A Multi-Agent System for Dynamic Ontologies

Kévin Ottens, Nathalie Jane Hernandez, Marie-Pierre Gleizes, Nathalie Aussenac-Gilles

A Multi-Agent System for Dynamic Ontologies

KEVIN OTTENS, NATHALIE HERNANDEZ, MARIE-PIERRE GLEIZES, and NATHALIE AUSSENAC-GILLES, *IRIT, Université de Toulouse, 118 route de Narbonne, F-31062 Toulouse Cedex, France*
E-mail: ottens@kde.org, {hernande, gleizes, aussenac}@irit.fr

Abstract

In the article, we present Dynamo (an acronym of DYNAMic Ontologies), a tool based on an adaptive multi-agent system to construct and maintain an ontology from a domain-specific set of texts. The originality of our proposal is that the adaptative multi-agent system is used both to represent the ontology itself and to produce the ontology. This enables us to propose a system building and maintaining dynamically an ontology according to interactions with the user (also called the ontologist). We present our system and the mechanisms used to build and maintain the ontology from the texts and for the interactions with the ontologist. We also give results of the evaluation of our system.

1 Introduction

It is well established that ontologies are needed for the semantic web, for knowledge management or B2B applications. However, manually building an ontology is a slow, tedious, costly, complex and time-consuming process. Several approaches have been proposed for building ontologies automatically or semi-automatically, using mainly ontology learning techniques from text [14, 15]. Currently, a real challenge lies in proposing techniques and systems for keeping ontologies up-to-date in changing environments. It is thus necessary to create what is called dynamic ontologies [12].

In this article, we consider that 'ontology dynamics' deals with the evolution of the ontology content according to new knowledge representation needs. Making changes in the ontology can be necessary when the domain covered by the ontology evolves. This may imply modifying the ontology lexicon (adding, removing terms) or modifying the ontology structure (adding, moving, removing concepts or relations). We believe that the changes to be made in the ontology can be captured both from text analysis and from interactions with domain experts. As has been shown for ontology building, texts are a powerful input for extracting knowledge [5]. However, interaction with the experts is crucial when all the information is not stated in the texts. Moreover, we are convinced that experts must be able to supervise the evolution of the ontology in order to model the ontology according to their view of the knowledge.

Our research focuses on Dynamo (an acronym of DYNAMic Ontologies), a tool based on an adaptive multi-agent system to build and maintain an ontology from a domain-specific set of texts. The aim of our system is not to build an exhaustive, general ontology but a domain-specific one. We propose a semi-automated tool since an external resource is required: the 'ontologist'. An ontologist is a kind of cognitive engineer, or analyst, who uses information from texts and interviews with experts to design ontologies. Our purpose is to reduce the ontologist's work by helping him construct and update the ontology from texts in a coherent manner, by continually working with the same tool. To meet this objective, we propose to consider a multi-agent system.

The originality of Dynamo is that the adaptive multi-agent system is used both to represent the ontology itself and to produce the ontology. This aspect has two main advantages. First, the ontologist can go back to the reference texts analysed for building the ontology and modify the model automatically proposed. The system will take into consideration the modifications and compute a new ontology, which means a stable state of the agent system. Second, the ontologist can decide to update the knowledge represented in the ontology according to specific needs, either by introducing new texts to analyse or by modifying the ontology directly. Again, the system will propagate the impact of the modifications in the agent system until it reaches a stable state and forms a new ontology. From our knowledge, multi-agent systems have not been used up to now to build or maintain an ontology.

This article is organized as follows. First, we present our system. We explain why we propose to represent the ontology as an adaptive multi-agent system. After giving an overview of the system, we describe the different mechanisms used to build and maintain the ontology. These mechanisms are based on statistical and syntactical considerations. We present the classification distributed algorithm, which enables the organization concepts according to the occurring context of terms in the texts. We enrich this algorithm by taking into account new criteria such as the syntactical dependencies between terms. Second, we present an evaluation of our system. The evaluation deals with both the performance of our system in terms of algorithm complexity and the use of our system on a real case study. Finally, we compare our approach with related work.

2 Dynamo: multi-agent system for dynamic ontologies

In the following, we first present the motivation for considering the ontology as a multi-agent system. We then give an overview of the system. The last three sections are dedicated to the presentation of the mechanisms used to construct and maintain the ontology: we present the algorithm proposed for organizing concepts, the criteria considered and a simplification proposed for visualising the results.

2.1 The ontology as a multi-agent system

The adaptive multi-agent system is used to represent the ontology itself and to produce the ontology. This choice is motivated by the qualities offered by this kind of system: they can facilitate the interactive design of a functionally adequate system [10] (in our case, a conceptual network), they enable its incremental construction by progressively taking into account new data (in our case, data resulting from the textual analysis and from the user's interaction) and they can be distributed easily on a computer network.

In this first step of our work, we concentrate on the hierarchical components of the ontology, i.e. the concept taxonomy. In Dynamo, each agent is a computing entity representing a concept of the network. Its autonomous behaviour consists in finding its place in the network (or the ontology) according to a local view on its neighbours. All the agents have communication abilities and algorithms enabling them to move closer or further. The core of the agents' behaviour is inspired by classification algorithms.

Classical hierarchical classification approaches do not allow revision of results: when a hierarchical level is reached, it cannot be modified without losing intermediate results. To enable the ontologist to revise his choices at any time in function of the current result, we propose a new distributed classification algorithm relying on different rules. This algorithm is presented in Section 2.3 and it is completed with another set of rules taking more constraints into consideration in Section 2.4.

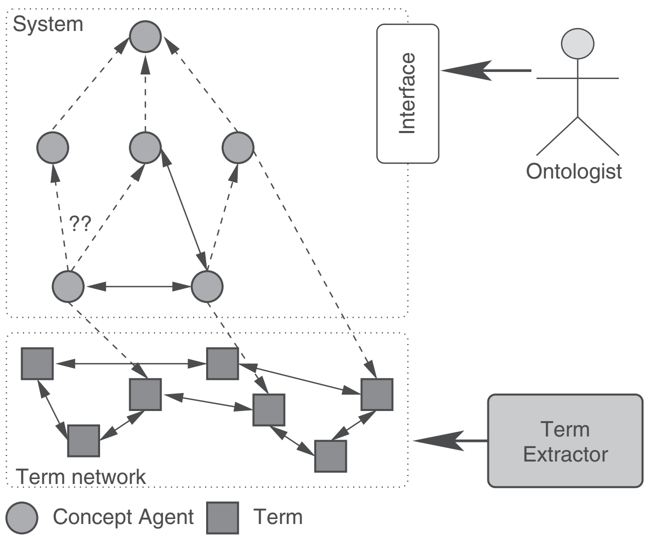

FIGURE 1. Dynamo system architecture

2.2 Overview of Dynamo

2.2.1 Proposed architecture

In this section, we present the architecture of Dynamo.

The system is composed of three main parts (cf. Figure 1):

- the term network obtained as output of a term extractor used to pre-treat the textual corpus. The term network is composed of nominal phrases in their lemmatized form (such as 'viral hepatitis' in example below). These data are not modified by the system;
- the multi-agent system that uses the term network to carry out a hierarchical classification in order to obtain the taxonomy. The system's agents cooperate in order to place themselves in the hierarchy according to the data and the ontologist's indications. It is the final state of the system that constitutes the taxonomy. As the system is the taxonomy and as it carries the construction process, the taxonomy can be maintained easily;
- the interface enabling the ontologist to visualize and control the classification process and its results.

