect to the corresponding norm using sufficiently many hidden neurons. It can be concluded that it is not the specific choice of the activation function, but rather the multilayer feedforward architecture itself which gives neural networks the potential to be universal approximators. A neural network can be exploited for representing,

---

4 \( I^n \) is the \( n \)-dimensional unit cube \([0, 1]^n\)

5 By \( C(I^n) \) we denote the set of all continuous real-valued functions on \( I^n \)

6 The supremum norm of the function \( f \) on \( X \) is defined by \( \|f(x)\| = \sup \{|f(x)|, x \in X\} \), where \( \sup \) stands for “superior limit”

7 \( L^p \mu(X) \) is the set of all real functions \( f \) on \( X \) for which the Lebesgue integral \( \int |f|^p d\mu \) is finite
in an approximate way, functions belonging to $C(I^n)$ of an agent ontology. The constraint of using functions with inputs belonging to the interval $[0,1]$ is not a relevant limitation (it is always possible to normalize a real input in the interval $[0,1]$ if we known the maximum and the minimum input value, as in most of real cases). However, if we have a function and we know the algorithm that associates to each configuration of the inputs the corresponding configuration of the outputs, there is no reason to represent the function by a neural network, that is only an approximation of the function. The use of neural networks is suitable if the algorithm underlying the function is not known. Consider, as an example, the function $John\_new\_offer$ of the agent $John$ that takes the object $proposedPrice$ as input and yields the object $offer$ as output. $John$, monitoring the behaviour of its user (a customer involved in an e-commerce transaction) observes some associations between values of $proposedPrice$ and corresponding values of $offer$ (e.g. if $proposedPrice = 20$ then $offer = 10$, if $proposedPrice = 15$ then $offer = 12$ and so on). The problem of $John$ consists in representing these observed associations by a computational model and, more important, in generalizing these associations for determining, in correspondence of a new, not previously observed, input an appropriate output. In other words, $John$ has to learn an unknown function from examples. Neural Networks give this possibility. In fact, it is possible to use a learning algorithm for determining the parameters (weights and biases) of the network by using a set of associations input-output as a training set for the algorithm. We define below how a neural network can be associated with a function schema belonging to an ontology.

**Definition 32** Let $fs = (input, output, returned)$ be a function schema. The neural network associated with $fs$, denoted by $NN_f$, has a node $n_i$ (resp. $n_o$) in the input layer $NN_I$ (resp. output layer $NN_O$) associated with each object schema $o_i$ (resp. $o_o$) and each category schema $c_i$ (resp. $c_o$) belonging to input (resp. output) and a node $n_r$ in the output layer $NN_O$ associated with the output object schema $returned$. □

The neural network associated with a function schema $fs$ can be trained by a set of patterns, and thus setting its internal parameters (weights and biases) in order to learn the patterns. After the training, the neural network will represent, with an approximation error arbitrarily small, a function $f$ instance of the function schema $fs$.

### 6.2 Neural networks that learn extended logic programs

Several authors [12,44,55] suggested that symbolic knowledge can be represented by a connectionist system, as a neural network, in order to build effective learning system. In particular, in [12], it is proved that, for each extended
logic program $P$, there exists a feedforward neural network $N$ with exactly one hidden layer and semi-linear activation functions, that is equivalent to $N$ in the sense that $N$ computes $T_P$. Authors give the following constructive definition of $N$.

**Definition 33** [12] Let $P$ an acceptable extended program. The neural network $N$ associated to $P$ is defined as follows. For each clause $c$

$b^c \leftarrow a_1^c, a_2^c, ..., a_n^c, \sim a_{n+1}^c, \sim a_{n+2}^c, ..., \sim a_m^c$

of $P$, a node $n_i^c$ in the input layer of $N$ is associated with each event $a_i^c$, $i = 1, ..., m$ of the clause’s body, a node $n_h^c$ in the output layer of $N$ is associated with the event $b^c$ of the clause’s head and a node $n_h^c$ in the hidden layer of $N$ is associated with the clause. If an event appears in more than one clause, only a node is associated to it in the network. A node can assume a value belonging to the interval $[-1,1]$, as the result of the bipolar semi-linear activation function

$a(x) = \frac{1}{1+e^{-\beta x}}$. A minimum activation value $A_m$, $0 < A_m < 1$ is defined, representing the minimum activation for a node to be considered true, and a maximum activation value $A_M$, $-1 < A_M < 0$ is also defined, representing the maximum activation for a node to be considered false. Thus, the state of a node is false if the result of its input function belongs to the interval $[-1, A_m]$, the state is true if the input function produces a value belonging to $[A_m, 1]$. Weights and biases computed by the algorithm do not allow the network to present activations in the range $[-A_M, A_M]$. An input node has a fixed value, true or false. By setting an input node $a$ equal to true, we add the fact $a \leftarrow$ to the network. By setting an input node $a$ equal to false, we mean that there is no evidence that $a$ is true, that is we represent the default negation of $a$. Note that, in order to mean that $a$ is false in the strong sense, we have to explicitly add the fact $\neg a \leftarrow$ to the network, by introducing the input node $\neg a$ and setting it to true. An arc $(n_h^c, n_h^c)$ with weight $W$ is introduced from the hidden layer node to the output node, an arc $(n_i^c, n_h^c)$, $i = 1, ..., n$ with weight $W$ is introduced from each input layer node corresponding to an event not negated by default to the hidden layer node, and an arc $(n_i^c, n_h^c)$, $i = n + 1, ..., m$ with weight $-W$ is introduced from each input layer node corresponding to an event negated by default to the hidden layer node.

In [12], the following translation method for computing the values of $W$ as well as the values of the bias associated to each hidden and output layer node, is also presented:

**Translation method**: Given a general logic program $P$, let $q$ denote the number of clauses $C_l$ ($1 \leq l \leq q$) occurring in $P$; $m$, the number of literals occurring in $P$; $A_m$, the minimum activation for a neuron to be considered active (or true); $A_M$, the maximum activation for a neuron to be considered not active (or false); $h(x) = \frac{2}{1+e^{-\beta x}} - 1$, the bipolar semi-linear activation function.
function, where $\beta$ is the steepness parameter (that defines the slope of $h(x)$); $W$ (resp. $-W$), the weight of connections associated with positive (resp. negative) literals; $\theta_l$, the threshold of hidden neuron $N_l$ associated with clause $C_l$; $\theta_A$, the threshold of output neuron $A$, where $A$ is the head of clause $C_l$; $k_l$, the number of literals in the body of clause $C_l$; $p_l$, the number of positive literals in the body of clause $C_l$; $n_l$, the number of negative literals in the body of clause $C_l$; $\mu_l$, the number of clauses in $P$ with the same atom in the head for each clause $C_l$; $\max_P(k_1, \ldots, k_q, \mu_1, \ldots, \mu_q)$, the greatest element among all $k$s and $\mu$s of $P$.

