An Adaptative Multi-Agent System to Co-Construct an Ontology from Texts with an Ontologist

Zied Sellami and Valérie Camps

IRIT, Université de Toulouse, France
email: {sellami, camps}@irit.fr

Abstract. Ontologies are one of the most used representations to model the domain knowledge. An ontology consists of a set of concepts connected by semantic relations. The construction and evolution of an ontology are complex and time-consuming tasks. This paper presents DYNAMO-MAS, an Adaptive Multi-Agent System (AMAS) that automates these tasks by co-constructing an ontology from texts with an ontologist. Terms and concepts of a given domain are agentified and they act, according to the AMAS approach, by solving the non cooperative situations they locally perceive at runtime. These agents cooperate to determine their position in the AMAS (that is the ontology) thanks to (*i*) lexical relations between terms, (*ii*) some adaptive mechanisms enabling addition, removing or moving of new terms, of concepts and of relations in the ontology as well as (*iii*) feedbacks from the ontologist about the propositions given by the AMAS. This paper focuses on the instantiation of the AMAS approach to this difficult problem. It presents the architecture of DYNAMO-MAS, and details the cooperative behaviors of the two types of agents we defined for ontology evolution. Finally evaluations made on three different ontologies are given in order to show the genericity of our solution.

1 Introduction

In the last ten years, ontology engineering from texts has emerged as a promising way to save time and to gain efficiency for the construction or the evolution of ontologies [10]. But texts do not cover all the required information to construct or evolve a relevant domain model, and human interpretation and validation are required at several stages in this process. That is why ontology engineering remains a particularly complex task [29].
Our contribution in this paper completes a previous work [38] that proposes an AMAS named DYNAMO-MAS[1] enabling to construct an ontology from texts. DYNAMO-MAS automatically proposes new concepts and/or terms to be evaluated by an ontologist. This paper presents the design and the evaluation of DYNAMO-MAS, an interactive software based on an AMAS that aims at evolving ontologies from text. Section 2 describes related works regarding existing tools for evolving ontologies from text. It also analyses the links between Multi-Agent Systems (MAS) and ontologies. Section 3 is devoted to the presentation of the AMAS approach that we used to implement DYNAMO-MAS. The overview

[1]DYNAMO: DYNAMic Ontology for information retrieval; `http://www.irit.fr/DYNAMO/`; MAS: Multi-Agent System

of the DYNAMO project, the defined architecture as well as our approach for ontology evolution are detailed in section 4. Section 5 expounds the cooperative behaviors of the two types of agents we defined. Section 6 contains the experiments of ontology evolution that were carried out with DYNAMO-MAS and an analysis of their results. We conclude and plan some future works in section 7.

2 Related Works

This section gives a brief overview of systems dealing with automatic ontologies evolution from texts. A more general state of the art on ontologies evolution from texts can be seen in [37]. As this paper is devoted to the presentation of the core of DYNAMO-MAS, the related works section focuses on the links between ontologies and multi-agent systems.

2.1 Ontologies Evolution from Texts

Few existing works deal with the automatic evolution of ontologies from texts. Most of them focus on the construction of ontologies; if there is a need of updating the ontology, another ontology is built from scratch (Text Onto Miner [19], OntoLearn [45], Text-To-Onto [14] and DOGMA [33]). Some works concern the management of the ontology evolution process (usually manual) or the management and the comparison of different versions of an ontology [25], [17], [43,16]. Other works focus on the propagation of ontology modifications on some artifacts (other ontologies, applications, data, ...) [43], [25].
To our knowledge, only two systems propose automatic ontology evolution from texts. The first one, EVOLVA [49] uses results of terms extraction from texts as well as other ontologies to identify new concepts to include to the ontology. For each concept to add, it tries to retrieve if there is a relation between this concept and a concept already present in the current ontology. EVOLVA is only useful for evolving English ontologies. When the domain modeled by the ontology is very specific, EVOLVA has difficulties to detect relations between a new concept and current concepts of the ontology. A more detailed experiment and comparaison between DYNAMO-MAS and EVOLVA is given in [50]. The second system is a first prototype of DYNAMO [31]. It is only able to construct an ontology from scratch but not to make it evolving. The agents of the MAS implement a distributed clustering algorithm that identifies clusters of terms from a large text corpus. These clusters lead to the definition of concepts as well as their organization into a hierarchy. Each agent represents a candidate term extracted from the corpus and estimates its similarity with others thanks to statistical features. Several evaluations conducted with this DYNAMO first prototype confirmed that statistical approaches [23] are inefficient when texts are short (it is our case in the context of our work).
The table 1 shows a comparaison between DYNAMO-MAS and the previously presented tools for ontologies evolution.

- **Reconstruction** means that the tool evolves the ontology by the construction from scratch of a new one;
- **Incremental evolution** means that the tool evolves the current version of the ontology;

– **Tool availability** means that the tool is available, can be downloaded from the Web and easily used;
– **NLP Knowledge** means that the person who will use the tool needs to have knowledge in NLP;
– **Processed language** indicates the language of the corpus;
– **Depending on corpus size** means that the tool only works with large corpus;
– **For multiple domains** means that the tool is able to evolve ontologies coming from many domains.

Characteristics	Text Onto Miner	OntoLearn	Text-To-Onto	DOGMA	EVOLVA	DYNAMO First Prototype	DYNAMO MAS
Reconstruction	X	X	X	X		X	
Incremental evolution					X		X
Tool availability		X	X		X	X	X
NLP knowledge	Necessary	Not necessary	Not necessary	Not necessary	Not necessary	Not necessary	Not necessary
Processed language	EN	EN	EN	EN	EN	FR	EN-FR
Depending on corpus size	X	X	X	X		X	
For multiple domains	YES	YES	YES	YES	NO	YES	YES

Table 1. Comparaison of DYNAMO-MAS and other ontologies evolution tools.

The table 1 tells us that these tools are only effective for large corpus and are not suitable for small-volume corpus such as those of DYNAMO-MAS. Moreover, the user of TextOntoMiner has to be strongly qualified in NLP techniques to organize the processes of text analysis and ontology construction. Another difference is the attention given to the ontologist. In DYNAMO-MAS, we emphasize the notion of *interactive co-construction* of an ontology. This means the ontologist can accept, reject, or modify the proposals made by the system; the system has then to integrate the ontologist' s decisions to provide new proposals that the ontologist has then to check again, and so on. In other approaches, the tools are less interactive or not interactive at all (only the final ontology is proposed to the ontologist). This postpones the validation. Furthermore, these works manage in a different way ontology construction and ontology evolution, whereas we want to handle them uniformly. Finally, DYNAMO-MAS is a generic tool for ontology evolution from texts. A person who uses DYNAMO-MAS does not need to have NLP knowledge. DYNAMO-MAS can work with both small and large corpus in French or English languages.

2.2 Ontologies and Multi-Agent Systems

The notion of ontology is often mentioned when talking about communication and interaction between agents. Indeed, ontologies are used to allow agents to communicate and interact. They provide a formal basis for modeling languages communications between agents. By sharing the same ontology, that is to say the same vocabulary, agents are able to understand the messages exchanged and to respond efficiently. Much of researches concerning communication between agents and MAS became interested in exploiting ontologies and reasoning about ontologies in agents. Today, the use of ontologies and the need to evolve these ontologies highlight new challenges especially in terms of supports and tools. Other works have then used MAS in systems for ontologies evolution.

Ontologies for communication between agents To communicate and interact appropriately, agents need to understand and share a common language. Ontologies have been developed in this direction, to provide formal vocabularies that depend on application domain.

Stable ontologies in agents. Several studies have integrated ontologies in MAS especially the Semantic Web and Web Services [44] [26] [18] [21].
Jointly with MAS design, designers construct an ontology of the application domain (often with OWL[2]). This ontology then forms a knowledge support for agents and allows them to formulate messages and to reason about messages. Other systems include several ontologies in the MAS functioning.
Elmore et al. [15] uses a set of ontologies for the treatment of heterogeneous data in order to unify them. Specifically they offer a MAS which aligns a view of data from five laboratories. Each agent processes structured data of one laboratory in XML format. It also has an ontology describing the semantics of these data. All agents of the system have a common language of communication. Ontologies allow them to interpret the data sent in messages in order to construct a unified view of data.
Another system using ontologies is COMMA [8]. Each ontology is encapsulated in an agent and, in collaboration with an interface agent, helps the user to add (or to retrieve) documents in a knowledge base. For this, the implemented ontology agent is an aid in order to correctly construct the metadata of the new document or the query.

Evolving ontologies in agents. It is sometimes necessary to modify the ontology used by an agent or by a MAS when the application data evolve. Some works [40] [26] [2] propose to add mechanisms to agents (such as replacing a concept with another according to interactions with other agents) enabling them to modify an ontology. The aim of each agent is then to continue to communicate but not to construct a common view of a domain nor to make evolve an ontology. Each agent then has its own point of view on the application domain. Similarly, Gasser [47] and Viollet [46] propose ontologies alignment tools in order to ensure communications between agents when several ontologies are used. This enables to solve interoperability issues between heterogeneous agents.
Viollet uses ontologies to represent knowledge of agents [46]. To communicate, these agents exchange messages using FIPA-ACL. The semantic content of these messages is expressed using ontologies. Viollet adds then mechanisms of ontologies alignment to agents in order to relate different ontologies and therefore to understand the semantics contained in messages [46].
More recent works have focused on the learning of new concepts using the MAS paradigm. Afsharchi et al. [1] and Safari et al. [35] propose a MAS able to learn new concepts in order to answer to user's requests. The MAS is distributed over a set of sites (several universities). Each agent manages an ontology built from texts describing the proposed university programs (medicine, archeology, engineering, etc..). Each concept is described in extension, namely by the set of instances that describe it. The role of an agent is to answer to users' requests for university courses. To respond, the agent sometimes needs to learn a new concept because the query contains a new knowledge. For this, a learning agent (*learner*

[2]Web Ontology Language `http://www.w3.org/2004/OWL/`

agent) sends a request to teacher agents (*teacher agent*) from other universities that contains the unrecognized concept or instance. The aim of (*teacher agents*) is to return the instances containing the words formed by the concept. Based on these results, the (*learner agent*) determines the largest intersection between the data of each (*teacher agent*) which corresponds to a new concept formed by instances that are shared with the others (*teacher agent*). However, the authors do not specify how the concept is inserted in the ontology.

Multi-agent systems for the management of ontologies Few works use MAS for the construction or evolution of ontologies. MAS is often used as a tool to help the ontologist to seek knowledge or to check the consistency of an ontology. To our knowledge, only [30] used a MAS as an ontology.