2.2.2 Input and output

As input, Dynamo uses the results of terminological and syntactic analysis of texts carried out by a term extractor. The term extractor Syntex, notably used for ontology construction [2] was selected for its soundness and the large quantity of linguistic information extracted. The tool can also perform incremental analysis. This enables us to add new inputs to our system when new documents are considered to update the ontology. Syntex extracts the syntactic dependencies between terms and performs distributional analysis. We focused on the 'head expansion' network this tool produces. It has been proven that the use of this kind of network could be interesting for classification purposes in [1]. In this kind of network, each term extracted from the corpus

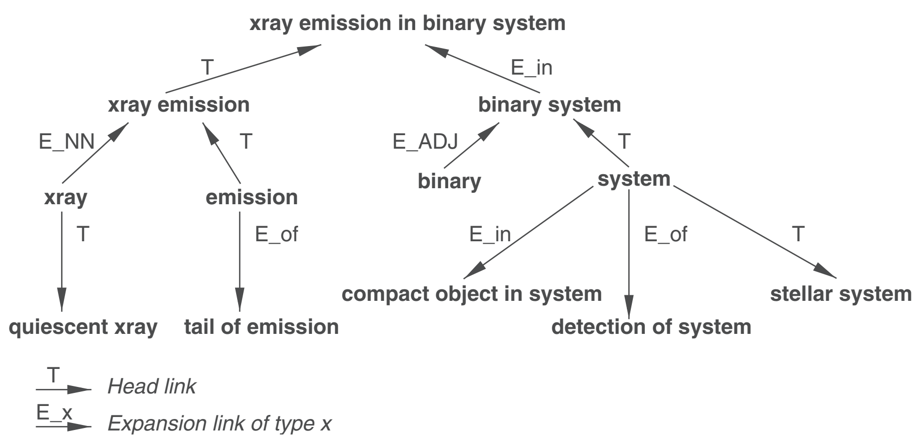

FIGURE 2. Extract of a 'head-expansion' network

is linked, on the one hand, to its head[1] and its expansion,[2] and, on the other hand, to all the other terms for which it is itself head or expansion. For example, 'classification centralised algorithm' has as head 'centralised algorithm' and as expansion 'classification'. In the same way, 'centralised algorithm' is composed of 'algorithm' and 'centralised'. For a more complete example, see Figure 2.

The network produced by the term extractor is stored by the system in a database. For each pair of terms, we suppose that a similarity value can be computed in order to use it in a classification process [1, 8]. The similarity allows to evaluate how two individuals in the same population are similar. Several similarity measures exist in the literature. According to the nature of the data, we focus on similarities described by binary variables, i.e. an individual is described by the presence/absence of a set of characteristics [17]. In the case of terms, they are usually described according to their use context in the corpus. With the syntex 'head-expansion' network, the context is identified by the expansion of what the considered terms are head. As output, Dynamo gives the ontologist a hierarchical organization of concepts (the multi-agent system itself) that can be validated, refined or modified until a satisfying state of the semantic network is obtained. This organization reacts dynamically to the decisions of the ontologist.

2.2.3 Behaviour overview

In order to produce this output, our system has to exploit the 'head-expansion' network and the information it provides as clues for concepts to be close to each other or not. Each agent of the system will try to compare itself with other agents to decide if it has to stay close to them or not regarding the 'is-a' hierarchy.

This kind of process is somehow similar to a classification process, but with two major differences. First, our process has to be distributed in order to take into account user modifications and

[1]That is the maximal sub-phrase situated in the head of the term.
[2]That is the minimal sub-phrase situated in the end of the term.

the resulting knowledge changes in the system more easily. Second, most hierarchical classification processes produce binary trees while a taxonomy is a n-ary tree.

In the following, we argue that the path of least resistance to obtain an adequate system in this context is to first concentrate on distributing the classification process—hence producing a binary tree—and then to complete it with specific facilities to refine the binary tree into a taxonomy. The use of a multi-agent system, and of condition/action rules to describe the agents behaviour, give us enough modularity to allow this incremental development of our system.

2.3 Hierarchy based on a classification process

2.3.1 Classification

In order to improve the understanding of our distributed classification algorithm and in perspective of its evaluation in Section 3.1, we recall here the algorithm used for ascendant hierarchical classification in a non-metric space. The similarity measure considered is symmetric (as the measures used by our system) [17].

In Algorithm 1, at each clustering step, the pair of the most similar elements is determined. Both elements are grouped and the resulting cluster is inserted in the list of the remaining elements. The algorithm stops when the list does not contain any element.

The resulting hierarchical structure is necessarily a binary tree as grouping is done two by two. Moreover, grouping the most similar elements is equivalent to moving them away from the least similar ones. Our distributed algorithm is designed based on those two facts. It is executed concurrently in each of the agents of the system.

Algorithm 1: Centralized hierarchical ascending clustering algorithm

```
Data: List $L$ of items to organize as a hierarchy
Result: Root $R$ of the hierarchy
while length($L$) $> 1$ do
    $max \leftarrow 0$;
    $A \leftarrow$ nil;
    $B \leftarrow$ nil;
    for $i \leftarrow 1$ to length($L$) do
        $I \leftarrow L[i]$;
        for $j \leftarrow i+1$ to length($L$) do
            $J \leftarrow L[j]$;
            $sim \leftarrow$ similarity($I, J$);
            if $sim > max$ then
                $max \leftarrow sim$;
                $A \leftarrow I$;
                $B \leftarrow J$;
            end
        end
    end
    remove($A, L$);
    remove($B, L$);
    append(($A, B$), $L$);
end
$R \leftarrow L[1]$;
```

Note that in the following to evaluate two algorithms, we use an average aggregation strategy [17]. It gives the following formula for computing the similarity between two groups of terms (of cardinality N and M) $\{i_n; n \in [1,N]\}$ and $\{j_m; m \in [1,M]\}$:

$$sim(\{i_n\},\{j_m\}) = \sum_{p=1}^{N}\sum_{q=1}^{M} \frac{sim(i_p,j_q)}{|\{i_n\}| \times |\{j_m\}|}$$

Moreover, we use the similarity measure presented in [1], which gives us the following formula for computing similarity between two terms i and j :

$$sim(i,j) = \frac{\alpha}{2}\left(\frac{a}{a+b} + \frac{a}{a+c}\right) + \frac{1-\alpha}{2}\left(\frac{d}{d+c} + \frac{d}{d+b}\right)$$

where a, b, c and d are, respectively, the number of contexts shared by i and j, present only for i, present only for j and not present for i and j. The contexts are determinded by exploring the 'head and expansion' network [1]. In our case, the contexts of a term correspond to the set of expansions of the terms of which it is the head. For example, in Figure 2, the contexts of 'system' are 'stellar' and 'binary'. For our experimentations, we fix α at $0,75$. All these choices impact the resulting tree, but do not impact the general process of the algorithm, nor its complexity.

2.3.2 A distributed classification algorithm

This section presents the distributed classification algorithm used in our system. We first present the initial state of the system, and then we describe the three steps of the process.

The system is initialized as follows:

- an agent *TOP* having no parent is created, it will be the root of the taxonomy;
- for each term given as input, we create a concept-agent whose goal is to position itself in the hierarchy. These initial agents have thus a term of the network as referent and the concept *TOP* as parent.

Initially, we thus create as many agents as terms given as input plus one for the *TOP*. Once this first structure is defined, the algorithm is processed in parallel by each agent until a global equilibrium position is obtained. A first version of the taxonomy is then presented to the user. The taxonomy modifications then made by the user will reactivate the agents concerned and the algorithm will be reprocessed locally.

The first step of the process (Figure 3) takes place in parallel in the agents (here, we focus uniquely on A_k) having one brother (as we wish to build a binary tree). The agent A_k sends a message to its parent P giving the list of its brothers by decreasing order of dissimilarity (here, we only represent A_1 as the most dissimilar). P receives a message of the same kind from each of its children. In the following, this kind of message will be named 'vote'.