Now, assume that the literals of $P$ are numbered from 1 to $m$ such that the input and output layers of $N$ are vectors of maximum length $m$, where the $i$-th neuron represents the $i$-th literal of $P$. Assume, for mathematical convenience and without loss of generality, that $A_M = -A_m$. The translation is realized by executing the following steps:

1. Calculate $\max_P(k_1, \ldots, k_q, \mu_1, \ldots, \mu_q)$ of $P$;
2. Calculate $A_m = \max_P(k_1, \ldots, k_q, \mu_1, \ldots, \mu_q) - 1$;
3. Calculate $W \geq \frac{2}{\beta} \cdot \frac{\ln(1 + A_m) - \ln(1 - A_m)}{\ln(1 + A_m) + A_m + 1}$;
4. For each clause $C_l$ of $P$ of the form $A ← L_1, \ldots, L_k$ ($k \geq 0$):
   a. Add a neuron $N_l$ to the hidden layer of $N$;
   b. Connect each neuron $L_i$ ($1 \leq i \leq k$) in the input layer to the neuron $N_l$ in the hidden layer. If $L_i$ is a positive literal then set the connection weight to $W$; otherwise, set the connection weight to $-W$;
   c. Connect the neuron $N_l$ in the hidden layer to the neuron $A$ in the output layer and set the connection weight to $W$;
   d. Define the threshold $\theta_l$ of the neuron $N_l$ in the hidden layer as $\theta_l = \frac{(1 + A_m)(k_l - 1)}{2} \cdot W$;
   e. Define the threshold $\theta_A$ of the neuron $A$ in the output layer as $\theta_A = \frac{(1 + A_m)(1 - \mu_l)}{2} \cdot W$.
5. Set $h(x)$ as the activation function of the neurons in the hidden and output layers of $N$. In this way, a gradient descent learning algorithm, such as backpropagation, can be applied on $N$ efficiently.
6. If $N$ ought to be fully-connected, set all other connections to zero.

As an example, consider the logic program of the example of John’s ontology. The neural network that represents it, built by following Definition 33, is shown in Figure 5. Note that in this example, the nodes interestingBook and tooHigh appear both in the input and the output layer. This happens because the associated events appear both in the body and in the head of some clauses. The network can compute in one step the operator $T_P(i)$. In order to compute the upward powers $T_P^n(i)$ of $T_P(i)$ it is necessary to iterate the computation of $T_P(i)$. To this purpose, an arc for each pair of homonym nodes is introduced, from the input layer node to the corresponding (homonym) output layer node. The weight $W_r = 1$ is associated to each of these arcs, obtaining a (partial) recurrent neural network. A recurrent neural network
can be used for performing deduction, in the sense that it can compute the
(unique) fixed point \( T^\infty_P(i) \) of an acceptable extended logic program. Hence,
a neural network may be fruitfully exploited for encoding the clauses of the
program \( P \) contained in an agent ontology. When the agent initially starts
to monitor the activities of its owner, the program \( P \) can be viewed as a
background knowledge relative to the agent’s owner. As an example of neural
network representation of a program, consider the case of the network of the
Figure 5, representing the background knowledge of the agent John, as
described above. For this case, we have \( q = 3, m = 5, k_1 = 2, k_2 = 2, k_3 = 1, \)
\( p_1 = 2, p_2 = 1, p_3 = 0, n_1 = 0, n_2 = 1, n_3 = 1, \mu_1 = 1, \mu_2 = 1, \mu_3 = 1, \)
\( \max_P(k_1, k_2, k_3, \mu_1, \mu_2, \mu_3) = 2. \) Therefore, we have \( A_m > 0.333. \) Then,
supposing to choose \( A_m = 0.5, \) we have \( W \geq \frac{4394}{\beta} \) and then, choosing \( \beta = 1, \)
we choose to set \( W = 4.5. \) The biases are \( \theta_{H1} = \theta_{H2} = 0.75 \cdot W = 3.375, \)
\( \theta_{H3} = 0, \) \( \theta_{MakeO} = \theta_{InterestingBook} = \theta_{¬tooHigh} = 0. \) We also use, as func-
tion \( h(x) \) of both the hidden and the output nodes, the standard bipolar
semi-linear function \( h(x) = \frac{2}{1+e^{-ax}} - 1. \) This network, given any initial acti-
vation in the (recursive) input layer nodes \( InterestingBook \) and \( ¬tooHigh, \)
always converges to the following state: \( MakeO = true, InterestingBook = true, ¬tooHigh = true, ¬smallPrice = false, LoveStory = false. \) For
instance, if we initially set all the inputs equal to \(-0.8, \) corresponding to set
\( InterestingBook = false, ¬tooHigh = false, ¬smallPrice = false, LoveStory = false, \)
\( InterestingBook = false, \) we can compute the hidden nodes’ values \( H1 = h(-4.5 \cdot 0.8 - 4.5 \cdot 0.8 - 3.375) = -0.999, H2 = h(4 \cdot -0.8 - 4 \cdot -0.8 - 3.375) = -0.936, H3 = h(-4 \cdot -0.8) = 0.922 \) and then the network yields the output nodes’
values \( MakeO = h(4.5 \cdot -0.999) = -0.977 \) (corresponding to the logic value
false), \( ¬tooHigh = h(4.5 \cdot -0.936) = -0.970 \) (corresponding to the logic value
false) and \( InterestingBook = h(4.5 \cdot 0.922) = 0.969 \) (corresponding to the logic value true). As you can see in Fig. 5, the change of the value of \( InterestingBook \) has effect on the network input, due to the presence
of recursive arcs from the output to the input layer: we have thus to re-compute the values \( H1 = h(4.5 \cdot 0.969 + 4 \cdot -0.970 - 3.375) = -0.934 \) and
\( H2 = h(4.5 \cdot 0.969 - 4 \cdot -0.8 - 3.375) = 0.979, \) and then we have
\( MakeO = h(4.5 \cdot -0.934) = -0.977 \) (corresponding to the logic value false),
\( ¬tooHigh = h(4.5 \cdot 0.979) = 0.976 \) (corresponding to the logic value true).
Finally, we have to execute another computation, since the value of \( ¬tooHigh \)
is changed, where we obtain \( H1 = h(4.5 \cdot 0.969 + 4 \cdot 0.976 - 3.375) = 0.991 \) and
\( MakeO = h(4.5 \cdot 0.991) = 0.977 \) (corresponding to the logic value true) and
thus the final, stable, outputs are \( MakeO = true, ¬tooHigh = true \) and
\( InterestingBook = true. \) Remember that the input nodes \( LoveStory \) and
\( ¬smallPrice, \) that are not recursive, have the fixed state false. Two rele-
vant cases can be studied, relative to the knowledge refinement process. The
first case corresponds to the application of a reasoning process to the refined
knowledge by the agent. The second case deals with the induction process. We
analyze these case in the following two subsections.
6.2.1 Reasoning