Aldea et al. use a MAS within a platform for ontologies evolution [3]. This MAS consists of two types of agents: the *coordinator agent* and the *internet agent* whose roles consist in retrieving, weighting and ranking documents on Internet, in order to help a user to make his ontology evolve. The *coordinator agent* takes as input an ontology. This ontology is split into several parts. Each part contains one or more concepts. Then, the *coordinator agent* sends each part to the *internet agent* whose role is to retrieve pages that contain instances of these concepts. Finally, each *internet agent* returns its results in the form of sorted pages. These results are then presented to the user in order to help him to make his ontology evolve. The MAS role is not to make the ontology evolve but rather to bring documents that may contain new concepts or new instances.

Hadzic et al. provide a system for ontologies evolution using the agent paradigm [22]. Four types of agents have been defined: the *information agent*, the *data warehouse agent*, the *data mining agent* and the *ontology agent*. To make his ontology evolve, a user sends a request to the *information agent*. It is a request for collecting information about the modeled domain. The *information agent* role is to retrieve information on the current domain in databases. Then, it sends these data to the *data warehouse agent* to store them. Upon receipt of new data, the *data warehouse agent* asks the *data mining agent* to process data. The role of this latter is to extract new knowledge by applying *data mining* techniques to identify concepts as well as relations between concepts. Finally, it sends the result to the *ontology agent* that compares new knowledge with the current ontology. If differences exist, the *ontology agent* proposes to the user a list of changes. The user can then accept or reject the proposals. Mechanisms of this system are very similar to those used in [49]. The difference is that [49] do not use agents and rely on ontologies that are available on the Web rather than on databases. Nevertheless, [50] demonstrated that using external data sources becomes ineffective if the domain knowledge are very specific.

The construction and evolution of ontologies are a teamwork involving several people. Often these people work remotely on several versions of the same ontology. To manage this teamwork, several ontologies management collaborative tools using MAS have been proposed.

Bao and Honavar use agents in a collaborative tool for the construction of ontologies [7]. They propose only one type of agent called *interface agent*. It represents a user of the tool. Its role is to ensure the consistency of the modeled ontology according to users' concurrent modifications.

Slimani et al. propose a tool called P^2*OManager* to manage the evolution of an ontology using a MAS [41] [42]. The aim of this tool is to maintain the consis-

tency between an ontology and the dependent ontologies using a set of agents. An agent can have three roles: *ontology agent*, *initiator ontology agent* and *dependent ontology agent*. The objective of these agents is to manage the changes of an ontology and their propagation to dependent ontologies. The role of the *ontology agent* is to detect if the ontology has changed or not. It is a “listener” that perceives the changes made on ontology by a user of the system. When this happens the agent changes its role and becomes *initiator ontology agent*. Its objective is then to detail all the changes that affected the ontology. Then the agent decides to propagate these changes to other agents having the *dependent ontology agent* role. For this, the *initiator ontology agent* checks for every change if there is a link between a modified element in the ontology and the dependent ontologies. If such a link exists, the agent sends a message to request the spread of change to the *dependent ontology agents*. Otherwise, the change is not propagated. Slimani et al. are interested in the propagation of ontologies changes to other dependent ontologies [41] [42]. However, the automatic identification and extraction of new knowledge, as well as the automatic addition of new concept in the ontology are not available in this tool.
To summarize these works, an ontology usually enables communication between agents. To maintain communication when the application domain evolves or when heterogeneous agents are related, some works propose mechanisms of ontology alignments, of replacement of concepts, or of learning of new concepts or instances of concepts. The goal then is not to propose a domain ontology, but rather to enable agents to understand themselves. Other works use MAS in tools for evolution of ontologies. The aim of these MAS is to help a user to find new knowledge or to maintain the consistency of ontologies. The works proposed by [22] and [30] appear to us the most fully developed because they automate the extraction of knowledge and their integration in an ontology. However, for our problem of evolution of ontologies from texts (where texts are very short, with very specific knowledge and where no external knowledge-rich data sources exist) they seem difficult to be used.

3 Adaptive Multi-Agent Systems (AMAS)

The use of Multi-Agent Systems in order to evolve an ontology from texts seems to be a relevant idea. Indeed, a system for evolution of ontologies can be seen as a complex problem whose environment (additions of texts, ontologist’s actions) is dynamic and opened. This system should be able to self-adapt to this environment. Several adaptation techniques exist but are ineffective for our particular problem because of constraints such as the small volume of data in our texts, the impossibility of defining the purpose of the system, etc.. That is why an innovative adaptation using the concepts of emergence and self-organization seems to be relevant.
To solve the problem of ontologies evolution from texts, we need a system whose processings are distributed among several entities, each of which possesses a local view of its environment and is able to interact in an autonomous way to answer to a purpose. The dynamic environment of such a system and the complexity of our problem put in evidence the interest of using the MAS paradigm. The AMAS approach provides the theoretical foundations enabling the construction of such a system.

3.1 Functional adequacy

The main asset of the AMAS approach [11], [20] is to tackle the design of complex systems that can be incompletely specified and for which an *a priori* known algorithmic solution does not exist. It provides an organizational approach enabling the construction of multi-agent systems that continuously and locally self-adapt to the dynamics of their environment. It proposes to conceive an adaptive system while only focusing on the interactions between the system and its environment on the one hand and between the parts (agents) of the system on the other hand. These interactions are based on a local processing of the information by the components of the system that only have a local view of their environment. This principle of locality guarantees the emergent nature of the system functioning.
In this approach, the designer has only to define when and how each agent composing the system has to locally decide to change its interaction links with other agents in order to achieve the expected overall function (from the viewpoint of an external observer who knows its purpose). In that case, the system is said "functionally adequate". We showed in previous works [11], [20] that algorithms, which do not directly depend on the overall function to be obtained, are a solution for dynamically implementing systems able to self-adapt to their contexts. That is why local behaviors we propose to assign to agents do not depend directly on this expected overall function. Each agent, according to its local perception, the local rules it pursues and its local task to be achieved, can change or adjust its interactions with other agents of the system or the environment. The modification of the interactions between the parts of the system will lead to the transformation of the resulting overall function of the system. So, according to interactions between the multi-agent system and its environment, the organization between agents emerges and constitutes an answer to unforeseeable events.

3.2 When does an agent need to self-adapt?

To reach this functional adequacy, we proved [11], [20] that each autonomous agent, which follows a cycle composed of three steps (perception/decision/action), has to keep relations as "cooperative" as possible with its social (other agents) and its physical environment. The definition of cooperation we use is not conventional (resources sharing, common work, etc.); it is a social attitude to which an agent must comply. Our definition is based on three local meta-rules the designer has to instantiate according to the problem to be solved:

- Meta-rule 1 (C_{per}): Every signal perceived by an agent must be understood without ambiguity.
- Meta-rule 2 (C_{dec}): Information coming from its perceptions has to lead the agent to produce a new decision.
- Meta-rule 3 (C_{act}): This reasoning must lead the agent to make actions that have to be useful for other agents and the environment.

An agent, that simultaneously locally checks these three meta-rules, is in a cooperative state. This means that this agent is situated at the best position in the current organization.

$$\text{Cooperation} = C_{per} \wedge C_{dec} \wedge C_{act}$$

On the contrary, an agent that does not locally check at least one of the three previous meta-rules, is facing a "Non Cooperative Situation" (NCS).

These cooperation failures can be assimilated to "exceptions" in traditional programming. Different generic NCSs were then highlighted: *incomprehension* or *ambiguity* if C_{per} is not checked, *incompetence* or *unproductiveness* if C_{dec} is not obeyed and finally *uselessness* or *competition* or *conflict* when C_{act} is not checked. These generic NCSs have to be instantiated according to the problems to be solved. This approach can be qualified as "proscriptive" because each agent in the system has, first of all, to anticipate, to avoid, and to repair the NCS that occur in its environment during the system functioning. Thus, the algorithm of a cooperative agent can be summarized by two main steps: (*i*) when an agent is facing a cooperative situation, it acts according to its partial function; (*ii*) when an agent is facing a NCS, it acts in order to come back to a cooperative state.

3.3 How does an agent self-adapt?

This approach has important methodological implications: designing an AMAS consists in defining and assigning cooperation rules to agents. Concretely, the designer, according to the current problem to solve, has (*i*) to define the nominal behavior of an agent, then (*ii*) to deduce the NCSs to which the agent can be confronted with, and finally (*iii*) to define the actions the agent has perform to come back to a cooperative state and to self-adapt to the environmental dynamics. This self-adaptation of an agent is implemented by two main behaviors [9]. The **nominal behavior** directly related to the partial function of the agent that contributes to the overall emerging function; The **cooperative behavior** which includes the detection of and the resolution of NCS as well as the anticipation, the prevention of the occurrence of NCS. This behavior, responsible of the specific adaptation process of the system, is subdivided into three behaviors:

1. The *tuning behavior* consists in analyzing the nominal behavior calculation in order to find cooperation failures. If there are some, it tries to solve these NCS by modifying the parameters that take an active part in the nominal behavior;
2. The *reorganization behavior* consists in modifying the way in which an agent interacts with its environment and the other agents. This behavior is usually carried out when a uselessness or an incompetence NCS is detected;
3. The *evolution behavior* consists in creating new agents or in suppressing the current agent.

3.4 Methodological impacts

This approach was applied successfully to the resolution of various types of problems related to different fields (user personalization [27], collective robotic [32], etc.). The obtained results encouraged us to promote the use of the AMAS approach and to build a methodology named ADELFE[3] for designing adaptive systems. ADELFE only concerns applications in which self-organization makes the solution emerge from the interactions of their parts. It also indicates to the designer if the use of the AMAS approach is relevant for the construction of his application. If the AMAS approach is relevant, ADELFE helps him to express the behavior of the agents composing the system and the behavior of the society

[3] `http://www.irit.fr/ADELFE`

formed by these agents. ADELFE mainly focuses on the identification of all NCS that may appear during the system functioning and then on the definition of the actions the agents have to perform to come back to a cooperative state.