When P has received the message of all its children, it executes the second step (Figure 4). Thanks to the messages indicating the preferences of its children, P can constitute three subgroups among its children:

- the child which globally gives problems to its brothers, i.e. the child which wins the 'election' when the 'votes' are taken into account by a Condorcet vote process [7]. According to the Condorcet criterion, the candidate which has won all the confrontations when expressing the

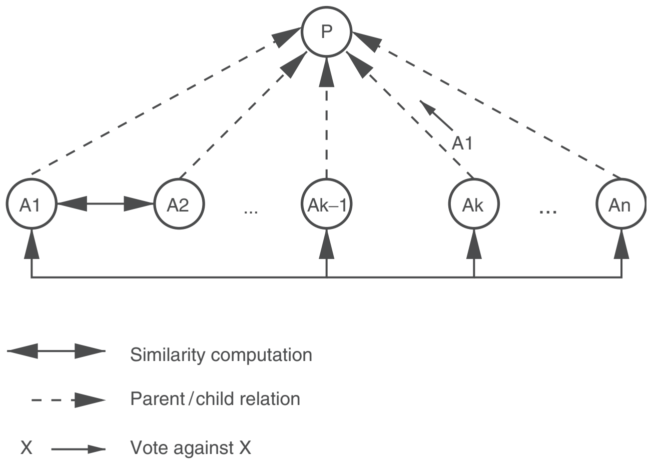

FIGURE 3. Distributed clustering: step 1

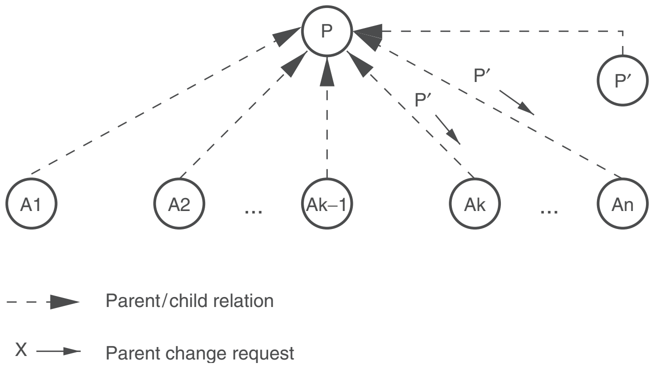

FIGURE 4. Distributed clustering: step 2

preferences must be elected. In the case of a circular conflict, P looks for the candidate which has won most confrontations and if there is a draw, one of the *ex aequo* is chosen randomly (in practice, this latter case has never occurred). In our example, the winner is A_1;

- the children having allowed the ‘election’ of the first group, i.e. the agents having chosen their brother in the first group as being the most dissimilar (here A_k to A_n);
- the remaining children (here A_2 to A_{k-1}).

P creates a new concept-agent P' (having as parent P) and asks the agents of the second group (here the agents A_k to A_n) to make it their new parent.

Finally, Step 3 (Figure 5) is quite obvious. The children pushed away by P (here the agents A_k to A_n) take into account its message and choose P' as new parent. The hierarchy is then built from one relative to the next.

Note that this algorithm is bound to converge as the number of brothers of an agent decreases. When an agent does not have any brothers, its activity is stopped (although it keeps processing

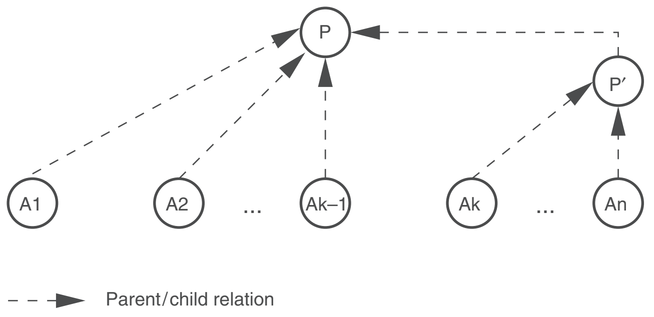

FIGURE 5. Distributed clustering: step 3

messages coming from its children). This property is dependent on the voting system that could not determine a winner. Initially, the winner was determined by the 'majority' and failed in $\sim$30% of cases. The choice of the Condorcet system with the process for resolving conflicts described above has eliminated this problem. The expression of the preferences by each agent has made the algorithm necessarily convergent. When using one vote by adhesion, the quantity of information is not enough to obtain a conclusion in all cases.

The algorithm is really a distributed hierarchical classification algorithm as decisions are made according to negotiations within groups of autonomous agents. As the treatment and the knowledge are local to agents, and as the communication is done by sending messages, such an algorithm can be distributed on a computer network. Moreover, the distribution of both the knowledge and the control enables the ontologist to act on the process whenever he wants. At any time, the system can be stopped to modify the current state. For example, if the user intervenes between Step 1 and Step 2 of the algorithm to move one of the A_k elsewhere in the hierarchy, and when the system restarts P will ignore the vote coming from the child that has moved. It can thus follow the deliberation process for constituting groups during Step 2.

2.3.3 Qualitative aspects

The qualitative aspects of Dynamo derive from the behaviour of its agents.

The main benefit of using an adaptive multi-agent system for a classification task is its dynamic feature. The ontologist can intervene to make corrections and the hierarchy is adapted according to his requests. This characteristic is particularly interesting in the context of knowledge management. As the building process relies on a statistical approach, the ontologist may want to modify the automatically constructed ontology. When going back to the texts to examine the terms contexts [2], the ontologist will be able to interpret their real meaning and revise the system's proposals.

It is difficult to achieve revisions with a centralized approach. In most cases, the reasoning step which has led to the wrong result must be identified and the corresponding class has to be modified manually. This is why a system such as ASIUM [8] presents to the user the classes created at each step. The user–system collaboration is intended to eliminate this problem. Moreover, the complete hierarchy is visible only at the end of the process. The user can see the errors when it is too late to modify the process.

Dynamo runs the distributed classification algorithm, but it is possible for the user to stop and restart it whenever he wants to. At any time, the user can visualize the hierarchy that he can modify.

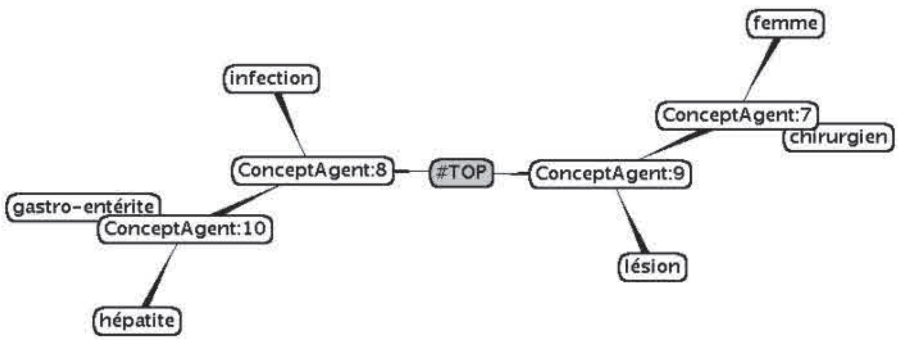

FIGURE 6. Concept agent tree after autonomous stabilisation of the system

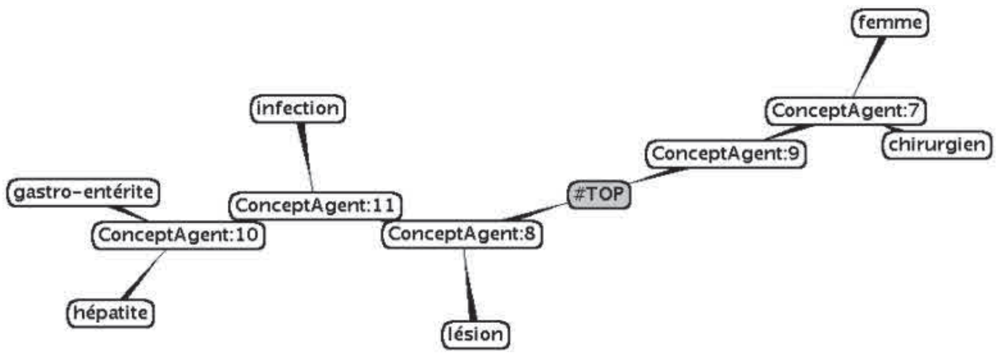

FIGURE 7. Concept agent tree after ontologist modification

His actions will restart the algorithm locally until a new stable state is obtained: this is how Dynamo provides a real ontology design process.