Firstly, it is obviously possible that events that do not appear in the head of any rules (i.e. that cannot be derived by other events) become true or false in consequence of the behaviour of the agent’s owner, and that other events are derived by applying the rules contained in the ontology. For instance, suppose that the agent John monitors the behaviour of its owner and observes that the event loveStory does not happen. In this case, the background knowledge is currently valid, and the neural network representing this knowledge converges to the state makeO = true, interestingBook = true, ¬tooHigh = true, ¬smallPrice = false, loveStory = false, as seen above. Consequently, since makeO fires, the action a_John activates the function John_new_offer. As another example, suppose that John observes that the event loveStory happens, that is, the node loveStory has the state true (this means that the John’s owner currently selects a book in the category loveStory). The neural network is changed with respect to the initial state: it now converges to the state makeO = false, interestingBook = false, ¬tooHigh = false, ¬smallPrice = false, loveStory = true. Consequently, since makeO does not fire, the function John_new_offer is not activated.

6.2.2 Induction

It is also possible that the agent observes a correlation between events, i.e., independently from the knowledge encoded in the ontology, a state configuration of events is produced after another configuration of events happens. This is the core of the induction process, because in this situation the agent
has to learn the causal implication between events directly from the observation. For instance, suppose that the agent John observes that the event interestingBook fires if both loveStory and ¬smallPrice happen. By applying the rules contained in the background knowledge, after having assigned to both the input nodes loveStory and ¬smallPrice the value true, we derive for interestingBook the value false. However, the direct observation of the human behaviour detects the event interestingBook fires. The agent could suppose that there exists a causal implication between the pair of events (loveStory, ¬smallPrice) and the event interestingBook and, as consequence, that it is necessary to modify the topology of the neural network for taking into account this fact. This modification can be automatically performed by applying to the neural network a constructive learning algorithm. Constructive Learning Algorithms are capable to obviate the need of an a priori choice of the network topology, but instead adaptively adding neurons as needed for improving classification accuracy [44]. Therefore, they are suitable to train an initial neural network, representing an initial background knowledge as, for instance, that depicted in Figure 5. There exist many different constructive neural network learning algorithm, as the M-Tiling described in [58] and the Pyramid proposed in [17], or the Dynamic Node Creation Method presented in [4]. As an example, the Dynamic Node Creation Method adds fully-connected nodes to the hidden layer of a feed forward neural network architecture trained using the Back Propagation algorithm. Training starts with an initial number of nodes in the hidden layer, and proceeds until the functional mapping is learned or the error ceases to descend and a new hidden node is added. After addition of a new node both the weights involving the new nodes and previous weights are retrained. If we apply the Dynamic Node Creation algorithm to the network in Figure 5, by presenting it the pattern composed by the inputs makeO = false, interestingBook = false, ¬tooHigh = false, ¬smallPrice = true, loveStory = true and the outputs makeO = false, interestingBook = true, ¬tooHigh = false, by setting a tolerance for the overall error equal to 0 (i.e. we want to obtain a perfect mapping between input an output), the algorithm verifies that, by using the three neurons of the initial configuration the error is over the tolerance, due to the fact that the network produces interestingBook = false. As a consequence, a hidden node is added to the network, and this node is connected with all the input and the output nodes. Then, the backpropagation algorithm is applied to the network. The new network that we obtain is described in the Figure 6, where we have omitted to depict the arcs (interestingBook, H₄), (¬tooHigh, H₄), (H₄, makeO) that are resulted equal to 0 at the end of the computation, and then are not influent. The backpropagation algorithm computes also the weights W₁ = W₂ = 4.63, W₃ = 4.57. The refined knowledge is numerically encoded in the weights of the network; in order to make clearly understandable such a knowledge, we need to transform it in an equivalent symbolic format. Several approaches to the problem of extracting symbolic knowledge from a trained neural network have been proposed in the literature [44,10]. In par-
6.2.3 Knowledge Extraction from Neural Networks

Given a particular set of weights and thresholds, resulting from a training process on a neural network, the extraction problem can be defined as follows: Find for each input vector $i$, all the outputs $o_j$ in the corresponding output vector $o$ such that the activation of $o_j$ is greater than $A_m$, where $A_m$ is the minimum activation function defined in 6.2 (in this case, we say that output neuron $o_j$ is active for input vector $i$). For example, consider a network with input neurons $a$ and $b$. If $i = (1, -1)$ activates output neuron $c$, then we derive the rule $c \leftarrow a \sim b$. As a result, if the input vector $i$ has length $p$, there are $2^p$ possible input vectors to be checked. We want to find the activation value of $o_j$, $\text{Act}(o_j) = h(\sum_{i=1}^{r} W_{ji} \cdot H_i - \theta_o)$, such that $\text{Act}(o_j) > A_m$. Considering the monotonically crescent characteristic of the activation function $h(x)$ and given that $0 < A_m < 1$ and $\beta > 0$, we can rewrite $h(x) > A_m$ as $x > h^{-1}(A_m)$, that is we can say that $o_j$ is active for $i$ iff $\sum_{i=1}^{r} W_{ji} \cdot H_i > A_m + \theta_o$.