4 DYNAMO Overview

DYNAMO is an ANR[4] funded research project. Our contribution in this project was to propose a method and a tool that allow the construction and the evolution of Terminological and Ontological Resource (TOR) from a corpus of documents in order to facilitate semantic information retrieval. A TOR is a resource having a conceptual component (an ontology) and a lexical component (a terminology) [29], [13]. A TOR contains not only a set of domain concepts but also a set of associated terms (their linguistic manifestations in documents: every term "denotes" at least one concept). These terms are used to annotate documents in order to do semantic information retrievals. This paper does not propose a new model for the representation of a TOR. Our TOR (called "ontology" in the rest of the paper) is formalized using the OWL-based TOR model and was provided by a partner of the DYNAMO project [34]. The TOR model used is a meta-model in which the OWL ontology concepts and associated terms are OWL classes (figure 1). A "concept" class is denoted by one or more classes "terms". Symmetrically, a "term" class must necessarily have a denotation link toward a "concept" class.

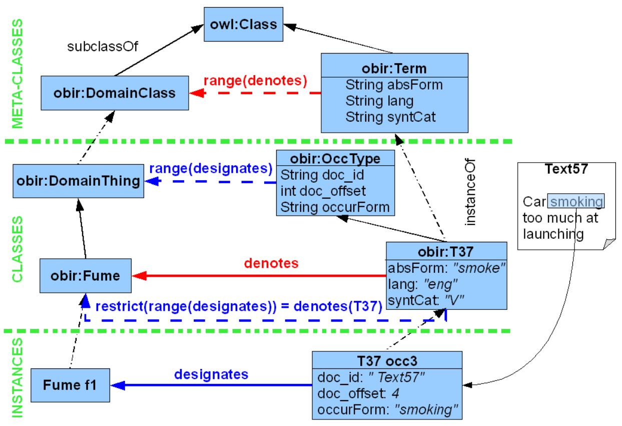


Fig. 1. The DYNAMO TOR model.

The core of our work was the definition and the conception of an AMAS called "DYNAMO-MAS" whose design principles follow the ADELFE methodology. Before detailing our system, we first present the main steps of our TOR evolution process from texts. Thereafter, we present the architecture of DYNAMO-MAS before explaining the cooperative behaviors of agents that we have defined.

[4] `http://www.agence-nationale-recherche.fr/`

4.1 Ontology Evolution Process

According to the principle of *ontological continuity* [48], we assume that the evolution of the domain knowledge does not affect the knowledge previously modeled in a TOR. Changing a TOR (ontology) consists then in adding other relations, other terms and/or other concepts. Our approach consists of 4 main steps (see Fig. 2). Initially, a TOR is modeled from a corpus of text documents. This TOR is consistent with this corpus. The addition of new documents in the corpus triggers the process of TOR evolution.

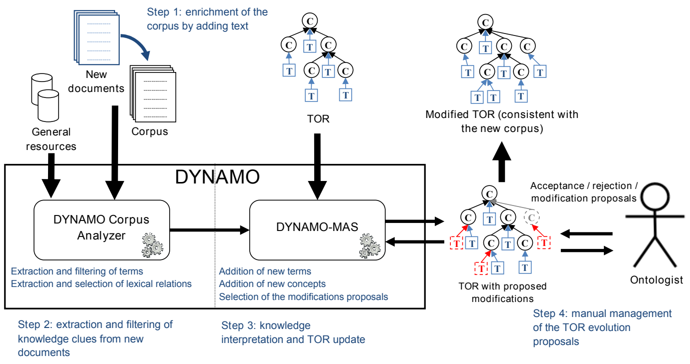

Step 1: enrichment of the corpus by adding text
New documents
General resources
Corpus
TOR
Modified TOR (consistent with the new corpus)
DYNAMO
DYNAMO Corpus Analyzer
DYNAMO-MAS
Extraction and filtering of terms
Extraction and selection of lexical relations
Addition of new terms
Addition of new concepts
Selection of the modifications proposals
Acceptance / rejection / modification proposals
TOR with proposed modifications
Ontologist
Step 2: extraction and filtering of knowledge clues from new documents
Step 3: knowledge interpretation and TOR update
Step 4: manual management of the TOR evolution proposals

Fig. 2. The proposed process for the evolution of an ontology from texts.

Step 1: enrichment of the corpus by adding new text. This step consists in adding new documents to a corpus that will be then annotated by the TOR.
Step 2: extraction and filtering of knowledge clues from new documents of the corpus. This step consists in extracting (from new documents) candidate terms of the domain as well as lexical relations between these terms. This step also consists in identifying, among all the terms and lexical relations proposed by the NLP tools, those that are relevant to keep.
Step 3: knowledge interpretation and TOR update. This step consists in representing the terms and lexical relations (selected in the previous step) in the form of terms, concepts and relations between concepts, and in adding them to the current TOR. To do this, the system uses domain knowledge. Each new term to be proposed has to be connected (by a denotation link) to a concept (already existing or to be created) of the TOR. When a new concept is created, it has to be positioned in the TOR, and thus connected to one of the ontology concepts (if necessary to the TOP concept). To do this, the system uses lexical relations found in corpus, or uses the relations between concepts in a general ontology or in a general structured lexicon (WordNet or Wolf). When all the extracted terms are processed, some new concepts and new terms are selected by the system. The TOR, thus enriched by these most relevant new concepts and terms, is then proposed to the ontologist.
Step 4: manual management of the TOR evolution proposals. This step consists in finalizing the evolution of the TOR. To do this, the ontologist analyzes the TOR modifications proposals. He validates, modifies and / or rejects them

one by one via the TOR editor. He can also “manually” add other concepts and other terms. He can also reorganize some parts of the ontology in order to better structure it. The system takes into account these modifications in order to update its internal representation of the TOR. This process ends when the ontologist is satisfied by the modified TOR and when, from his viewpoint, the TOR considered as “consistent” with the new corpus.
We propose to automate the second and the third steps of this process.

4.2 TOR evolution Architecture

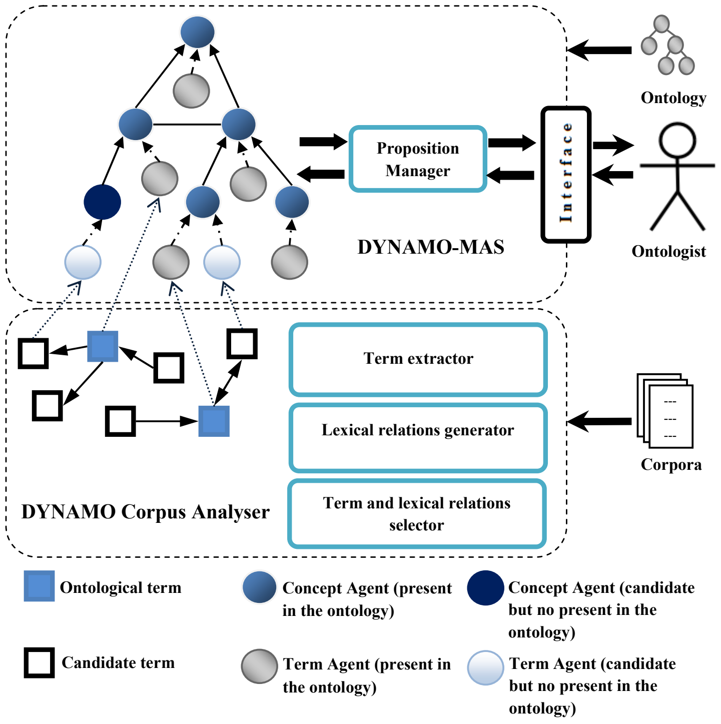

Ontology
Proposition Manager
Interface
Ontologist
DYNAMO-MAS
Term extractor
Lexical relations generator
Term and lexical relations selector
Corpora
DYNAMO Corpus Analyser
Ontological term
Concept Agent (present in the ontology)
Concept Agent (candidate but no present in the ontology)
Candidate term
Term Agent (present in the ontology)
Term Agent (candidate but no present in the ontology)

Fig. 3. DYNAMO architecture.

The DYNAMO architecture (fig. 3) consists of (*i*) a corpus analyzer, (*ii*) an Adaptive Multi-Agent System (DYNAMO-MAS) and (*iii*) a graphical user interface (GUI). The input of DYNAMO is a corpus of documents. The output of DYNAMO is an OWL ontology. The goal of this paper is to focus on the instantiation of the AMAS approach to the ontology evolution issue. That is why only the main characteristics of the Corpus Analyzer and the GUI are given. More information about the corpus analyzer are available in [37].
The goal of the **corpus analyzer** is to identify relevant candidate terms as well as relevant lexical relations that will be later agentified; it prepares the inputs for

MAS. It includes a *terms extractor* named YaTeA [4] a *lexical relations generator* and a *term and lexical relations selector*. In this project, we are interested in four types of lexical relations: (*i*) *Hyperonymy* that expresses a generic-specific relation between terms; (*ii*) *Meronymy* that expresses a part-hood relation between terms; (*iii*) *Synonymy* that relates semantically close terms; (*iv*) Other relations (called *transverse relations*) that are any other kinds of lexical relations that will lead to a specific set of semantic relations, such as *causes, leads_to, etc.*.
These lexical relations are extracted by three ways: (*i*) a lexico-syntactic patterns projection [24]; (*ii*) a syntactic dependency analysis between terms and candidate terms in order to extract hyperonym relations; and (*iii*) a similarity calculation between terms and candidate terms with the Levenshtein distance [28] to compute synonymy relations.
The *corpus analyzer* generates triplets $< T_i$, Rel, $T_j >$ where T_i and T_j are candidate terms or terms (whether the term belongs or not to the ontology) and *Rel* is a lexical relation. Each triplet has a confidence (Q, I) where Q is the quality of the relation (value between 1 and 10) and I is the number of instances of the relations in the corpus. The triplets are the inputs of the MAS.
The **MAS** receives as input, the triplets provided by the *corpus analyzer* and possibly an existing ontology. As output, it provides a modified ontology in the form of an OWL file respecting the used TOR model. DYNAMO-MAS also contains a component called *Nest* (not shown in fig. 3) that is rather technical (it manages the creation of agents and their communications).
A **GUI** is implemented in the ontology editor Protégé[5]. It enables the ontologist to visualize the ontology as well as the MAS proposals. Through this interface, the ontologist can validate, delete or modify the DYNAMO-MAS proposals. He can also manually add other terms, concepts or relations. They will then be added to the MAS and they will lead to new proposals.