EXAMPLE

To illustrate our approach, we present a simple example thanks to screenshots of the Dynamo interface. In the example, the system runs on a medical-related corpus.

By letting our system work by itself, we obtain the structure presented in Figure 6 after stabilization. It is clear that the concept described by the term 'lésion' (lesion) is misplaced. The similarity computations place it closer to 'femme' (woman) and 'chirurgien' (surgeon) than to 'infection', 'gastro-entérite' (gastro-enteritis) and 'hépatite' (hepatitis). This wrong position for 'lesion' is explained by the fact that without ontologist input the reasoning is only done on statistical criteria.

The user intervenes to place the concept in the right branch by giving it the 'ConceptAgent:8'[3] as new parent. The system then reacts by itself to refine the classification tree and goes back to a binary tree by creating the 'ConceptAgent:11'. The new stable state is then the one presented in Figure 7.

[3] 'ConceptAgent:X' is the name given automatically to an agent that is not directly represented by a term.

2.3.4 Scalability

The 'system–ontologist' coupling is necessary to create the ontology. No adjustment of the algorithm is required as each agent is autonomous and communicates with its neighbours by sending messages. The algorithm is conceptually distributed and is de facto distributable on a machine network. Here again, no adjustment of the algorithm is required. However, it does imply modifying the communication layer and the management of agent creation from the current implementation. It is totally conceivable to execute each agent on different machines. Moreover, because there are current parallel computation capabilities such as those provided by grid computing, the algorithm provided is scalable.

2.4 Multi-criteria hierarchy

In the previous sections, we have considered that the similarity could be computed between any pair of terms. But, when we consider real data, this property is no longer verified. Some terms have no similarity with any other extracted terms. Moreover, for the taxonomy leaves, it is, in some cases, interesting to use other ways to position the concepts. For the fine organization of the leaves, ontologists base their choice on simple heuristics. In the 'head expansion' network, the relation exploited can be quite close to the 'generic/specific' relation. Ontologists thus consider this relation to position the concepts. If the terms representing two concepts are linked by a 'head' relation, then the two concepts can be linked by the 'is a' relation. Through this observation, we have proposed a set of rules that are not based on similarity to structure the leaves of the taxonomy.

2.4.1 Adding 'head coverage' rules

We propose to let agents act from a local point of view by analysing the parent/child relations. Each agent can try to determine if its parent is adequate. It is possible to guess this because each concept agent is described by a set of terms and thanks to the 'head-expansion' term network.

In the following, T_X will be the set of terms describing the agent concept and $head(T_X)$ the set of all the terms that are head of at least one element of T_X. Thanks to these two notations, we can propose the adequacy function of a parent $a(P,E)$ between a parent P and a child E:

$$a(P,E) = \frac{|T_P \cap head(T_E)|}{|T_P \cup head(T_E)|} \tag{1}$$

The best parent for E is the agent P which maximises $a(P,E)$. The higher $a(P,E)$, the greater the coverage between the terms P (T_P) and the terms' head of E ($tete(T_E)$). However, the concept parent is adequate when the set of its terms is a subset of the set of the head terms of its child, the number of terms not shared being minimal. An agent dissatisfied by its parent can then decide to find a better one by evaluating a set of candidates. We have proposed an algorithm to conduct this search.

RULE 1
When an agent E is dissatisfied by its parent P, it evaluates $a(B_i,E)$ with all its siblings (named B_i). The one maximizing $a(B_i,E)$ is then chosen as new parent.

The behaviour of this rule is illustrated on an example. Figure 8 shows the state of the system after the stabilization on our test data set. We can note that 'viral hepatitis' is still under the taxonomy root. This is due to the fact that no similarity can be computed between the term 'viral hepatitis' and the other terms of the agent concepts.

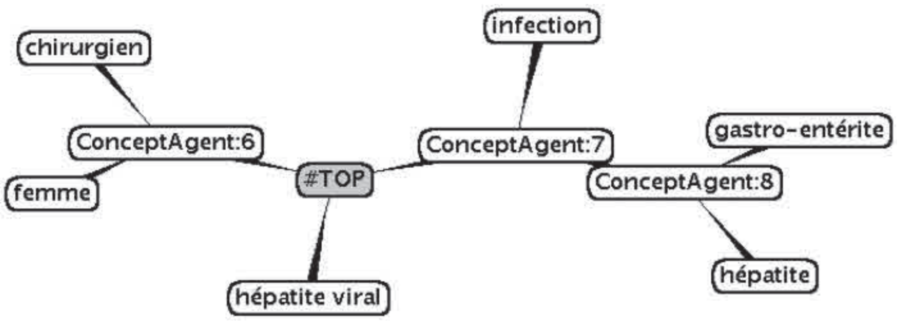

FIGURE 8. Concept agent tree after autonomous stabilization of the system without head coverage rule

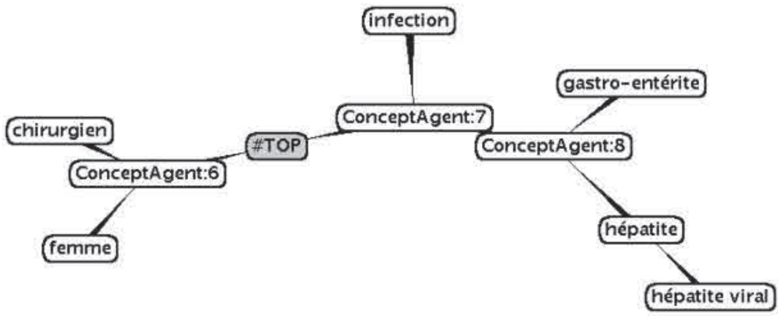

FIGURE 9. Concept agent tree after activation of the head coverage rule

After activating the head coverage rule and letting the system stabilise, we obtain Figure 9. Note that 'viral hepatitis' has moved to the 'hepatitis' branch and has chosen it as new parent. This rule tends to push agents described by a set of terms towards the taxonomy leaves. This achieves the enhancement we expected.

2.4.2 On using several criteria

In the previous sections, we have used only one criterion at a time. The distributed classification algorithm tends to introduce new levels in the taxonomy, while the head coverage algorithm pushes some agents towards the taxonomy leaves. Obviously this raises the issue of how to deal with multiple criteria when constructing the taxonomy and how agents are supposed to determine their priority at a given moment.

The solution we have chosen is based on our concern to minimalize the non-cooperation in the system according to the AMAS approach [10]. Each agent computes three non-cooperation degrees and chooses its current priority depending on which degree is the highest. For a given agent A having a parent P, a set of brothers B_i and which received a set of messages M_k having the priority p_k, the three non-cooperation degrees are:

- $\mu_H(A) = 1 - a(P, A)$ is the 'head coverage' non-cooperation degree, determined by the head coverage of the parent;

- $\mu_B(A) = max(1 - similarity(A, B_i))$ is the 'brotherhood' non-cooperation degree, determined by the worst brother of A regarding similarities;
- $\mu_M(A) = max(p_k)$ is the 'message' non-cooperation degree, determined by the most urgent message received.

Then, the non-cooperation degree $\mu(A)$ of agent A is:

$$\mu(A) = max(\mu_H(A), \mu_B(A), \mu_M(A)) \tag{2}$$

Then, we have three cases determining which kind of action A will choose:

- if $\mu(A) = \mu_H(A)$ then A will use the head coverage algorithm detailed in Subsection 2.5;
- if $\mu(A) = \mu_B(A)$ then A will use the distributed clustering algorithm (Section 2.3);
- if $\mu(A) = \mu_M(A)$ then A will process M_k immediately in order to help its sender.