For instance, applying this approach to the simple network of Figure 7.(a), we see that the the output interestingBook is active for the input loveStory = ¬smallPrice = 1, and then we extract the rule interestingBook ← loveStory, ¬smallPrice. We can easily see that no other rules can be extracted from this network, without checking all the other input configurations. In fact, since $W_1, W_2 > 0$, it is easy to verify that the ordering of Figure 7.(b) on the set of input vectors $I =$

Fig. 6. The neural network representing a refinement of the John’s background knowledge

Fig. 6. The neural network representing a refinement of the John’s background knowledge

Fig. 6. The neural network representing a refinement of the John’s background knowledge

Fig. 6. The neural network representing a refinement of the John’s background knowledge

Fig. 6. The neural network representing a refinement of the John’s background knowledge

Fig. 6. The neural network representing a refinement of the John’s background knowledge

Fig. 6. The neural network representing a refinement of the John’s background knowledge

Fig. 6. The neural network representing a refinement of the John’s background knowledge

Fig. 6. The neural network representing a refinement of the John’s background knowledge

Fig. 6. The neural network representing a refinement of the John’s background knowledge

Fig. 6. The neural network representing a refinement of the John’s background knowledge

Fig. 6. The neural network representing a refinement of the John’s background knowledge

Fig. 6. The neural network representing a refinement of the John’s background knowledge

Fig. 6. The neural network representing a refinement of the John’s background knowledge

Fig. 6. The neural network representing a refinement of the John’s background knowledge

Fig. 6. The neural network representing a refinement of the John’s background knowledge

Fig. 6. The neural network representing a refinement of the John’s background knowledge
Fig. 7. A single hidden layer neural network and the ordering of the set of its input vector

\[ \text{[loveStory, } \neg \text{smallPrice]} \] holds with respect to the output \text{interestingBook} of the network. The ordering says that the activation of the output is maximum for \( I = [1, 1] \) and minimum if \( I = [-1, -1] \). Therefore, the activation value of \text{interestingBook} is governed by this ordering.

This example helps us to understand that, in the case of a positive neural network, that is a network whose weights have all a positive value, it is easy to find an ordering on the set of input vectors \( I \) with respect to the set of output vectors \( O \) and that it is possible to exploit such information for extracting symbolic knowledge from the network. The ordering can help to reduce the search space, since we can safely avoid checking irrelevant input vectors, in the sense that those vectors that are not checked would not generate new rules, as in the case of the example above. Notice that in the worst case we still have to check \( 2^n \) input vectors, where \( i \) is the number of input nodes, and in the best case we only need to check one input vector (either the minimum or the maximum element in the ordering).

While the knowledge extraction problem can be efficiently solve, in many cases, in presence of positive networks, it becomes more difficult in the case of networks having some negative weights. The approach presented in [10] gives an algorithm that solve the extraction problem for non positive networks that present a “regularity”. They define regular networks those networks either with all the weights from the hidden layer to each output neuron positive or with all of them negative. For a regular network \( N \) they define a Transformation Algorithm that generates a positive network equivalent to \( N \), and therefore they propose a Knowledge Extraction Algorithm based on the Transformation Algorithm. They also define an extraction algorithm for network that are not regular, but in this case the algorithm is not complete, in the sense that it is not sure that all the possible rules encoded in the network will be extracted.
The formalization of the Knowledge Extraction problem is complex enough, and it is out of the scope of this paper. For more details, we refer the interested reader to the paper [10].

6.3 The IACOM model

In the previous two subsections, we have seen that:

- Standard feedforward neural networks with as few as a single hidden layer are capable of approximating any (Borel measurable) function from one finite dimensional space to another, provided sufficiently hidden layer nodes are available [29].
- Standard feedforward neural networks with as few as a single hidden layer are capable of learning any extended acceptable logic programs [12].
- It is possible to extract an extended logic program from a trained neural network [10].

Here, we propose to apply these results to the problem of building agent ontologies. In particular, we propose to associate to an agent ontology a neural network obtained by composing a set of neural networks with three layers. One of the neural networks of the set (partially recurrent) is used for learning the program of the ontology. To this purpose, the first and the third layer of such a network are associated with events that appear in the ontology program, whereas the second layer is used as a hidden layer. Furthermore, for each function represented in the ontology, a neural network of the set is used for learning the function. To this purpose, the first (resp. third) layer of this network is associated with objects that appear as input (resp. output) parameters in a function, whereas the second layer is used as a hidden layer. Finally, in order to represent an action that associates an event to a function, we introduce an arc, called action arc, having a weight equal to 1, from the node e, associated to the event, to the node f representing the output of the function. Moreover, we modify the transfer function of f, in such a way that f produces the function output only if the event e is equal to true. We call Information Agent Connectionist Ontology Model (IACOM) such a neural network representation of an ontology. More formally:

Definition 34 Let O be an ontology. Let \( LF = \{f_1, f_2, ..., f_k\} \) be a set containing \( k \) functions of O to be learnt and let \( \mathcal{N}_1 = (N_1, A_1), \mathcal{N}_2 = (N_2, A_2), ..., \mathcal{N}_k = (N_k, A_k) \) be the neural networks associated to the functions \( f_1, f_2, ..., f_k \), resp. Let \( \mathcal{N}_P = (N_P, A_P) \) be the neural network associated to the program \( P \). Let \( AA \) be a set of action arcs \((e, f_o)\), where e is an event that appears in the head of some rules of \( P \), \( f_o \) is the output node of a neural network \( NN \) associated to a function \( f \in LF \) and \( e, f \) belong to an action in
The IACOM representation of $\mathcal{O}$ is the network $\mathcal{N}\mathcal{N} = (N, A)$, where $N = N_0 \cup N_1 \cup N_2, \ldots, \cup N_k$ and $A = A_0 \cup A_1 \cup A_2, \ldots, \cup A_k \cup AA$. □

It is always possible to build an IACOM representation of an agent ontology, equivalent to the ontology in the sense that each function of the ontology is approximated in the IACOM representation with an approximation error arbitrary small and the logic program is represented with an equivalent neural network. More formally:

**Definition 35** Let $\mathcal{O}$ be an agent ontology. Let $LF = \{f_1, f_2, \ldots, f_k\}$ be a set containing $k$ functions of $\mathcal{O}$ to be learnt and let $\mathcal{N}\mathcal{N}_1, \mathcal{N}\mathcal{N}_2, \ldots, \mathcal{N}\mathcal{N}_k$ be the neural networks associated to the function $f_1, f_2, \ldots, f_k$, respectively. Let $\mathcal{N}\mathcal{N}_p$ be the neural network associated to the program $\mathcal{P}$. The IACOM representation of $\mathcal{O}$, represented by the neural network $\mathcal{N}\mathcal{N}$, built following the Definition 34 is equivalent to $\mathcal{O}$ if both $\mathcal{N}\mathcal{N}_p$ is equivalent to $\mathcal{P}$, and $\mathcal{N}\mathcal{N}_j$ approximates $f_j$ with an approximation error arbitrarily small, for each $j = 1, \ldots, k$. □

This is only the definition of equivalence between an agent ontology $\mathcal{O}$ and an IACOM neural network. However, it is not always possible to find an IACOM representation for whatever ontology. The limitation in finding an IACOM representation is represented by the typology of both the logic program and the functions contained in $\mathcal{O}$. Only if the logic program is acceptable and all the functions are Borel measurable, we can be sure that an IACOM representation of $\mathcal{O}$ exists. This is formally expressed in the following theorem.

**Theorem 3** Let $\mathcal{O}$ be an agent ontology and let $LF = \{f_1, f_2, \ldots, f_k\}$ a set containing $k$ functions of $\mathcal{O}$, such that $\mathcal{P}$ is an acceptable extended logic program and, for each $f_j \in LF$, $f_j$ is a Borel measurable function. Then, there exists an IACOM representation of $\mathcal{O}$ equivalent to $\mathcal{O}$.

**Proof.** It is always possible to build the neural network $\mathcal{N}\mathcal{N}$, associated to $\mathcal{O}$, by using the method defined in the Definition 34. In fact, since $\mathcal{P}$ is acceptable, the Definition 33 gives an algorithm for building a neural network $\mathcal{N}\mathcal{N}_p$ equivalent to $\mathcal{P}$. Moreover, the Definition 32 gives a method for building a neural network $\mathcal{N}\mathcal{N}_j$ equivalent to the function $f_j \in LF$, for each $j = 1, \ldots, k$, due to the fact that $f_k$ is Borel measurable. Since the parameters associated to the arcs of the set $AA$ of the action arcs do not have to be learnt, the theorem is true. □

39
In this subsection we study the computational costs (both in terms of time as well as of requested storage space) for constructing and handling the IACOM ontology of an agent. There are two types of costs to be considered, namely the cost for storing the agent ontology, the cost for refining the knowledge by a learning phase and the cost for using the IACOM ontology for simulating agents’ behaviour. Below, we estimate these costs with respect to a single agent (the corresponding costs for the whole community can be obtained by multiplying each single agent’s costs for the number of the agents in the community). In the following, we refer to an agent ontology $\mathcal{O}$ containing (i) the program $\mathcal{P}$, (ii) a set $F$, with cardinality $\Phi$, of functions and (iii) a set $A$, with cardinality $\Lambda$, of actions. We denote by $\Psi$ and $\Gamma$ the number of distinct events belonging to the bodies of $\mathcal{P}$, and the number of clauses, respectively. Moreover, we denote as $e_i$ and $h_i$ the number of schema instances and the number of hidden nodes of the function $f_i \in F$, $i = 1...\Phi$, respectively.

### Computational Complexity of constructing an IACOM ontology

The neural network that represents the IACOM ontology has a size that depends both on the size of the extended logic program (number of events and causal implications), as well as on the number of schema instances and hidden nodes involved in the functions.

**Theorem 4** The number of biases and weights in the IACOM representation of $\mathcal{O}$ are $O(\Psi + 2 \cdot \Gamma + \sum_{i=1}^{\Phi} (e_i + h_i + 1))$ and $O(\Gamma \cdot (\Psi + \Gamma) + \sum_{i=1}^{\Phi} h_i \cdot (e_i + 1) + \Lambda)$, respectively.

**Proof.** The number of biases and weights in a neural network having a number $I$ of input nodes, a number $O$ of output nodes and a number $H$ of hidden nodes are $O(I + O + H)$ and $O(H \cdot (I + O))$, respectively [31]. The neural network representing $\mathcal{P}$ has $\Psi$ nodes in the input layer (a node for each distinct event) and $\Gamma$ nodes both in the hidden and the output layer, than the number of biases is $O(\Psi + 2 \cdot \Gamma)$ and the number of weights is $O(\Gamma \cdot (\Psi + \Gamma))$. The neural network associated with each function $f_i$ has a number $e_i$ of input nodes, a number $h_i$ of hidden nodes and just one output node, then the number of biases is $O(\sum_{i=1}^{\Phi} (e_i + h_i + 1))$ and the number of weights is $O(\sum_{i=1}^{\Phi} h_i \cdot (e_i + 1))$. The complexity of the whole IACOM representation is obtained by summing the contributions of both the network associated to $\mathcal{P}$ and the networks associated to each function $f_i$, by also considering in the complexity of the weights the additional contribution of the $\Lambda$ arcs associated to the actions of $A$. □

### Computational Complexity of the learning phase

First of all, we need to define a basic terminology concerning the class of backpropagation algorithms we use for performing the learning phase. A pattern of data is one
complete sample of the input-output correlation for one time-step. If the network has multiple inputs, then a pattern corresponds to one piece of data from each input. An epoch is a complete set of data presented to the network. The number \( K \) of epochs required for obtaining a requested approximation error strictly depends both on the number of nodes in the hidden layer and on the number of epochs used for the training, as well as on the particular backpropagation algorithm we choose, e.g. standard momentum, windowed momentum, etc. (see [31] for more details). In particular, in [1] it is pointed out that, for large \( K \), the approximation error decreases as \( 1/K \).