5 DYNAMO-MAS: an AMAS for TOR evolution

DYNAMO-MAS consists of two components: (*i*) a MAS that represents the current TOR and (*ii*) a Proposition Manager whose role is to manage the MAS proposals as well as the interactions between the ontologist and the MAS.
We used ADELFE [36] to determine and define the two types of agents composing our MAS: (*i*) *term* agents that represent the terminological component of the TOR and (*ii*) *concept* agents that represent the conceptual part of the TOR.
The initial state of the MAS is an agentified TOR. The concepts of the TOR are *concept* agents connected by conceptual relations. The terms of the TOR are *term* agents connected to *concept* agents by denotation relations. The addition of a new text to the corpus triggers the corpus analyzer that identifies candidate terms and lexical relations between candidate terms and/or terms. The agentified candidate terms as well as lexical relations to be processed are then added to the MAS. When new *term* agents and new *concept* agents appear in the MAS, they have to locally find their best position in the organization. It is the local goal of every agent. To achieve this goal, each agent has a nominal behavior and a cooperative behavior that subsumes the first one according to the AMAS approach (sub-section 3.3).

[5]`http://protege.stanford.edu/`

5.1 Term agent behaviors

Term agents represent the terminological part of a TOR. A *term* agent has a status (*term* or *candidate term*) indicating whether the agent is part of the TOR (that is to say, a valid *term* agent) or is at the proposal stage (an invalid *term* agent). Each *term* agent is linked to other *term* agents in accordance with the lexical relations extracted from the corpus. It must also be linked to at least one *concept* agent according to the TOR model. Each relation between *term* agents is tagged by the confidence of the triplet $< T_i$, Rel, $T_j >$.

***Term* agent nominal behavior** The goal of a *term* agent is to find its best position in the MAS and to propose itself to the ontologist. To do this, it must achieve three objectives: (*i*) to denote a *concept* agent ; (*ii*) to process all its outgoing lexical relations ; (*iii*) if the conditions are met, to propose itself to the ontologist. During its life cycle, each *term* agent processes its goals and the received requests (messages from other *term* agents or *concept* agents), from the highest priority to the lowest priority. A *Term* agent process its outgoing lexical relations from the most confident (most priority) to the less confident (less priority). Its priority objective is to denote a *concept* agent. Once this objective achieved, the next one is determined according to the confidence of the relation to be processed, the relevance of the agent for proposing itself to the ontologist and the confidences of the requests it receives. The algorithm explaining the nominal behavior of a *term* agent can be seen in [37].

To achieve its first objective, a *term* agent asks for the creation of a *concept* agent to the *Nest* tool. This creation is done if, in the current MAS, a *concept* agent having the same label does not exist. The *Nest* tool transmits thereafter the identifier of this new *concept* agent to the *term* agent. Then, the *term* agent sends to the *concept* agent a request for establishing a denotation relation (❶). This request is always accepted by the *concept* agent (❷). The confidence of the denotation relation is equal to the greatest confidence of the lexical relations of the *term* agent.

NCS1: At its creation a *term* agent may be faced with a *uselessness* NCS. Indeed, if a *term* agent is not linked to a *concept* agent, it cannot achieve its objectives; it is therefore useless.
NCS1 solution: To solve this NCS, the *term* agent sends a request to create a *concept* agent to the *Nest* tool. It sends then a denotation request to this *concept* agent. During its lifecycle, the *term* agent may change again its neighborhood or disappear if it becomes again useless.

To achieve its second objective, a *term* agent processes its outgoing lexical relations. A lexical relation has a confidence and a status (*not treated, treated or refused*). A *term* agent processes its relations from the most relevant (having the greatest confidence) to the less relevant. To do this, a *term* agent sends a request to its *concept* agent in order to transform the lexical relation ❸ (fig. 4). The *concept* agent processes the request, then notifies the *term* agent with a message of acceptance or refusal ❹. The *term* agent updates the status of the processed relation (*treated or refused*). If the relation is refused, a *term* agent may later request to process the refused relation if its confidence increased. When a *term* agent asks for a synonym relation ❺ processing (fig. 5), its *concept* agent sends a denotation

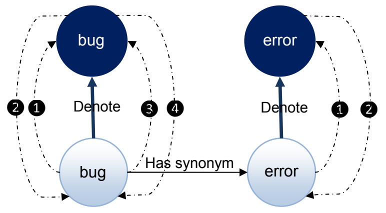


Fig. 4. Interactions between agents.

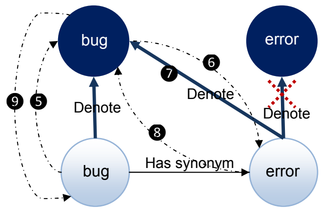


Fig. 5. Interactions between agents.

request ❻ to the target *term* agent of this relation. If the confidence of the request is higher than the current denotation link of the target *term* agent, this latter accepts the request, changes its denotation relation ❼ and notifies the *concept* agent by a message of acceptation ❽. The target *term* agent refuses the request otherwise. The initial *term* agent is then notified ❾.

NCS2: During the processing of a denotion request sent by a *concept* agent, a *term* agent may be faced with an ambiguity NCS. The *term* agent cannot choose the *concept* agent that it wants to denote if the confidence of the request denotation $(Q_1; I_1)$ is equal to its current denotation confidence $(Q_2; I_2)$.
NCS2 solution: A *term* agent prefers a neighborhood having valid *concept* agents. In this way, to solve an ambiguity NCS, it chooses to denote the valid *concept* agent. When the *concept* agent that has sent the denotation request and the *concept* agent that is currently denoted by the *term* agent are both valid or invalid, the *term* agent chooses the agent having the greatest confidence (the most relevant according to it). In case of a tie, the *term* agent refuses the denotation request.

To achieve its third objective, a *term* agent calculates its relevance value (a score between 0 and 10). When this score is above a threshold (fixed to 5 but adjustable), a *term* agent proposes itself to be part of the ontology by sending a request to the Proposition Manager. The ontologist can tune this threshold starting with a low value at the beginning of the evolution process (to get a lot of suggestions) and increasing this value when the ontology provides good annotations for all the documents in the corpus. When the ontologist accepts or rejects a proposal, the Proposition Manager notifies the *term* agent with a rejection or an acceptance. The *term* agent updates then its status. To prevent *term* agents to propose themselves again after a rejection, a high value (equal to 9) is assigned to the ontologist's interventions (this parameter is adjustable). Indeed, during the ontology evolution, the ontologist can qualify a candidate term as irrelevant for the domain and can eventually later revise his decision. In this case, a *term* agent may propose itself again if its relevance value exceeds this reject value.
To summarize, a *term* agent tries to find its best position in the MAS organization by processing lexical relations. It moves from a *concept* agent towards another *concept* agent essentially by processing synonymy relations. To locally assess the adequacy of its position, the *term* agent calculates its relevance:

$$termAgentRelevance = \alpha_1 * P_1 + \alpha_2 * P_2 + \alpha_3 * P_3 + \alpha_4 * P_4$$

where P_1 is the maximum value of all its lexical relations; P_2 is the accuracy of its neighborhood; P_3 expresses the accuracy of the *term* agent's lexical relations;

P_4 expresses the diversity of the *term* agent lexical relations and $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ are the different weights of the P_i.

More precisely:
- P_1 = MaxLexicalRelationConfidence;
- P_2 = (nbTermAgentInTOR - nbTermAgentNotInTOR)/(nbTermAgentInTOR + nbTermAgentNotInTOR);
- P_3 = (nbAcceptedLexicalRelation - nbRefusedLexicalRelation) / (nbAcceptedLexicalRelation + nbRefusedLexicalRelation);
- P_4 = nbDifferenteLexicalRelation / nbAllDifferentLexicalRelation.

After various experiments with DYNAMO-MAS we empirically fixed the values of α_1 to 0.5, α_2 to 2, α_3 to 2 and α_4 to 1. These values best weighed the parameters of the agent relevance.

***Term* agent cooperative behavior** The cooperative behavior of a *term* agent is defined through two cooperative behaviors (according to the AMAS approach): a reorganization behavior and an evolution behavior. The **reorganization behavior** occurs when a *term* agent denotes a valid *concept* agent (or when it receives from its *concept* agent a message telling it is a son of a valid *concept* agent).

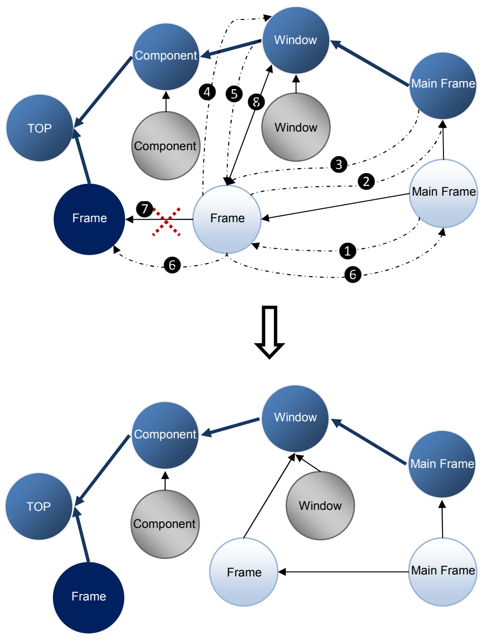


Fig. 6. Cooperative self-organization between *term* agents.

For example (Fig. 6), `Main Frame`-term informs `Frame`-term that it denotes `Frame`-concept (❶). It also sends the relevance score of `Frame`-concept. This information is useful to a *term* agent that is the target of an hyperonymy relation.

Indeed, in order to increase its confidence, `frame`-term may decide, when it receives this message, to move towards another *concept* agent. For this, it asks to `Main Frame`-concept which is its father *concept* agent (❷). `Main Frame`-concept then sends `Window` (❸). `Frame`-term is then faced with three denotation possibilities: either it can decide to not move and to remain linked to `Frame`-concept, or to denote `Main Frame`-concept, or to denote `Window`-concept. Its decision depends then on the following rule:
Rule: A *term* agent prefers denoting a valid *concept* agent than a invalid one because this can increase its relevance.
This rule enables to eliminate `Frame`-concept because it is an invalid *concept* agent. The *term* agent has then to choose between `Main Frame`-concept and `Window`-concept. The agent uses then the following rule:
Rule: A *term* agent prefers to denote a *concept* agent whose label is equal to its. This more semantic rule was incorporated into the *term* agents by considering that a term is often mistaken with a concept. When this rule is not used by the *term* agent, it then uses the more abstract following rule:
Rule: In case of the reception of a denotation request from a *concept* agent, a *term* agent prefers to denote a *concept* agent whose relevance is the greatest (because the new denotation link will have a confidence equal to the confidence of the *concept* agent). The higher the confidence, more it maximizes the confidence of the *term* agent and thus enables it to propose itself to the ontologist. As this rule is based on confidences, an agent may be faced with a NCS.