These three cases summarize the activity of our agents: they must find their best parents ($\mu(A) = \mu_H(A)$), enhance the ontology structure thanks to classification ($\mu(A) = \mu_B(A)$) and treat the messages sent by other agents ($\mu(A) = \mu_M(A)$) to help them achieve their goal. The agent therefore tries to find an equilibrum between altruism and egoism [23].

2.5 *Simplification of the hierarchy*

In the previous sections, we have seen the distributed classification algorithm implemented in Dynamo, and the complementary rules to obtain a multi-criteria hierarchy. However, as we are still close to the classification algorithm, for a taxonomy the results are not necessarily optimal from a structural point of view. The system requires further improvements which are presented below.

2.5.1 Eliminating degraded branches

After modifications carried out by the ontologist, concepts introduced initially to structure the hierarchy remain, creating degraded branches (some nodes have only one child). This phenomenon can be seen in the example given previously in Figure 7. These branches need to be eliminated of course as they provide no additional conceptual information. To do this, it is simply necessary to address the uselessness of the agents present in these branches.

In a degraded branch, the situation is similar to that in Figure 10(a). For example, the brothers of P' have been moved to another place in the hierarchy. In this case, P' does not contribute anything

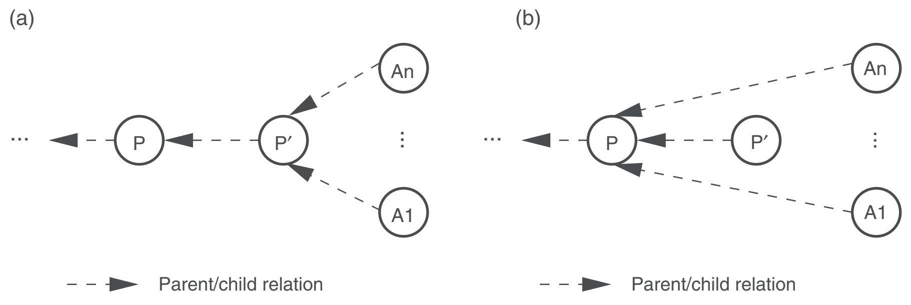

FIGURE 10. (a) Degraded branch; (b) Simplified branch

to the hierarchy. Having been introduced initially to improve the structure, because of changed conditions, it has become superfluous; its children could just as well be under P for a similar but simpler structure. This observation leads to the first rule (Rule 2) to eliminate degraded branches:

RULE 2
When an agent P' has children (noted A_k), but no brother, then P' proposes the children have parent P as new parent.

The application of this rule leads to the situation in Figure 10(b). The branch studied is indeed no longer degraded but P' is still present. As it was a structuring concept, which was not represented directly by terms and which does not have a child, it is in fact useless to the system and must disappear. This brings the second rule (Rule 3) to eliminate degraded branches, which is the rule to detect the uselessness of an agent.

RULE 3
When an agent P' does not have a child and is not represented by a term, it must withdraw from the system.

With the application of this rule, this zone of the hierarchy is simplified in order to eliminate the degraded branch. We push P' to disappear by moving its children toward P because it generates less communication inside the system. This comes from the fact that P' has no siblings, so its disappearance does not generate any knowledge updates, while P having siblings would trigger such knowledge updates.

2.5.2 Moving away from binary trees

Now that our system is able to eliminate degraded branches, one last point must be treated in order to obtain a solution which generates taxonomies. With what has been presented so far, we automatically obtain binary trees, except for the very last level in the hierarchy if the coverage rule has been applied. However, our aim is not to provide a hierarchical clustering distributed algorithm, in which case the one that has been presented would suffice, but to obtain a dynamic taxonomy. As this would rarely be a binary tree, an additional criterion must be added to form groups and thus obtain n-ary nodes.

As seen in Section 2.3, the groups which are formed impact the form of the hierarchy. It would seem possible therefore to move away from binary trees by modifying the agents' voting strategy, since it is the way one agent has voted which determines in which partition it is situated and whether the parent is able to make partitions among its children. For memory, if we take the last step of the algorithm, once the groups have been formed, we obtain Figure 11.

By construction, the group under agent P' is composed of the most dissimilar agents situated under P and which are most dissimilar to each other. So, if $sim(A,B)$ is the measure of similitude between two agents A and B, the agents must verify the following property throughout the hierarchy

$$\forall i \in [k;n], \forall j \in [k;n] \setminus \{i\}, \forall l \in [1;k-1] : sim(A_i, A_j) > sim(A_i, A_l) \tag{3}$$

The binary profile of the hierarchy comes from the fact that agents try to move away from the groups when there is a dissimilarity. A simple way to avoid this binary profile is to introduce a tolerance: ε. The verified property becomes:

$$\forall i \in [k;n], \forall j \in [k;n] \setminus \{i\}, \forall l \in [1;k-1] : sim(A_i, A_j) > sim(A_i, A_l) + \varepsilon \tag{4}$$

For this, we have readjusted the agents' voting strategy. They still take part in a Condorcet vote. But, instead of expressing a total classification of their preferences, they express a partial classification,

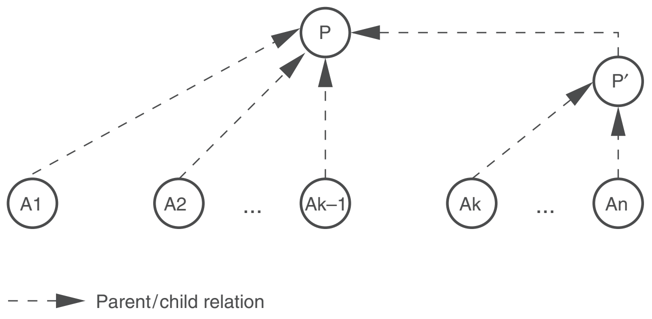

FIGURE 11. Clustering, partitions done

allowing for tolerance. In this way, an agent A, having classified its brothers B_k in decreasing order of dissimilarity, will only keep in its vote the B_k for which $1 - sim(A, B_k) > \varepsilon$.

We now have a tool which impacts the branch factor of the taxonomy. It can be used in two different ways, either by setting a global ε to the system or by setting a ε_A for each agent A in the system. This choice affects the type of result obtained:

- tolerance is set globally; in this case all the agents use it, although the tree is n-ary, the branching factor may vary significantly according to the zone in the hierarchy (some groups may be less dissimilar than others);
- tolerance is adjusted locally; in this case, each agent manipulates two different tolerance values: the first come from its parent, used for voting (e.g. to vote P' uses ε_P), and the second is its own tolerance which enables it to impact the branch factor.

We retained this second option. It seems clearer from the point of view of the ontologist to give the system an interval for the branching factor and to see the agents readjust their local tolerances to try to conform to it depending on the dissimilarity of their neighbours. This implies that tolerance is no longer a constant for the agents. We have thus been able to formulate a new rule to take into account tolerance variations:

RULE 4
When an agent P' has one of its children (noted A_k) which no longer respects the tolerance property $\varepsilon_{P'}$, then P' proposes its children have parent P as new parent.

When applied, this rule allows P to carry out a new distribution. There is no risk of oscillation[4] since at the creation of the new intermediary level, the replacement of P' will initially have $\varepsilon_{P'} = \varepsilon_P$.

We are now able to take tolerance into account during the classification and to have it vary locally. It must now be linked to the branch factor. We have not managed to establish a function which correlates local tolerance and the local branch factor in our system. This problem has however been solved by introducing a final rule which pushes each agent to look for the optimal tolerance to obtain a branch factor in the interval specified by the user:

RULE 5
When an agent P' has a number of children higher than the maximum of the interval (respectively, lower than the minimun), it lowers (respectively, increases) its tolerance $\varepsilon_{P'}$.

[4] P creates P' which disappears because of a tolerance threshold which is weaker than the one applied by P, then the process loops.

We note that this new value of $\varepsilon_{P'}$ will be used by P' to verify whether its children conform to the tolerance property and by the children of P' during the next votes.