**Theorem 5** Let \( B \) be the number of distinct events appearing in the bodies of the clauses of \( \mathcal{P} \), let \( C \) be the number of the clauses of \( \mathcal{P} \) and let \( P \) the number of distinct patterns we show to the ontology network during the training. The computational time complexity of the learning is \( \mathcal{O}(K \cdot \Gamma^2 \cdot (\Psi + 1) + \sum_{i=1}^{\Phi}(K \cdot P \cdot (e_i + 1))) \).

**Proof.** In [31], it is proved that a feedforward neural network with a number \( I \) of hidden nodes, a number \( O \) of output nodes and a number \( H \) of hidden nodes can be trained for \( K \) epochs by a backpropagation algorithm with a computational time complexity equal to \( \mathcal{O}(K \cdot H \cdot O \cdot (I + 1)) \). The complexity of the learning of the network associated to the program \( \mathcal{P} \) is thus \( \mathcal{O}(K \cdot \Gamma^2 \cdot (\Psi + 1)) \), deriving from the above general result, by considering that: \( I = \Psi; O = H = \Gamma \)

Analogously, the complexity of the learning of the network associated to each function \( f_i \) is \( \mathcal{O}(K \cdot h_i \cdot (e_i + 1)) \), by considering that, in this case, \( I = e_i; O = 1; H = h_i \)

In [30] it is proved that an upper bound for the number of the hidden neurons is \( \mathcal{O}(P) \), then the latter complexity time is \( \mathcal{O}(K \cdot P \cdot (e_i + 1)) \). The overall time of the learning phase is then obtained by summing the cost for training the network associated to \( \mathcal{P} \) and the costs for training the network associated to each function \( f_i \). \( \square \)

**Computational Complexity of simulating user behaviour.** In order to simulate the user behaviour, the agent has to use the IACOM ontology for (i) computing the operator \( T_{\mathcal{P}}(i) \), (ii) activating the functions by means of the action arcs and (iii) computing the result of each function \( f_i \in F \). We give the follow result:

**Theorem 6** The time for simulating one step of the behaviour of a user having ontology \( \mathcal{O} \) is \( \mathcal{O}(\Gamma \cdot (\Psi + \Gamma) + \sum_{i=1}^{\Phi}(h_i \cdot (e_i + 1))) \)

**Proof.** The time for computing the result of a neural network having a number \( I \) of input nodes, a number \( H \) of hidden nodes, and a number \( O \) of output nodes is \( \mathcal{O}(H \cdot (I + O)) \); this directly derives from the translation method presented in 6.2. Thus, the time for computing the operator \( T_{\mathcal{P}} \) is \( (\Gamma \cdot (\Psi + \Gamma)) \),
by considering that in the case of the network associated with the program \( P \) we have \( I = \Psi; O = H = \Gamma \). Analogously, the time for computing the result of the network associated with a function \( f_1 \in F \) is \( \sum_{i=1}^{\Phi} (h_i \cdot (e_i + 1)) \), since in this case \( I = e_i, H = h_i \) and \( O = 1 \). The overall time for simulating the user’s behaviour is then obtained by summing the cost for computing \( T_P \) and the costs for computing the result of the networks associated to each function \( f_i \).

\[ \square \]

7 Experimental results

In order to evaluate the practical utility deriving from the use of the inter-ontology similarities for supporting agent mutual monitoring, we have realized a set of experiments that compare the performances of our approach and those of other well-known techniques in the field of Collaborative Filtering Recommendation Systems (CFRS). CFRS are Recommendation Systems that require each user to specify his ratings about system recommendations; after this, they recognize commonalities among users on the basis of their ratings and generate new recommendations taking into account inter-user comparisons. CILIOS can be used for generating recommendations, by suggesting to a user \( u \) the pages most visited by the users that have the highest similarity with \( u \). For this purpose, an IACOM agent is associated to each user, that automatically constructs its ontology by monitoring user behaviour. Then, each IACOM ontology is translated into an equivalent IAOM representation by an IAOM agent, and a grid of OSM agents computes the inter-ontology similarities necessary for generating the recommendations.

We have performed our experiments by using a set of 35 agents, each associated to a human user and provided with an IACOM representation. Each agent has a background knowledge containing some rules involving literals that represent click event. A click event is an action performed by the user that clicks on a link belonging to a particular Web category. For instance, the rule \( CD \leftarrow Portal, \neg Book \) represents the behaviour of a user that clicks on a link to a Web site of a seller of Compact Discs each time he has both previously clicked on a link to a Web Portal (e.g. Ebay) and he has not clicked on a link to a Web bookstore. We define 120 possible Web categories (as Portal, CD, Book, etc.), denoted by \( g_1, g_2, ..., g_{120} \). Therefore the IACOM representation of each agent is a three layers neural network having an input layer with 120 nodes. Each node \( i \) of the input layer is associated to the Web category \( g_i \) and receives a boolean input equal to 1 if the user has clicked on a link belonging to \( g_i \), -1 otherwise. Analogously, also the output layer of the network is composed by 120 nodes, where each node \( j \) is associated to the Web category \( g_i \) and yields a boolean output that, when its is equal to 1, represents the prediction that the user will click on a link belonging to \( g_i \). For each of our 35 agents, we have represented by
the neural network a different set of rules, by using the translation algorithm presented in Section 6.2 for inserting the necessary hidden nodes into the hidden level, and consequently setting the initial weights and biases. Then, we have trained this network by using the Dynamic Node Creation algorithm and a training set composed by 2156 patterns. Each pattern represents a Web session performed by a user, and it is composed by 120 input \(i_1, i_2, \ldots, i_{120}\) and 120 output \(o_1, o_2, \ldots, o_{120}\), each input \(i_k\) (resp. output \(o_k\)) associated with the Web category \(g_k\).