NCS3: A *term* agent is facing an ambiguity NCS when it cannot choose between the *concept* agents towards whom moving, because their relevance are equal.
NCS3 solution: To solve this NCS, the *term* agent chooses to move towards `Window`-concept and not towards `Main Frame`-concept. This resolution is semantics. Indeed, a hyperonymy relation is often assimilated to an *is_a* relation. In this sense, a *term* agent is more likely to be at a best position if it denotes `Window`-concept.

In fig. 6, `Frame`-term decides to move to `Window`-concept whose relevance $(Q;I)$ is equal to its denotation relation with `Frame`-concept (❹). `Window`-concept accepts the denotation request and sends an acceptance message to `Frame`-term (❺). When it receives this notification and before moving, `Frame`-term informs its neighbors that it moves towards `Window`-concept (❻). This message enables agents to update their knowledge. Finally, `Frame`-term removes the old denotation relation (❼) and creates a new denotation relation with `Window`-concept (❽) with a confidence equal to the confidence it had with the former *concept* agent (because `Frame`-term agent is the initiator of the denotation relation request).
The **evolution behavior** occurs during the detection and elimination of useless *term* agents. According to the AMAS approach, all agents in a MAS must be useful in order to achieve the functional adequacy of the system.

NCS4: An intermediate *term* agent is facing a uselessness NCS when (*i*) it is linked to a *concept* agent denoted by other *term* agents; (*ii*) it is the target of hyperonymy relations and this *term* agent has the same frequency as the *term* agents whom it is target of hyperonomy relations; (*iii*) it is not a target of a hyperonymy relation.

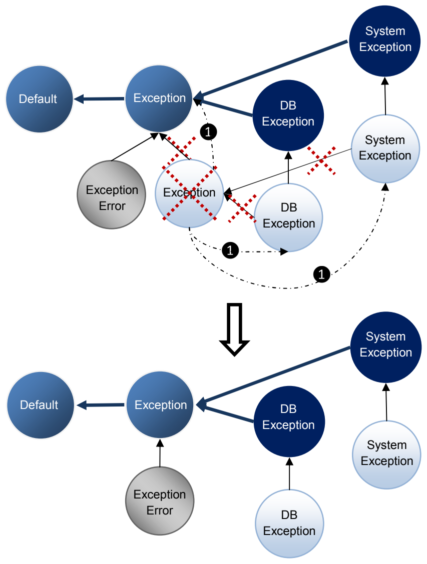


Fig. 7. Local treatment of a uselessness NCS by a *term* agent.

NCS4 solution: When a *term* agent is facing a uselessness NCS, it disappears from the MAS. Before disappearing, a *term* agent informs its neighboring agents in order to allow them to update their knowledge (➊) (Fig. 7).

5.2 Concept agent behaviors

Concept agents represent the conceptual part of a TOR. A *concept* agent has a status (*concept* or *candidate concept*) indicating whether the agent is part of the ontology or is at proposal stage. A *concept* agent is linked to other *concept* agents by conceptual relations. It is also linked to *term* agents by denotation links. Every relation has a status (*not treated, treated or refused*) and a (Q,I) confidence.

***Concept* agent nominal behavior** : The goal of a *concept* agent is to find its *best* position in the MAS and to propose itself to the ontologist. To do this, it must achieve three objectives: (*i*) to have semantic relations with *concept* agents and denotation links with *term* agents; (*ii*) to determine a preferred label and (*iii*) to propose itself to the ontologist. Each *concept* agent processes its goals and the received requests from the highest priority to the lowest priority. The algorithm explaining the nominal behavior of a *concept* agent can be seen in [37].

To achieve its first objective, a *concept* agent has to consider requests from *term* agents ➊) to process lexical relations, and requests from *concept* agents (to establish conceptual relations (➋)). We assumed as in [39], [12] and [5] that to each lexical relation will match a specific conceptual relation. A *hyperonymy* may lead to define an *is_a* relation between the concepts denoted by these terms;

a *meronymy* may lead to define a *part_of* or an *ingredient_of* or a *member_of* relation between the concepts denoted by these terms; a *synonymy* may lead to connect the related terms to the same concept with a *denote* relation; *other relations* may lead to define specific semantic relations between the concepts denoted by the related terms, such as *causes*, *contributes_to*, *affects*, etc..

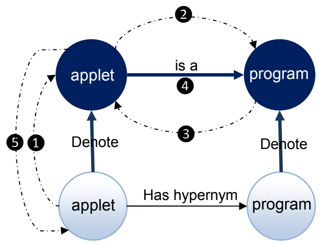


Fig. 8. Establishment of a *is_a* relation.

These principles of transformation are not generalizable to all results provided by the Corpus Analyzer because there may be extraction errors or conflicting lexical relations. A *concept* agent first operates on the basis of these principles, but it may afterwards change its relations and/or delete them. When a *concept* agent processes a request from a *term* agent to process a lexical relation, it sends a request to the concept agents denoted by the *term* agent that is target of the lexical relation (❷). For example (Fig. 8), a request to process a hyperonymy is sent to `applet`-concept. This message corresponds to a request to establish an *is_a* relation (❷). This request is sent by `applet`-concept towards `program`-concept agent (❷).
According to the AMAS approach, a cooperative agent must anticipate the appearance of NCSs that may affect the agent or its neighborhood. That is why, when a *concept* agent receives a request to process a lexical relation, it anticipates the appearance of conflicting relations:

1. A hyperonymy relation must not be processed if: (*i*) a *concept* agent is target of a *is_a* (resp. *has_part*) relation with the *concept* agent denoted by the *term* agent that is target of the hyperonymy relation and (*ii*) if that hyperonymy relation has a confidence $(Q_1;I_1)$ lower than the confidence $(Q_2;I_2)$ of the *is_a* (resp. *has_part*) relation. For example, `applet`-concept will not process the hyperonymy if it is the target of a *is_a* (resp. *has_part*) relation with `program`-concept that has a higher confidence.
2. A meronymy relation must not be processed if: (*i*) a *concept* agent is target of a *is_a* (resp. *has_part*) relation with the *concept* agent denoted by the *term* agent that is target of the meronymy relation and (*ii*) if that meronymy relation has a confidence $(Q_1;I_1)$ lower than the confidence $(Q_2;I_2)$ of the *is_a* (resp. *has_part*) relation. For example, `applet`-concept will not process the meronymy if it is the target of a *is_a* (resp. *has_part*) relation with `program`-concept that has a higher confidence.

A *concept* agent may have to choose between two conflicting relations in order to keep only one. When two conflicting relations have the same confidence (very rare), the agent detects a NCS.

NCS5: A *concept* agent is facing an ambiguity NCS concerning two conflicting relations R_1 and R_2 if the confidence of R_1 is equal to the confidence of R_2.
NCS5 solution: If among the relations R_1 and R_2 one and only one is a hyperonymy relation to be processed (resp. to be kept) or is an *is_a* relation, the *concept* agent will then prefer to process (resp. keep) this relation.

A *concept* agent prefers hierarchical relations (hyperonymy or *is_a*) compared to non-hierarchical relations (meronymy or *has_part*). This rule enables to solve the NCS. When this decision is semantically false, it will be checked again by the *concept* agent if the confidences of the lexical relations processed by the *term* agents that are targets of conceptual relations are reassessed when new texts are added to the corpus.
If a *concept* agent refuses a request to process a lexical relation, it sends a notification message to the *term* agent with a confidence equal to the confidence $(Q;I)$ of the conflicting relation. For example, in Fig. 8, `applet`-concept sends to `program`-concept a request for the establishment of an *is_a* conceptual relation (❷). This latter can accept or reject it by sending a notification to the agent (❸). A *concept* agent may refuse the establishment of a conceptual relation. Indeed, some combinations of conceptual relations are wrong and must not be established between *concept* agents. For this, a *concept* agent has the following knowledge: (*i*) an *is_a* relation is not symmetric;(*ii*) a *has_part* relation is not symmetric; (*iii*) an *is_a* relation between two *concept* agents cannot exist with a *has_part* relation between these two agents.
A request to establish a conceptual relation will be accepted only if this relation has a confidence $(Q;I)$ bigger than the confidence of the conflicting conceptual relation. In case of equal confidences, a NCS is detected.

NCS6: A *concept* agent A_1 is facing to an ambiguity NCS concerning two conceptual relations (it is the target of the first relation R_1 and the second relation, R_2, comes from a request for a conceptual relation establishment from a *concept* agent A_2), if both relations have equal confidences.
NCS6 solution: The A_1 agent compares its relevance P_1 with the relevance P_2 of the agent A_2. If P_2 is higher than P_1 then the A_1 agent accepts the establishment of the relation, otherwise it refuses. When these relevance are equal, the *concept* agent refuses the establishment of the required relation. This relation will be asked again if the lexical relations that led to its apparition in the AMAS evolve when new texts are added to the corpus.

When it receives a notification, the *concept* agent updates the status of the concerned relation as well as its relations with *concept* agents. In our example, *applet*-concept creates an *is_a* relation with *program*-concept (❹). As, in our AMAS, a *concept* cannot be polysemous, the establishment of an *is_a* relation leads to the displacement of a *concept* agent *A* of an old *concept* agent *B* towards another *concept* agent *C*. Finally, the *term* agent that is target of the lexical relation is notified by the *concept* agent that established the conceptual relation (❺).
Finally, the processing of synonymy relations leads to the displacement of *term* agents from a *concept* agent towards another. In this case, it is possible that a *concept* agent has no denotation relation. This is a NCS, because without a *term*

agent a *concept* agent cannot pursue its objectives.

<u>**NCS7:**</u> A *concept* agent is facing a uselessness NCS if it has no denotation relation.
<u>**NCS7 solution:**</u> The *concept* agent disappears from the MAS. Before disappearing, it informs the single *concept* agent to which it is connected with a *is_a* relation.

To achieve its second objective (to determine a preferred label), a *concept* agent choses the label of the *term* agent having the denotation relation with the highest confidence. This label can evolve if new *term* agents denote the *concept* agent or if the confidences of the denotation links evolve.

To achieve its third objective (propose itself to the ontologist), a *concept* agent calculates its relevance value. When this value is above a threshold, a *concept* agent proposes itself to be part of the ontology by sending a request to the Proposition Manager.
An invalid *concept* agent considers itself as relevant if two criteria are checked:
1. The *concept* agent believes it is at the best position in the AMAS organization. This criterion is estimated by the parameters P_1, P_2, P_3, P_4 ;
2. the *concept* agent believes that it is an interesting *concept* agent to add to a TOR, that is to say it is closer to the "valid" status than the "Invalid" status. This criterion is estimated by the parameter P_5.