3 Evaluation of the System

3.1 Performances of the system

In each subsection, we will assess the performances of a subset of the rules of the system. As seen in the previous section, we have several subsets of rules which are complementary. It is for this reason that we begin our discussion with the core of the rules composing our classification distributed algorithm and then complete it with each of the additional measures assessing their impact on performances [20].

We assess the performances of each set of rules from several angles when this is possible. For the rules which allow such calculations, we determined the theoretical complexity of the resulting algorithm. Moreover, we have conducted a number of experimental measures whenever this was possible. In this experimentation, we used different sized sets of data from a corpus of surgical resuscitation reports.

3.1.1 Complexity of the clustering distributed algorithm

To evaluate the properties of our distributed algorithm, it is necessary to begin with a quantitative evaluation based on its complexity, at the same time comparing it to Algorithm 1 of the previous section.

The theoretical complexity will be calculated for the worst case by considering the operation calculating the similarity between two elements as elementary.

Proposition for evaluating the complexity

$$t_1(n) = \sum_{l=1}^{n} \frac{l \times (l-1)}{2} \tag{5}$$

PROOF. For the distributed algorithm, the worst case means that, at each step, only one group of two individuals can be formed. In these conditions, for an initial set of data of n elements, we can determine the number of similarity computations done by each algorithm.

For Algorithm 1, we note $l = length(L)$; the most interwoven loop 'for' is executed $l-i$ times. Its body contains the only similarity computation, its cost is therefore $l-i$. The second loop 'for' is executed $l+1$ times for i going from 0 to l. Its cost is then $\sum_{i=0}^{l}(l-i)$ which can be simplified as $(l \times (l-1))/2$. Finally, for each execution of the loop 'while', l is decremented from n to 1, i.e. $t_1(n)$ the number of similarity computations for Algorithm **??** :

Proposition for evaluating the runtime

$$t_{dist}(n) = \sum_{l=1}^{n} l \times (l-1) \tag{6}$$

■

PROOF. During an execution of the distributed algorithm, each of the l agents evaluates its similarity with its $l-1$ brothers. Therefore, each execution has a cost of $l \times (l-1)$. Groups are then created and

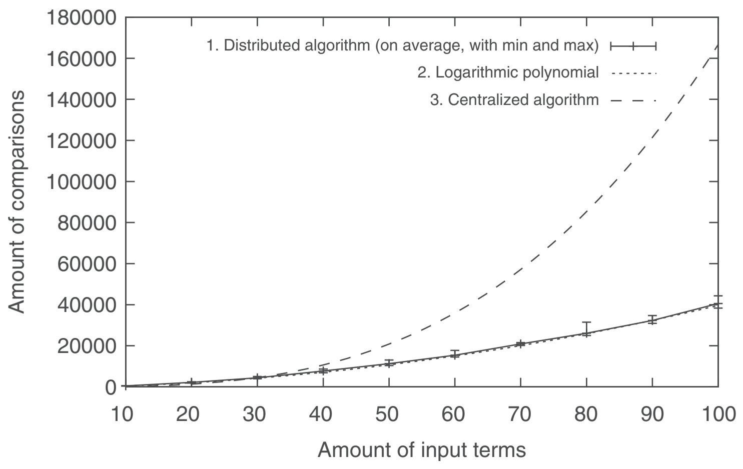

FIGURE 12. Experimental results of the clustering distributed algorithm

another vote takes place with l decremented by 1 (since we suppose in the worst case only groups of 2 or $l-1$ are constituted). Since l is equal to n at the first execution, we obtain $t_{dist}(n)$ for the number of similarity computations for the algorithm.

The two algorithms thus have a complexity $O(n^3)$. In the worst case, the distributed algorithm carries out twice as many elementary operations as the centralized algorithm. This Factor 2 difference come from the locality of the decision making among the agents. The similarity computations are thus made twice for each pair of agents. It could be envisaged that one agent, having done such a calculation, could send its result to the other one. However, this would simply mean displacing the problem by generating more communications within the system.

The observed complexity of the algorithm was determined experimentally. For this, the multi-agent system was executed with sets of input data varying from 10 to 100 terms. The value retained is the average of the number of comparisons carried out for 100 executions without the intervention of the user. This gives the curves in Figure 12.

The distributed algorithm is thus on average more efficient than the centralized algorithm and its complexity on average more reduced than in the worst case. This can be explained simply by the weak probability that a set of data would push the system to constitute only minimal (two elements) or maximum ($n-1$ elements) groups at each stage of reasoning. Curve 2 represents the logarithmic polynomial minimizing the error with Curve 1. The term with the highest degree of this polynomial is in $n^2 log(n)$, so our distributed algorithm has on average a complexity in $O(n^2 log(n))$. Finally, we can note the reduced deviation between the performances on average at the minimum and maximum. In the worst case, for 100 terms, the standard deviation is 1960.75 for an average of 100 executions of 40550.70 (i.e. around 5%) which underlines the stability of the system. ■

3.1.2 Impact of the ‘head coverage’ rules

We evaluated the experimental complexity for the multi-agent system when the ‘head coverage’ rules are activated in addition to the classification distributed algorithm. In this case, the metric used is the number of messages exchanged in the system. Once again, the system was launched on sets of input varying from 10 to 100. The value given is the average number of messages sent in the whole

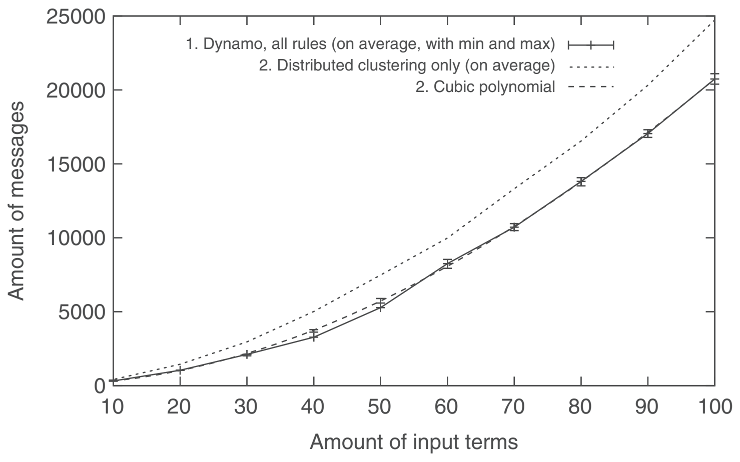

FIGURE 13. Experimental results with the ‘head-coverage’ rules

of the system for around a 100 executions, without the intervention of the user. This gives the curves in Figure 13.

Curve 1 represents the average of the values obtained. Curve 2 represents the average of the values obtained when only the clustering distributed algorithm is activated, but not the ‘head-coverage’ set of rules. Curve 3 represents the polynomial minimizing the error with Curve 1. The highest degree of this polynomial is in n^2, so our multi-agent system has an average complexity of $O(n^2)$. Note also the weak variance of performances on average at the maximum and minimum. In the worst case for 100 terms, the variance is 126.73 for an average of 20737.03 (around 0.6%) which proves the excellent stability of the system.

Finally, the additional rules for head coverage are a real improvement over the classification distributed algorithm alone. They add new constraints and the stability is obtained with less interaction and decision making by the agents. This means that fewer messages are exchanged in the system but we obtain a tree which takes into consideration additional criteria in its construction.

3.1.3 Impact of the elimination of degraded branches

To evaluate the complexity of the operation to eliminate degraded branches, an experimental approach is not really appropriate, given the simplicity of the operation. Moreover, as this treatment is only carried out following manual intervention of the user, it is difficult to set up an experimental protocol generating enough data to be exploitable. For this reason, we evaluate the impact of this treatment from a theoretical viewpoint only.

The theoretical complexity is calculated considering the operation of sending a message between two agents as elementary. We can determine the number of messages sent by the agents concerned when a degraded branch is eliminated. To visualize this operation, we just need to take Figures 14(a) and 14(b) of the previous section.