An agent \(a\) is able to give a recommendation to its associated user \(u\), in the sense that, when the user clicks on a link for visiting a Web page, the agent automatically proposes a number of Web categories that \(u\) should consider interesting to visit. The recommendation is generated on the basis of two different contributions, namely (i) the predictions produced by CILIOS about the Web categories that \(u\) probably will visit in the future and (ii) the predictions produced by CILIOS about the Web pages that those users, whose agents have high overall similarity with \(a\), probably will visit in the future. CILIOS generates its predictions about the future pages for a user on the basis of the last 120 pages visited by this user, that are represented by the 120 input neurons of the IACOM representation of its agent. If the user has chosen in the last 120 clicks, one or more times, a page belonging to the group \(g_k\), the input of the node \(i_k\) of the network is set equal to \(true\); it is set equal to \(false\) otherwise. Those output neurons resulting equal to \(true\), produced by the network computation, represent the Web categories that probably the user will visit in the future. Let \(AS_u\) the set composing by both the agent \(a\) of \(u\) and those agents having an overall similarity with \(a\) greater than a fixed threshold \(t_s\). The score of a category \(k\) with respect to the user \(u\) is equal to the number of agents belonging to \(AS_u\) that have predicted \(k\) as a category to be visited. Let \(TOP N\) the number of Web categories that CILIOS has to recommend to \(u\) (this parameter is set by the user of CILIOS). CILIOS recommends the \(TOP N\) Web categories that have the highest scores. In this Section, we quantitatively compare the effectiveness of this Multi-Agent System based on CILIOS Architecture with:

(i) four of the most used collaborative-filtering approaches, namely the Markov Model (MM) [13], the Association Rule (AR) Model [37], the Sequential Association Rule (SAR) Model [38], and the hybrid model [33], that exploits different combinations of MM, AR and SAR\(^8\). These approaches are not based on ontologies, but the similarities between users are computed only comparing the choices made in the past in selecting Web pages;

(ii) A CFRS based on ontology, i.e. X-COMPASS [18].

\(^8\) The interested reader can found some details about these collaborative filtering techniques in [33]
Analogously to the CILIOS system, we have considered a set of 35 agents for all of the above approaches, that support the same 35 users considered by CILIOS. The performance metrics, precision, recall and F-measure, are used for measuring the approaches effectiveness. In order to measure the performance of the approaches in the experiments, accordingly with [33], a session data set is divided into a training and a test set. The training set is used to generate the user models underlying each approach (for instance, for training the neural networks of the CILIOS agents), while the test set is treated as current session data and used to evaluate the models. Each session $s$, composed by 120 group of links, in the test set is divided, as in the training set, into two parts. The first 60 values in $s$ are treated as an active session and are used for making predictions, while the remaining portion of the session is used to evaluate the prediction models. We define active session window, denoted by $as_s$, as the portion of a users active session used by an approach for prediction. The remaining portion of the session is denoted by output$_s$. Each approach takes $as_s$ as an input and makes a prediction. We denote the recommended pages as $P(as_s)$. Then precision, recall, and the F-measure can be represented as follows.

$$Pre(P(as_s)) = \frac{|P(as_s) \cap output_s|}{|P(as_s)|}$$

$$Rec(P(as_s)) = \frac{|P(as_s) \cap output_s|}{|test_s|}$$

$$F(P(as_s)) = \frac{2 * Rec(P(as_s)) * Pre(P(as_s))}{Rec(P(as_s)) + Pre(P(as_s))}$$

Following [33], we evaluated the performance of the considered approaches over varying TOP$N$, i.e. the number of pages recommended by an approach. TOP$N$ is a parameter that varies depending on the application. Typically, for applications that recommend relevant pages like a search result, the TOP$N$ is very large while, in other recommendations tasks like pre-fetching Web pages or cross-selling in EC, TOP$N$ is relatively small. For all the methods, the exploited training data set contains a total of 2156 session records, having the same structure described above relatively to the neural network training. The exploited test sets contains 533 patterns having the same structure. Tables 1,2,3 present the results obtained on this data set by an agent that uses our approach and those obtained by the approaches MM, AR, SAR, HYBRID and X-COMPASS, in terms of average precision, average recall and average F-measure, respectively. Each average has been evaluated by considering all the single results obtained in each session $s$.

These results show that CILIOS performs better, in the average, than all the other systems. Indeed, it presents, for each TOP$N$, the highest recall and F-measure.
<table>
<thead>
<tr>
<th></th>
<th>TOP 2</th>
<th>TOP 4</th>
<th>TOP 8</th>
<th>TOP 16</th>
<th>TOP 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM</td>
<td>0.280</td>
<td>0.225</td>
<td>0.182</td>
<td>0.155</td>
<td>0.156</td>
</tr>
<tr>
<td>SAR</td>
<td>0.232</td>
<td>0.199</td>
<td>0.164</td>
<td>0.128</td>
<td>0.118</td>
</tr>
<tr>
<td>AR</td>
<td>0.183</td>
<td>0.177</td>
<td>0.133</td>
<td>0.105</td>
<td>0.109</td>
</tr>
<tr>
<td>HYBRID</td>
<td>0.238</td>
<td>0.178</td>
<td>0.149</td>
<td>0.113</td>
<td>0.108</td>
</tr>
<tr>
<td>X-COMPASS</td>
<td>0.277</td>
<td>0.221</td>
<td>0.178</td>
<td>0.149</td>
<td>0.151</td>
</tr>
<tr>
<td>CILIOS</td>
<td>0.295</td>
<td>0.231</td>
<td>0.188</td>
<td>0.159</td>
<td>0.151</td>
</tr>
</tbody>
</table>

Table 1
The precision average

<table>
<thead>
<tr>
<th></th>
<th>TOP 2</th>
<th>TOP 4</th>
<th>TOP 8</th>
<th>TOP 16</th>
<th>TOP 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM</td>
<td>0.044</td>
<td>0.065</td>
<td>0.092</td>
<td>0.115</td>
<td>0.119</td>
</tr>
<tr>
<td>SAR</td>
<td>0.059</td>
<td>0.094</td>
<td>0.139</td>
<td>0.197</td>
<td>0.209</td>
</tr>
<tr>
<td>AR</td>
<td>0.049</td>
<td>0.08</td>
<td>0.125</td>
<td>0.199</td>
<td>0.231</td>
</tr>
<tr>
<td>HYBRID</td>
<td>0.061</td>
<td>0.098</td>
<td>0.16</td>
<td>0.232</td>
<td>0.257</td>
</tr>
<tr>
<td>X-COMPASS</td>
<td>0.071</td>
<td>0.134</td>
<td>0.177</td>
<td>0.254</td>
<td>0.281</td>
</tr>
<tr>
<td>CILIOS</td>
<td>0.079</td>
<td>0.143</td>
<td>0.188</td>
<td>0.274</td>
<td>0.281</td>
</tr>
</tbody>
</table>