A *concept* agent processes its local relevance according to the following formula:

$$conceptAgentRelevance = \alpha_1 * P_1 + \alpha_2 * P_2 + \alpha_3 * P_3 + \alpha_4 * P_4 + \alpha_5 * P_5$$

where P_1 is the maximum confidence value of all its conceptual relations; P_2 is the accuracy of the *concept* agents that are in relation with; P_3 expresses the accuracy of the conceptual relations of the *concept* agent ; P_4 is the depth in the ontology; P_5 is the proportion of relevant *term* agents that denote this *concept* agent and $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$ are the different weights of P_i. More precisely:
- P_1 = MaxConceptualRelationConfidence;
- P_2 = (nbRCAInTOR - nbRCANotInTOR) / (nbRCAInTOR + nbRCANotInTOR) where RCA (Related Concept Agent) are *concept* agents in relation with the evaluated *concept* agent;
- P_3 = (nbAcceptedCRA - nbRejectedCRA) / (nbAcceptedCRA + nbRejectedCRA) where CRA (Conceptual Relations of the Agent) are conceptual relations known by the evaluated *concept*;
- P_4 = {-1;1}: -1 if the *concept* agent is connected to the TOP agent of the ontology, 1 otherwise;
- P_5 = (nbRelevantTermAgent - nbNotRelevantTermAgent) / (nbRelevantTermAgent + nbNotRelevantTermAgent) where the *term* agents are denoting the *concept* agent.

After some experiments with DYNAMO-MAS we empirically fixed the values of α_1 to 0.5, α_2 to 1, α_3 to 1, α_4 to 1 and α_5 to 2.

***Concept* agent cooperative behavior** The cooperative behavior of a *concept* agent is defined through three cooperative behaviors (according to the AMAS approach): two reorganization behaviors and an evolution behavior.

The first **reorganization behavior** is similar to the one of *term* agents. When a *concept* agent is connected to a valid *concept* agent, it informs its neighbors of this information. This helps *term* agents or *concept* agents to move in the AMAS. *Rule:* When a *concept* agent is connected to another valid *concept* agent, it informs its neighbors of this information.

The second **reorganization behavior** concerns *concept* agent moving with transverse relations. To allow a *concept* agent to change its neighborhood thanks to its transverse relations, we took inspiration from the differentials principles [6]. These principles enable to identify one unit (or concept), its significance according to its identities (or similarities) and its differences with its neighbors.

In the DYNAMO project, each TOR is built around a core of stable concepts (2, 3 or 4 concepts) linked to the *DomainThing concept* agent by an *is_a* relation and connected amongst themselves by properties. So the addition of other concepts to the core concepts leads to the evolution of the TOR. For the AMAS, the goal of a *concept* agent is then to find under which core *concept* agent it has to be located. For that, when a *concept* agent has transverse relations from which it is the source or the target and when it is connected to the *DomainThing concept* agent, it moves in order to increase its relevance. For this, it chooses its transverse relation whose confidence is maximal. Then it compares this relation with the transverse relations of the core *concept* agents in order to choose the agent toward which to move. Finally, it sends a request to establish a *is_a* relation with a confidence equal to the confidence of the transverse relation with the concerned *concept* agent (❶) (Fig. 9). The latter accepts the establishment of the relation and sends a notification of acceptance (❷). Once the notification received, the *concept* agent changes its neighborhood by adding this new *concept* agent. This behavior can trigger a NCS if a *concept* agent has two or more transverse relations with the same confidence.

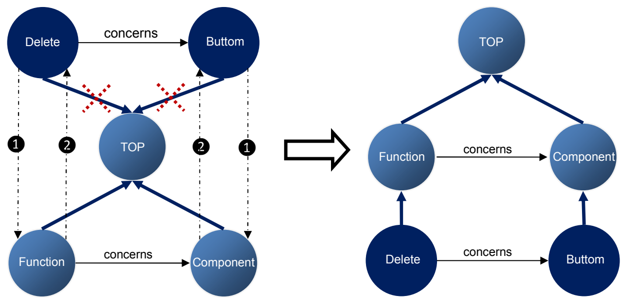


Fig. 9. Self-organization with transverse relations.

NCS8: A *concept* agent is facing an ambiguity NCS if it cannot choose between two or more transverse relations in order to change its neighborhood.

NCS8: To solve this NCS, a *concept* agent chooses the transverse relation whose end is a valid *concept* agent. If all the *concept* agents are valid or invalid, it chooses the relation with the *concept* agent that has the highest confidence. Finally, if all the confidences are equal, the *concept* agent chooses a arbitrary transverse relation. Thus, by changing its neighborhood, a *concept* agent increases its relevance and therefore increases its chance to propose itself to the ontologist.

The **evolution behavior** aims at preventing useless intermediate *concept* agents.

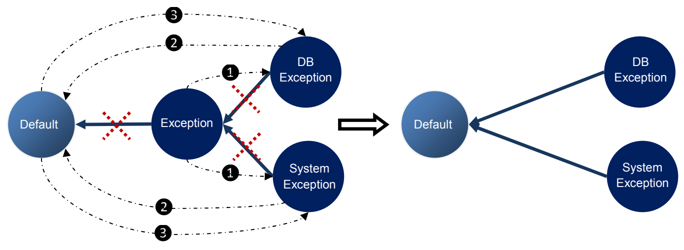


Fig. 10. Example of interaction for the simplification of the hierarchy.

NCS9: An intermediate *concept* agent is facing a uselessness NCS if it has the same frequency[6] as its father *concept* agent and it has the same frequency as its son *concept* agent(s).
NCS9 solution: A *concept* agent faced with a uselessness NCS disappears from the AMAS.

An intermediate *concept* agent informs its neighboring agents before disappearing in order to allow them to update their knowledge (❶) (Fig. 10). This disappearance can lead to a uselessness NCS. To avoid this, the *concept* agent proposes its father *concept* agent to its son *concept* agents as well as to its *term* agents that denote it (❶). Each agent then sends a request to establish a relation with this new *concept* agent (❷). The latter accepts the establishment of the relation and it sends a acceptance notification to the concerned agent (❸).

5.3 The Proposition Manager

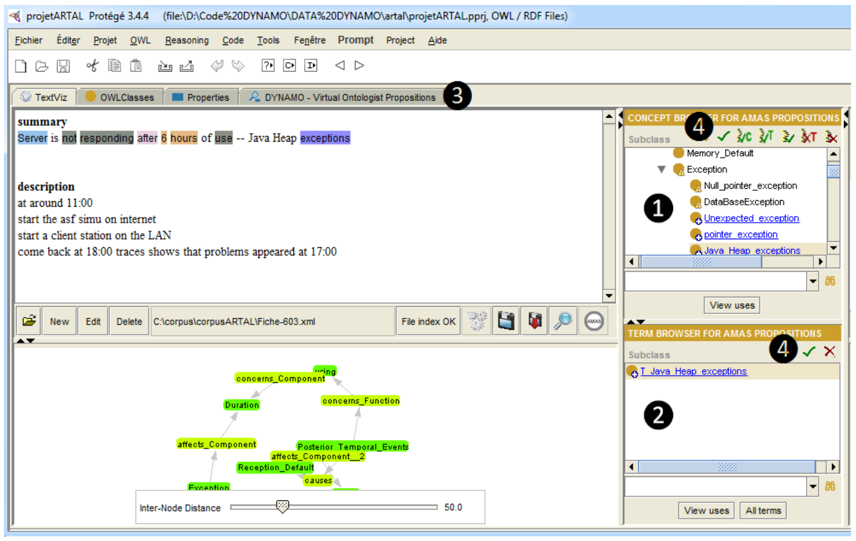
The Proposition Manager enables the ontologist to visualize the ontology and the MAS proposals as well as to interact with the MAS (Fig. 3). Its main goals are (*i*) to sort out the proposals (to be displayed *via* the GUI) sent by the *concept* agents and the *term* agents; and (*ii*) to convey the corrections made by the ontologist to the *concept* and *term* agents with regard to their proposals.
Once the activity of the MAS is stabilized, i.e. when all the agents have processed the requests that they received, the Proposition Manager sorts the proposals, deletes contradictory ones and conveys the final list to the ontologist. Some contradictions may appear either when an agent proposes itself but later disappears from the MAS, or when an agent proposes a conceptual relation and later proposes another relation involving the same concepts with a different relation. The system considers that the most recent one is the only one that is valid.

[6]The frequency of a *concept* agent is calculated from the sum of the frequencies of *term* agents that denote it.

The Proposition Manager finally sends back to agents that made proposals, the evaluation coming from the ontologist. This notification corresponds to a new request sent to the agents that will process it.

6 Evaluation of the DYNAMO-MAS

The DYNAMO-MAS was implemented as a plugin in the ontology editor Protégé[7] (Fig. 11). It completes TextViz [34], a tool dedicated for the semantic annotation of documents.
When the ontologist adds new texts to the corpus, the DYNAMO-MAS is triggered. The corpus analyzer extracts new candidate terms and new lexical relations. It sends them as inputs to the MAS; new term agents are created and interact with the other agents. So most of the agents in the MAS update their knowledge, react and communicate with each other until a stable state is reached. Then the MAS proposes a new ontology (the initial ontology that has been enriched and modified) to the ontologist in the GUI (Fig. 11) (❹. Changes are displayed in the concept panel (❶) and in the terms panel (❷). Proposed concepts and terms are the underlined one. A second Tab Widgets, called *DYNAMO - Virtual Ontologist Proposals* (❸), has been added to Protégé. It is a tabular view of the MAS proposals, incorporating non-hierarchical relations that cannot be seen in the two first panels. The ontologist validates, deletes or modifies the concepts, terms and relations proposed by DYNAMO-MAS *via* the Graphical User Interface (❹).



summary
Server is not responding after 6 hours of use -- Java Heap exceptions

description
at around 11:00
start the asf simu on internet
start a client station on the LAN
come back at 18:00 traces shows that problems appeared at 17:00

Fig. 11. The DYNAMO tool into the Protégé ontologies editor.

We tested our MAS on 3 different TOR and associated corpus provided by 3 partners of the DYNAMO project: (*i*) an English one on software bugs reports (made

[7] http://protege.stanford.edu/

up of 887 terms and 582 concepts; the corpus is composed of 287 documents _bug report files_ provided by Artal technology), (*ii*) a French one on automotive diagnosis (made up of 579 terms and 330 concepts; the corpus is composed of 710 documents _files that report fault descriptions and repair procedures_ provided by Actia) and (*iii*) a French one on archaeology (made up of 733 terms and 380 concepts; the corpus is composed of 299 documents _rule-based formulation of scientific papers_ provided by Arkeotek).