Agent P' has m sons called A_k. To carry out the simplification, P' sends a message to all of its children to propose P as new parent, i.e. m messages. Then, each of the A_k sends, a message to P to indicate the change of parent, i.e. again m messages. Finally, P sends a confirmation message to each A_k, yet again, m messages.

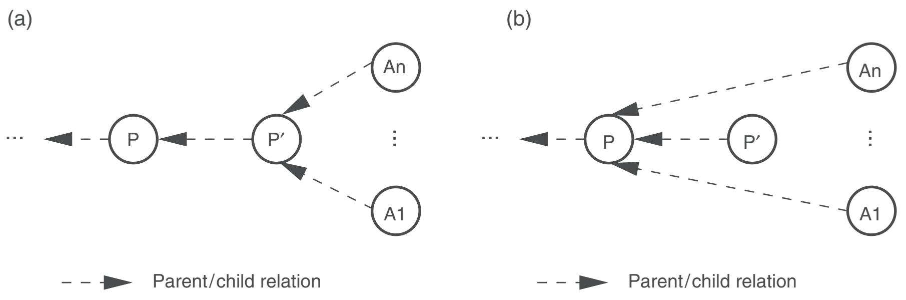

FIGURE 14. (a) Degraded branch(bis); (b) Simplified branch (bis)

In the end, the cost of the operation is $3 \times m$ with m the number of sons of agent P'. Then when a binary tree is produced from our classification decentralized algorithm, m is still equal to 2, which give this operation a complexity $O(1)$ on the scale of the system. Finally, for the production of n-ary trees as allowed by our rules evaluated in the next subsection, m may vary but remains bound, not able to rise above the value m_{max}. Here, the cost of the operation is in the worst case $3 \times m_{max}$, and we find the operation of a complexity of $O(1)$ on the scale of the system.

3.1.4 Impact of passing to n-ary trees

We finally evaluate the experimental complexity for the complete multi-agent system when all the rules are activated. As in the two previous sections, the metric used is the number of messages sent in the system. Again the system was launched with a set of input data varying from 10 to 100 terms. However, the experimental protocol is somewhat different.

In the previous experimentation, measures were taken between the launching of the system and its stabilization. Here, the cost of passing from a binary tree to an n-ary tree is measured. The best approach was to let the system stabilize first then start the rules enabling the formation of n-ary trees. To do this, after the first stabilization, the parameters were changed so that the system would look for a structure for which the branch factor at each node was 3 or 4. The given value is the average number of messages sent in the whole of the system between the first and second stabilization for around 100 executions without any manual intervention. Resulting curves are shown in Figure 15.

Curve 1 represents the average of the values obtained. Curve 2 represents the polynomial minimizing the error with Curve 1. The highest degree in this polynomial is n^3, so our multi-agent system has a complexity of $O(n^3)$ on average. We must note however the strong variance in performances on average at the maximum and minimum. To locate/identify maximum complexity, we added Curve 3 which represents the polynomial minimizing the error with the maximum. The highest degree of this polynomial is in n^4, so our multi-agent system has a complexity of $O(n^4)$ for the worst case.

In Section 3.1, concerning the head coverage rules, we observed that bringing additional constraints made the system converge more quickly. It is interesting to note that here we observe the inverse phenomenon. Passing from binary to n-ary trees presents a higher complexity, despite the constraints introduced. In our opinion, this behaviour is justified by the fact that the rules introduced do not constitute additional constraints to the solution space. On the contrary, passing to n-ary trees implies increasing the possible solutions space. However, this enlarged space contains a large number of structurally correct solutions. We can only distinguish between them via an interpretative approach, which requires reintroducing the user into the loop.

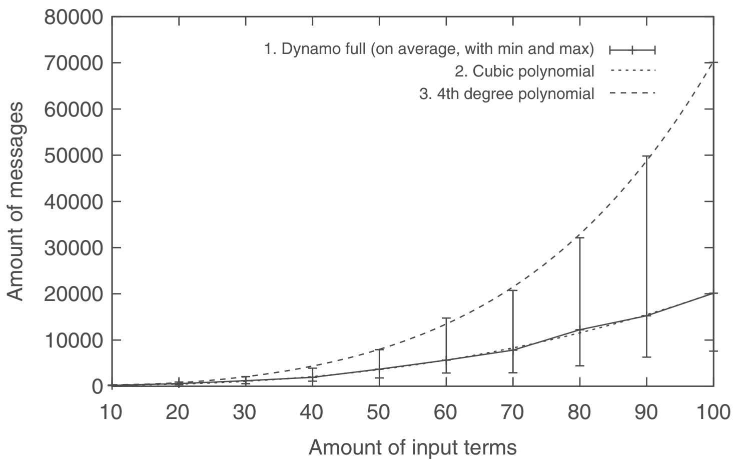

FIGURE 15. Experimental results of building a taxonomy from a binary tree

This last set of rules allows us to reach the limits of possible evaluations for the automatic running of the system without any outside intervention. It is important therefore to focus on the interactions with the user and to evaluate the behaviour of our system in this context. This is the main focus of the rest of this section.

3.2 Application

As seen in the first part of this article, the Dynamo system cannot be evaluated solely on the complexity criterion. The limits of the experimental field cannot take into account the contribution of the user in the Dynamo approach. This is why we will now concentrate on the use of our system in conditions similar to a real use. Our aim is to evaluate the quality of the taxonomy proposed by Dynamo and the ontologist cognitive workload.

We propose to evaluate the ontology building process of Dynamo in two ways. First, we compare the quality of a taxonomy automatically built by the system with a target taxonomy. To do so, we propose to measure the distance between both taxonomies. Second, we measure the effort made by the ontologist to transform the taxonomy proposed by Dynamo into a suitable taxonomy.

The first subsection presents the experimental protocol and the results obtained for evaluating the quality of the taxonomy. The second subsection presents the different steps and modifications the ontologist has to do. Finally we discuss the limitations of our system.

3.3 Evaluating the quality of the taxonomy

3.3.1 The corpus and target taxonomy

To evaluate the quality of a taxonomy built with Dynamo, we considered a reference taxonomy constructed from a specific corpus according to the approach presented in [18]. This target ontology has been validated by domain experts. We propose to build a new ontology on this corpus with Dynamo. This corpus is named Astro. It is a collection of more than 3000 summaries of scientific articles published in 1995 and 2002 in the journal ‘Astronomy & Astrophysics’.

3.3.2 Experimental protocol

The approach for our evaluation is to isolate subparts of the target ontology then to compare the ontology built by our system. This type of evaluation is difficult. In particular, our reference ontology does not have a universal scope, it is therefore difficult and even dangerous to characterize the system solely by trying to obtain exactly the same reference ontology. In fine, the quality of the result is necessarily subjective and we must keep this in mind and consider the reference ontology as a point of comparison rather than an aim to reach. To this must be added the fact that just comparing two ontologies requires using an appropriate metric. For this, we examined different metrics to determine which was the most suitable for our task. In our case, the structure is more important than the lexicon. It is the experimenter who chooses a number of terms in the reference ontology which are also taken from the texts. We found an interesting measure to compare the structure of two ontologies in [16]. We use this measure of *taxonomy overlapping* to compare the current state of the system and the reference ontology. This measure goes from 0 to 1, 1 being equality between the two taxonomies.

3.3.3 Measuring distance after the automatic execution

For the evaluation, the system begins by using as input the terms associated with the concepts present in the ontology. Our aim being to examine the structuration mechanisms and the system's behaviour, we make sure that the vocabulary is the same in the two ontologies. The system is then launched in totally automatic mode without the intervention of the user. With this stage, we know the behaviour of the corpus from a purely statistical point of view, and can determine the initial parameters. In function of these parameters, the draft obtained is different and it is up to the user to determine the results which suit him most and thus fix the initial parameters. This is evidently the most subjective stage.