Table 2
The recall average

<table>
<thead>
<tr>
<th></th>
<th>TOP 2</th>
<th>TOP 4</th>
<th>TOP 8</th>
<th>TOP 16</th>
<th>TOP 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM</td>
<td>0.075</td>
<td>0.107</td>
<td>0.123</td>
<td>0.127</td>
<td>0.124</td>
</tr>
<tr>
<td>SAR</td>
<td>0.09</td>
<td>0.124</td>
<td>0.157</td>
<td>0.159</td>
<td>0.155</td>
</tr>
<tr>
<td>AR</td>
<td>0.072</td>
<td>0.011</td>
<td>0.128</td>
<td>0.143</td>
<td>0.141</td>
</tr>
<tr>
<td>HYBRID</td>
<td>0.096</td>
<td>0.126</td>
<td>0.157</td>
<td>0.159</td>
<td>0.156</td>
</tr>
<tr>
<td>X-COMPASS</td>
<td>0.122</td>
<td>0.135</td>
<td>0.173</td>
<td>0.174</td>
<td>0.176</td>
</tr>
<tr>
<td>CILIOS</td>
<td>0.131</td>
<td>0.144</td>
<td>0.182</td>
<td>0.189</td>
<td>0.176</td>
</tr>
</tbody>
</table>

Table 3
The F-measure average

Measure. Note that recall measures the relevance of recommendation. This is important in some applications, e.g., cross-selling in EC, where the recall measure counts more than precision. Moreover, X-COMPASS and CILIOS present a precision almost equal to that of the Markov Model, that has the maximum precision but that shows the lowest recall. Among the considered methods, HYBRID has a recall and a F-measure comparable to (but lower than) CILIOS and X-COMPASS; however, its precision is sensibly lower, especially when we consider a high **TOP N**. The main reason of the better performances of CILIOS with respect to the other considered approaches is that CILIOS agents store internal representations of their associated users more rich than those of the corresponding MM, SAR, AR, HYBRID and X-Compass agents. In particular, a MM, SAR, AR or HYBRID agent that supports a user $u$ stores only the the Web categories visited by $u$, without considering any other information about the interests of $u$. An X-Compass agent stores an ontology of $u$ that however contains, beside the Web categories visited by $u$, only some correlations between these categories. The similarity measures between agents, computed by these approaches, do not consider logical implications as, for instance, $CD \leftarrow Portal, ¬Book$. Instead CILIOS considers logical implications, and it is able to compute similarities in a more precise way. It is important to point out that the construction of the CILIOS ontology, as well as the similarity computation, produces a cost that has to be considered as a disadvantage.
with respect to the other approaches seen above. Therefore, CILIOS seems adapt to be used when we need high performances in term of quality of the service, by paying a price in terms of time-cost. However, this price does not seem too high: The average times for performing the results described above, reported in Table 4, are competitive with X-COMPASS, although more high than the other approaches of about 15 percent.

<table>
<thead>
<tr>
<th></th>
<th>TOP 2</th>
<th>TOP 4</th>
<th>TOP 8</th>
<th>TOP 16</th>
<th>TOP 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM</td>
<td>0.92</td>
<td>0.95</td>
<td>1.13</td>
<td>1.42</td>
<td>1.87</td>
</tr>
<tr>
<td>SAR</td>
<td>0.949</td>
<td>0.99</td>
<td>1.18</td>
<td>1.55</td>
<td>1.81</td>
</tr>
<tr>
<td>AR</td>
<td>1.11</td>
<td>1.18</td>
<td>1.24</td>
<td>1.62</td>
<td>1.9</td>
</tr>
<tr>
<td>HYBRID</td>
<td>1.18</td>
<td>1.32</td>
<td>1.49</td>
<td>1.8</td>
<td>2.01</td>
</tr>
<tr>
<td>X-COMPASS</td>
<td>1.28</td>
<td>1.44</td>
<td>1.61</td>
<td>1.99</td>
<td>2.15</td>
</tr>
<tr>
<td>CILIOS</td>
<td>1.30</td>
<td>1.45</td>
<td>1.63</td>
<td>2.02</td>
<td>2.19</td>
</tr>
</tbody>
</table>

Table 4
The time costs (seconds)

8 Conclusions

In order to face the problem of realizing mutual monitoring in autonomous multi-agent systems, we have here presented an architecture organized in four different levels of agents, suitable to be implemented as JADE containers. Agents belonging to the first level are able to construct a connectionist, neural network-based representation of end-user, called IACOM, while agents of the second level translate such a connectionist representation into a symbolic ontology, following an ontology model called IAOM. IAOM allows to represent the elements of the agent’s reality as well as events and causal implications, by a unique formalism, and it appears as a suitable model for supporting the derivation of semantic similarities among agents. This derivation is performed, in our architecture, by the agents of the third level, called OSM. The capability of automatically deriving causal implications among events and mathematical mappings (functions) among objects and object collections are, in our opinion, characteristics that help CILIOS agents to improve their autonomy, while the ability of CILIOS system deriving inter-ontology similarities by a grid of dedicated agents allows to efficiently support agent mutual monitoring, that is a kind of agent cooperation. The presence of an underlying, rich enough ontology model makes CILIOS capable of improving its effectiveness in choosing the best agents for cooperation, as it is shown by some experiments we have conducted in comparison with other cooperative techniques. It is worth to point out that the efficiency of CILIOS is strongly related to the efficiency of IACOM agents in constructing and continuously updating ontologies. However, we note that CILIOS architecture coordinates mutual knowledge by activating centralized computations on distributed evolutionary models. This means that in its essence, CILIOS hardest computation is centralized, and this also explains its efficiency. The neural-symbolic network architecture here proposed
for performing such a task might be improved by other, more efficient, inductive (either connectionist or symbolic) mechanisms: this is the main topic of our ongoing research. Finally, it is important to point out that learning capabilities are not sufficient for guaranteeing an effective agent autonomy. For instance, when an agent receives a message, it cannot refuse to respond. So the “planning” or “scheduling of tasks” by the agent should also be guaranteed, based on the run time (learning) knowledge the agent has at its disposal. These and other problems concerning agent autonomy are subjects of our ongoing research.

References