To evaluate the quality of DYNAMO-MAS we made a comparison between manual ontology evolution and automatic ontology evolution. The evolution is manual if the ontologist adds by itself new terms and new concepts in the ontology. The evolution is automatic, if the ontologist uses our tool. Obtained results can be seen in [37].
After the addition of 21 documents in the Artal corpus, DYNAMO-MAS provided **67%** of term proposals and **56%** of concept proposals that are relevant. It was also able to suggest 12 terms and 9 concepts not manually identified by the ontologist. After the addition of 12 documents in the Arkeotek corpus, DYNAMO-MAS provided **68.75%** of relevant term proposals and **59.26%** of relevant concept proposals. It was also able to suggest 18 terms and 14 concepts not identified. Finally, after the addition of 21 documents in the Actia corpus, DYNAMO-MAS reached only **16.98%** of relevant term proposals and **22.22%** of relevant concept proposals. It was also able to suggest 5 terms and 5 concepts not manually identified. These results are less good because new documents added to the corpora contained very little new knowledge. Furthermore, conversely to Artal and Arkeotek ontologies (that are ontologies under construction), Actia ontology covers most of the knowledge expressed in the corpus.
However, obtained results are encouraging and the AMAS approach seems relevant. Even if DYNAMO-MAS is composed of a large number of agents (more than 1000 agents), the new *concept* agents and *term* agents, with only local and distributed mechanisms, come up to find by themselves their best position in the MAS organization (the ontology). Results of the quality evaluation of DYNAMO-MAS are more precisely presented and discussed in [37].
To evaluate the performances of DYNAMO-MAS we added different numbers of documents (4, 8, 16, 32, 64, 128, 256 and 512) to the 3 ontologies. We studied the **time performance** of DYNAMO-MAS to stabilize and its **scalability**. Indeed, when new agents are added to the MAS, this latter becomes perturbed. The goal of the MAS is then to solve its perturbations in order to come back to a stable state and give rise to a new ontology. Initially, after the agentification of the 3 ontologies, the ontology of Artal was composed of 1469 agents, the one of Arkeotek 1328 and the one of Actia 1176.
Fig. 12 shows the time required for the MAS to self-stabilize after the addition of new documents. The 3 curves having a X^2 coefficient close to zero, the time required for the MAS to stabilize is almost linear. The addition of 512 documents to the Arkeotek corpus has generated the creation of 1468 new agents in the MAS and the stabilization time took around 4 minutes and 30 seconds. The addition of 512 documents to the Artal corpus has generated the creation of 694 new agents and the stabilization time was around 1 minute. Finally, the addition of 512 documents to the Actia corpus has generated the creation of 270 new agents and the stabilization time was around 20 seconds. We can conclude that DYNAMO-MAS stabilizes very quickly despite the large number of agents composing the MAS.

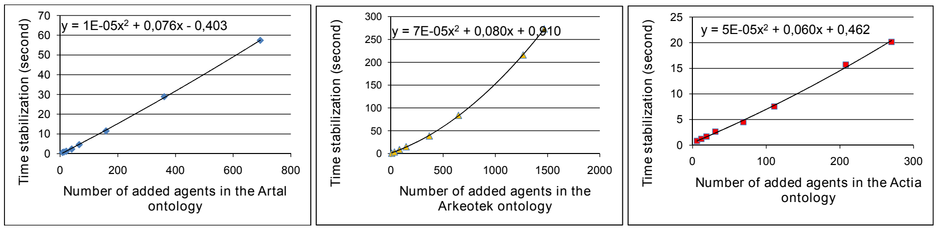


Fig. 12. Stabilization time of the MAS and number of added agents further to the addition of new documents.

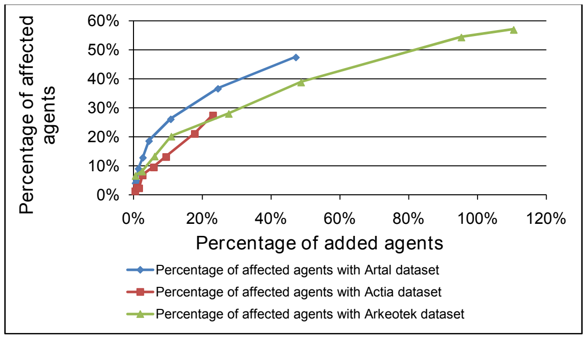


Fig. 13. Perturbation diffusion inside DYNAMO-MAS.

Fig. 13 shows the percentage of perturbed agents after the introduction of new agents. In the 3 curves, we can notice that the addition of new agents to the MAS does not disrupt the whole system. When we added 512 documents to the Arkeotek corpus, the 1468 new agents (representing **110,47%** of the MAS) perturb only 755 agents (representing **56,85%** of the MAS). Also, when we added 512 documents to the Artal corpus, the 694 new agents (representing **47,24%** of the MAS) perturb only 696 agents (representing **47,38%** of the MAS). Finally, when we added 512 documents to the Actia corpus, the 270 new agents (representing **23,14%** of the MAS) perturb only 319 agents (representing **27,34%** of the MAS). This result shows that the disturbance can be considered as local.

As we seen, results provided by DYNAMO-MAS have a different quality depending on the corpus and on the initial state of the ontology. It seems that as the ontology construction progresses, the insertion of results supplied by DYNAMO-MAS becomes more complex, or conversely when the conceptualization is in its early stages, the MAS brings more help to the ontologist. In other words, the results of the qualitative evaluation of DYNAMO show that more an ontology is unachieved, more the AMAS proposals are accepted (Artal and Arkeotek data). Conversely, if an ontology is already built, the AMAS proposals are often rejected (Artal data). The completion of an ontology can be quantified thanks to the annotations of new documents added to the corpus. These values are calculated by TextViz and indicate the degree of annotation of a document by the ontology.

We are currently working on some improvements of DYNAMO-MAS in order to better adapt the results provided by the MAS to the ontologist, that is to say to personalize the system to the ontologist's actions. The idea here is to adjust the threshold proposals of agents depending on the state of completion of the ontology as well as according to the ontologist's actions.
We propose to take into account the results of annotation for the calculation of the proposals threshold.

$$InitialProposalThreshold = \frac{\sum_{i=1}^{n} annotationValue_i}{n}$$

This formula increases the proposals threshold when documents are rightly annotated and decreases otherwise. Thus, for the data provided by Actia, only the *term* agents and the *concept* agents having a relevance value higher than 9 will propose themselves to the ontologist.
Thereafter, the system has to learn how to vary this threshold according to the interactions with the ontologist. For that, we consider that more the ontologist accepts proposals more the un-proposed agents are potentially interesting. Thus, the threshold has to be decreased in order to propose them to the ontologist. Conversely, more the ontologist refuses proposals, more it makes sense to limit the number of proposals. The solution is then to increase the threshold. We propose to implement this adaptation mechanism with the following formula:

$$ProposalThreshold = ProposalThreshold - ProposalThreshold * Coef$$

Where ProposalThreshold is the current proposals threshold and agentRelevancy is the confidence of the accepted/refused/moved agent by the ontologist. The value of the $Coef$ variable has to be different according to the ontologist's actions. It can have a value equal to 0.5 when the ontologist accepts a proposal (proposal at the right position in the TOR). It can have a value equal to 0.25 the ontologist moves a proposal (proposal correct but misplaced in the TOR).
When the ontologist refuses a proposal the ProposalThreshold can be defined with the following formula where $coef$ can be equal to 0.5:

$$ProposalThreshold = ProposalThreshold + ProposalThreshold * Coef$$

Subsequently after a given number of the ontologist's actions, the Proposition Manager updates the proposals made by the ontologist by eliminating those that do not exceed the proposals threshold and by adding those that exceed it. A new updated version of the TOR is then proposed to the ontologist. Another perspective, is to introduce an Adaptive Value Tracker (AVT) component [27] to adjust the value of the $Coef$ parameter. An AVT is a software component that finds the optimal value of a dynamic variable in a given space thanks to successive feedbacks. In our case, feedbacks come from the ontologist. When the ontologist rejects or accepts a set of concepts, relations and terms, the AVT will adjust $Coef$.

7 Conclusion

The aim of this paper was to propose a system to automatize the co-construction and the evolving (with an ontologist) of an ontology from texts. This issue is very

interesting because manual construction and evolution of an ontology are complex and consuming tasks. The goal of our system was to automatically propose to the ontologist new concepts and/or terms to improve the ontology after the addition of new documents in the corpus. The ontologist can then make corrections on the ontology to be taken into account at runtime by the system in order to improve the ontology. After having proposed a state of art on this issue, and considering the characteristics of such issue, we justified why the Multi-Agent System paradigm and especially the Adaptive Multi-Agent System (AMAS) approach seem very relevant to resolve this problem. The AMAS approach proposes an original solution to solve complex problems evolving in a dynamic environment and for which an a priori know solution does not exist.
The core of this paper was to present the instantiation of this approach to the problem of co-construction and evolution of an ontology from texts with an ontologist. We gave in that sense a quick overview of the DYNAMO project in which this work took part as well as the architecture of the proposed system (DYNAMO-MAS). Then we focused on the cooperative behaviors we gave to the two types of agents we defined. In accordance with the AMAS approach, we put the emphasize on the Non Cooperative Situations to which each type of agents can be confronted with during the system functioning, as well as the actions they have to implement to come back to a cooperative state. Some experiments that were carried out within the project were then presented, with three ontologies more or less achieved, expressed in two different languages (French and English), concerning three different domains. These experiments confirmed that linguistic clues are not sufficient to decide the content of an ontology, and that the intervention of an ontologist is fundamental. We proposed then some improvements that are currently on-going in order to better co-construct the ontology, that is to say to personalize the results provided by DYNAMO-MAS with the actions of an ontologist (who can be more or less strict).
The originalities of our work are triple. First it enables to make evolve an ontology when new documents are added in the corpus without restarting the process of construction from scratch. Also, our proposition considers an ontology as a MAS that interacts with an ontologist and self-adapts when new domain knowledge are added. Finally our approach is independent of the language and the domain of the handled texts and it expresses the ontology in OWL format, a standard that makes it easily reusable.
Acknowledgments: We thank all the members of the DYNAMO project for their contribution and especially S. Rougemaille[8] and M. Mbarki[9] for their contribution to the implementation and the evaluation of the DYNAMO tool.

References

1. M. Afsharchi and B. H. Far. Automated ontology evolution in a multi-agent system. In *1st international conference on Scalable information systems*, InfoScale ’06, New York, NY, USA, 2006. ACM.
2. T. M. Akinsola. Automated Ontology Evolution. Masters of science informatics, University of Edinburgh, Edinburgh, Scotland, 2008.

[8]Unsolved Problems for Emerging Technologies - Upetec company
[9]Artal Technologies company

3. A. Aldea, R. Bañares-alcántara, J. Bocio, J. Gramajo, and D. Isern. An ontology-based knowledge management platform. In *Workshop on Information Integration on the Web associated to IJCAI*, pages 177–182, 2003.
4. S. Aubin and T. Hamon. Improving term extraction with terminological resources. In *Advances in Natural Language Processing, 5th Int. Conf. on NLP, FinTAL*, pages 380–387, Turku, Finland, 2006. Springer.
5. N. Aussenac-Gilles and N. Hernandez. Du linguistique au conceptuel : identification de relations conceptuelles à partir de textes. In *Atelier "Acquisition et modélisation de relations sémantiques", Toulouse*, 2009.
6. B. Bachimont. Engagement sémantique et engagement ontologique: conception et réalisation d'ontologies en ingénierie des connaissances. *Ingénierie des Connaissances: Evolutions récentes et nouveaux défis*, 1:1–16, 2000.
7. J. Bao and V. Honavar. Collaborative ontology building with wiki@nt. *Workshop on Evaluation of Ontology-Based Tools*, 2004.
8. F. Bergenti, A. Poggi, G. Rimassa, and P. Turci. Comma: a multi-agent system for corporate memory management. *Int. Joint Conf. on AAMAS*, pages 1039–1040, 2002.
9. C. Bernon, D. Capera, and J.-P. Mano. Engineering Self-Modeling Systems: Application to Biology. In *Int. Workshop on Engineering Societies in the Agents World, St-Etienne*, LNCS 5485, pages 236–251. Springer, 2009.
10. P. Buitelaar, P. Cimiano, and B. Magnini. *Ontology Learning from Text: Methods, Evaluation and Applications*. Frontiers in Artificial Intelligence and Applications Series. IOS Press, Amsterdam, 2005.
11. V. Camps. *Vers une théorie de l'auto-organisation dans les systèmes multi-agents basée sur la coopération : application à la recherche d'information dans un système d'information répartie*. PhD thesis, Université Paul Sabatier, Toulouse, Janvier 1998.
12. M. Chagnoux, N. Hernandez, and N. Aussenac-Gilles. An interactive pattern based approach for extracting non-taxonomic relations from texts. In *Workshop on Ontology Learning and Population (associated to ECAI 2008)*, pages 1–6. University of Patras, juillet 2008.
13. P. Cimiano. *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications*. Springer, 2006.
14. P. Cimiano and J. Volker. Text2onto - a framework for ontology learning and data-driven change discovery. In *10th Int. Conference on Applications of Natural Language to Information Systems*, volume 3513 of *LNCS*, pages 227–238, Heidelberg, 2005. Springer.
15. M. T. Elmore, T. E. Potok, and F. T. Sheldon. Dynamic data fusion using an ontology-based software agent system. *7th World Multiconference on Systemics, Cybernetics and Informatics*, 2003.
16. G. Flouris. *On belief change and ontology evolution*. PhD thesis, University of Crete, Department of Computer Science, Heraklion, Greece, 2006.
17. G. Flouris, D. Plexousakis, and G. Antoniou. A classification of ontology change. In *CEUR-WS 201*. CEUR-WS.org, 2006.
18. F. Gandon. *Distributed Artificial Intelligence and Knowledge Management: ontologies and multi-agent systems for a corporate semantic web*. Thèse de doctorat, Univ. de Nice - Sophia Antipolis, novembre 2002.
19. P. Gawrysiak, G. Protaziuk, H. Rybinski, and A. Delteil. Text onto miner - a semi automated ontology building system. In *the 17th Int. Symposium on Methodologies for Intelligent Systems*, pages 563–573, Toronto, 2008.

20. M.-P. Gleizes, V. Camps, J.-P. Georgé, and D. Capera. Engineering Systems which Generate Emergent Functionalities. In *Engineering Environment-Mediated Multiagent Systems - associated to ECCS, Dresden, Germany*, number 5049 in LNAI, page (on line). Springer-Verlag, juillet 2008.
21. D. Greenwood, M. Lyell, A. Mallya, and H. Suguri. The IEEE FIPA Approach to Integrating Software Agents and Web Services. *AAMAS*, 2007.
22. M. Hadzic and D. Dillon. An agent-based data mining system for ontology evolution. In *Confederated International Workshops and Posters on the Move to Meaningful Internet Systems*, OTM '09, pages 836–847, Berlin, Heidelberg, 2009. Springer-Verlag.
23. Z. S. Harris. *Mathematical Structures of Language*. Wiley, New York, 1968.
24. M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *14th Int. Conference on Computational Linguistics*, pages 539–545, 1992.
25. M. Klein. *Change management for distributed ontologies*. PhD thesis, Dutch Graduate School for Information and Knowledge Systems, Germany, 2004.
26. S. Leen-Kiat. Multiagent distributed ontology learning. In *Workshop on Ontologies in Agent Systems, associated to AAMAS*, volume 66, pages 75–79, Bologna, Italy, July 2002.
27. S. Lemouzy. *Systèmes interactifs auto-adaptatifs par systèmes multi-agents auto-organisateurs : application à la personnalisation de l'accès à l'information*. Thèse de doctorat, Univ. Paul Sabatier, Toulouse, juillet 2011.
28. V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.
29. A. Maedche. *Ontology learning for the Semantic Web*, volume 665. Kluwer Academic Publisher, 2002.
30. K. Ottens. *Un système multi-agent adaptatif pour la construction d'ontologies à partir de textes*. Thèse de doctorat, Univ. Paul Sabatier, Toulouse, octobre 2007.
31. K. Ottens, N. Hernandez, M.-P. Gleizes, and N. Aussenac-Gilles. A multi-agent system for dynamic ontologies. *Journal of Logic and Computation, Special Issue on Ontology Dynamics*, 19:1–28, 2008.
32. G. Picard and M.-P. Gleizes. Cooperative Self-Organization: Designing Robust and Adaptive Robotic Collectives. In *Third Eur. Workshop on Multi-Agent Systems , Brussels, Belgium*, pages 495–496, Brussel, 2005. KVAB.
33. M.-L. Reinberger and P. Spyns. Discovering knowledge in texts for the learning of dogma-inspired ontologies. In *Workshop on Ontology Learning and Population*, ECAI04, pages 19–24, Valencia, 2004.
34. A. Reymonet, J. Thomas, and N. Aussenac-Gilles. Modelling ontological and terminological resources in OWL DL. In *OntoLex07 - associated to ISWC*, Busan, 2007.
35. L. Safari, M. Afsharchi, and B. H. Far. Concepts in action: Performance study of agents learning ontology concepts from peer agents. In *ICAART'09*, pages 526–532, 2009.
36. Z. Sellami. *Gestion dynamique d'ontologies à partir de textes par systèmes multi-agents adaptatifs*. Thèse de doctorat, Université de Toulouse, juillet 2012.
37. Z. Sellami, V. Camps, and N. Aussenac-Gilles. DYNAMO-MAS: a multi-agent system for ontology evolution from text. *Journal on Data Semantics*, Volume 2, Issue 2(DOI 10.1007/s13740-013-0025-1):145–161, mai 2013.

38. Z. Sellami, V. Camps, N. Aussenac-Gilles, and S. Rougemaille. Ontology Co-construction with an Adaptive Multi-Agent System: Principles and Case-study. In *Knowledge Discovery, Knowledge Engineering and Knowledge Management*, volume 128 of *CCIS*, pages 237–248. Springer, 2011.
39. P. Séguéla. *Construction de modèles de connaissances par analyse linguistique de relations lexicales dans les documents techniques*. Thèse de doctorat, Université Paul Sabatier, Toulouse, mars 2000.
40. R. Siebes and F. van Harmelen. Ranking agent statements for building evolving ontologies. *Workshop on Meaning Negotation, in conjunction with the Eighteenth National Conference on Artificial Intelligence*, July 2002.
41. S. Slimani, S. Baina, and K. Baina. A framework for ontology evolution management in ssoa-based systems. *Web Services, IEEE International Conference on*, 0:724–725, 2011.
42. S. Slimani, S. Baïna, and K. Baïna. Interactive ontology evolution management using mutli-agent system: A proposal for sustainability of semantic interoperability in soa. In *WETICE*, pages 41–46, 2011.
43. L. Stojanovic. *Methods and Tools for Ontology Evolution*. PhD thesis, Karlsruhe University. Germany, 2004.
44. V. Tamma and T. Bench-Capon. An ontology model to facilitate knowledge-sharing in multi-agent systems. *Knowl. Eng. Rev.*, 17:41–60, March 2002.
45. P. Velardi, R. Navigli, A. Cucchiarelli, and F. Neri. Evaluation of ontolearn, a methodology for automatic learning of domain ontologies. In *Ontology Learning from Text: Methods, Evaluation and Applications*. IOS Press, Amsterdam, 2005.
46. A. Viollet. Un protocole entre agents pour l'alignement d'ontologies. Rapport de master, Université Joseph Fourier, Grenoble, 2004.
47. J. Wang and L. Les Gasser. Mutual online ontology alignment. In *Workshop on Ontologies in Agent Systems, associated to AAMAS*, volume 66, pages 103–113, Bologna, Italy, July 2002.
48. D. N. Xuan, L. Bellatreche, and G. Pierra. Un modèle à base ontologique pour la gestion de l'évolution asynchrone des entrepôts de données. In *Modélisation, Optimisation et Simulation des Systèmes : Défis et Opportunités)*, pages 1682–1691, Rabat, Maroc, 2006.
49. F. Zablith, M. Sabou, M. d'Aquin, and E. Motta. Ontology evolution with evolva. In *6th European Semantic Web Conference*, volume LNCS 5554, pages 908–912, Berlin, Heidelberg, 2009. Springer-Verlag.
50. F. Zablith, Z. Sellami, M. D'Aquin, N. Aussenac-Gilles, and N. Hernandez. Vers la combinaison de deux techniques d'évolution d'ontologies à partir de ressources générales et de ressources linguistiques. In *Atelier "Evolution d'ontologies" des 21e Journées francophones d'Ingénierie des Connaissances, Nîmes*, 2010.