For the first stage, we tested several settings for the similarity measure used by our system. First, we tested the Jaccard measure which did not give a satisfactory result on the corpus as the notions were difficult to locate in the hierarchy. We then used again the measure proposed by [1] (Section 2.3) with different settings for the parameter α. The best result, presenting in our opinion the least dispersion, was obtained with α at 0,75. We retained this measure to parameterize our system.

We then had to choose an indicative range for the branch factor and for similarity tolerance, used by the agent rules presented in Section 2.5. As we wanted to obtain a product comparable to our reference ontology, we were guided in our choice by the latter. Each level of the reference ontology is quite large—a large number of brother concepts per level. We therefore chose the branch range factor varying from 2 to 7. With an input of 90 terms, this is above the average branch factor of the reference ontology. We wanted to push the system to propose a slightly deeper structure.

During these adjustments, we were able to obtain complementary information on the corpus. This first stage is important for the next part. In the case of the Astro corpus, its synthetic but well-written style gave quite good results with the clustering but was even more receptive with the head coverage rule, which denotes a frequent use of the ellipsis. This phenomenon is probably due to the constraints of space inherent in summaries of scientific articles.

Figure 16 shows the result obtained after automatic execution with the above mentioned parameters. Some of the leaves of the taxonomy have been withdrawn from the screenshot to improve readability. It is striking that with an input of only 90 terms, the readability of the structure obtained is so limited. Evidently, this is worsened by the fact that the system develops in time before stabilizing and it is difficult to follow the process. In this way, we can consider that it is too difficult to interpret the results of our system. The current interface seems inadequate and the representation of a graph

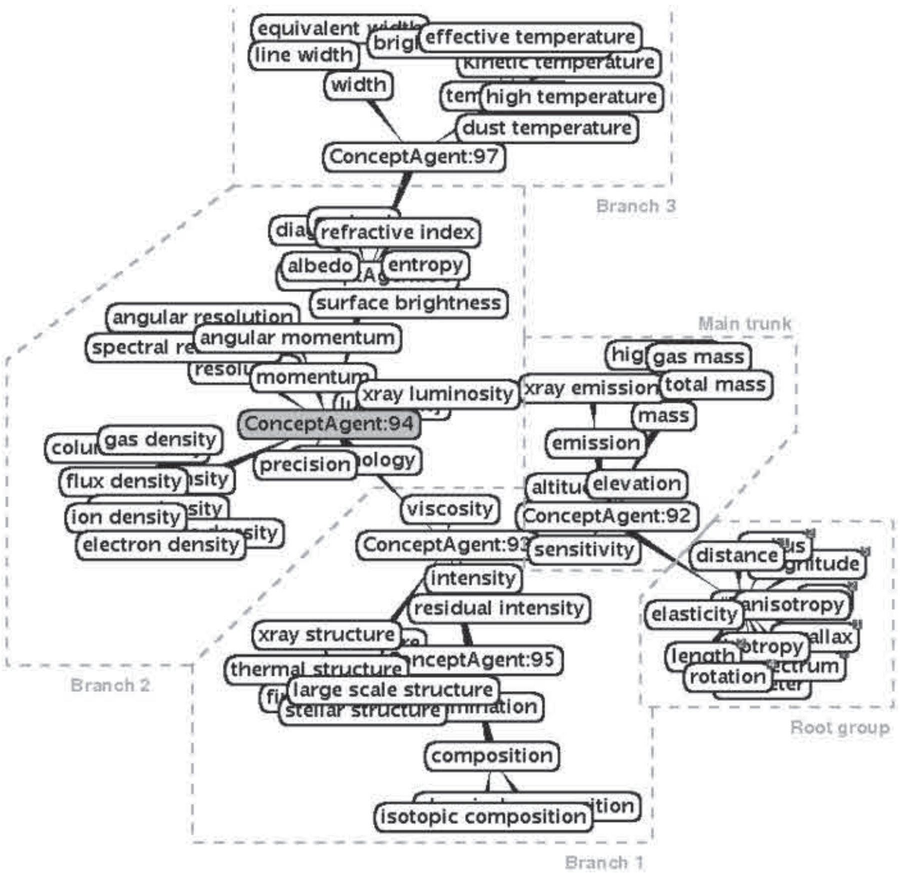

FIGURE 16. First taxonomy after a fully automated execution on the Astro corpus

in quasi-permanent evolution seems a difficult problem. Our system then, in its present state, needs an improved interface before it can be considered as a complete solution to ontology engineering.

The metric retained to compare the ontologies gives around 0.78. This value is quite high despite our structure which is less 'flat' than the reference ontology. This means that, despite the existence of additional branches at a high level, the relative placement of concepts in the lower layers is comparable. Now, looking more closely, it can be observed that the structure is rather unbalanced. Indeed, a long branch with three sub-groups with no well defined theme stems from the root of the tree structure (at the centre of the root group, situated in the bottom right-hand corner of the figure). This constitutes the first angle of attack to interact with the system. We are now working on the system to obtain a more satisfactory structure. This section presents the four modifications made to obtain the final result.

3.4 Considerations on the effort to produce the taxonomy

Once the system has proposed the ontology, the user intervenes on the structure to make it converge towards the result he wants to obtain, taking into account the behaviour of the system coming from the statistics calculated on the text.

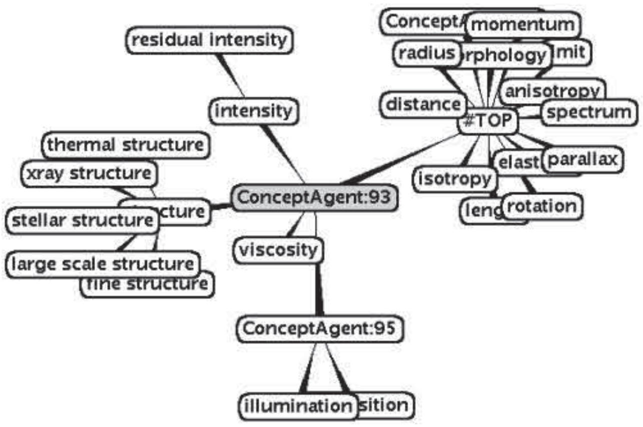

FIGURE 17. System stabilization after the first modification (extract centered on ConceptAgent:93)

3.4.1 First modification

As seen in the previous section, the structure obtained automatically presents a degraded branch and must be improved. This branch presents three sub-groups and each of them is brought back to the root. The system reacts and reorganizes itself.

Being flatter, this new structure has a value of 0.80 when compared to our reference ontology. We observe that 'ConceptAgent:93' groups essentially mechanical and thermodynamic properties, whereas '#TOP' has more geometrical properties. Figure 17 presents an extract of the hierarchy obtained centered on 'ConceptAgent:93'. We are now going to try to improve the separation of the different notions represented.

3.4.2 Second modification

As we have just seen, 'ConceptAgent:93' constitutes an attractive pole for mechanical and thermodynamic properties. Our second modification is to clean the other branches and move the concepts concerning mechanical and thermodynamic properties to 'ConceptAgent:93'. The concepts concerned are: *isotropy, morphology, momentum, anisotropy, spectrum, elasticity, mass, sensitivity, emission, density, entropy, diagram* and *temperature*, i.e. we will make 13 moves in all.

The new structure obtained (Figure 18) presents a more complex branch concerning only mechanical and thermodynamic properties. Because of the structure of this branch, we have moved away from the reference ontology and the comparison metric gives around 0.76. This value remains stable until the end of the experimentation.

3.4.3 Third modification

We now clean the branch beginning with 'ConceptAgent:93' which until now has been ignored. It contains geometrical properties and a subbranch containing optical properties. We therefore move this branch under '#TOP', and all the geometrical properties of '#TOP' under 'ConceptAgent:99'.

The system then reacts and moves all the geometrical and optical properties under '#TOP', then creates a new branch containing only optical properties and *width* (Figure 19). The destruction of our branch for geometrical properties indicates too great a disparity with its contents on the statistical

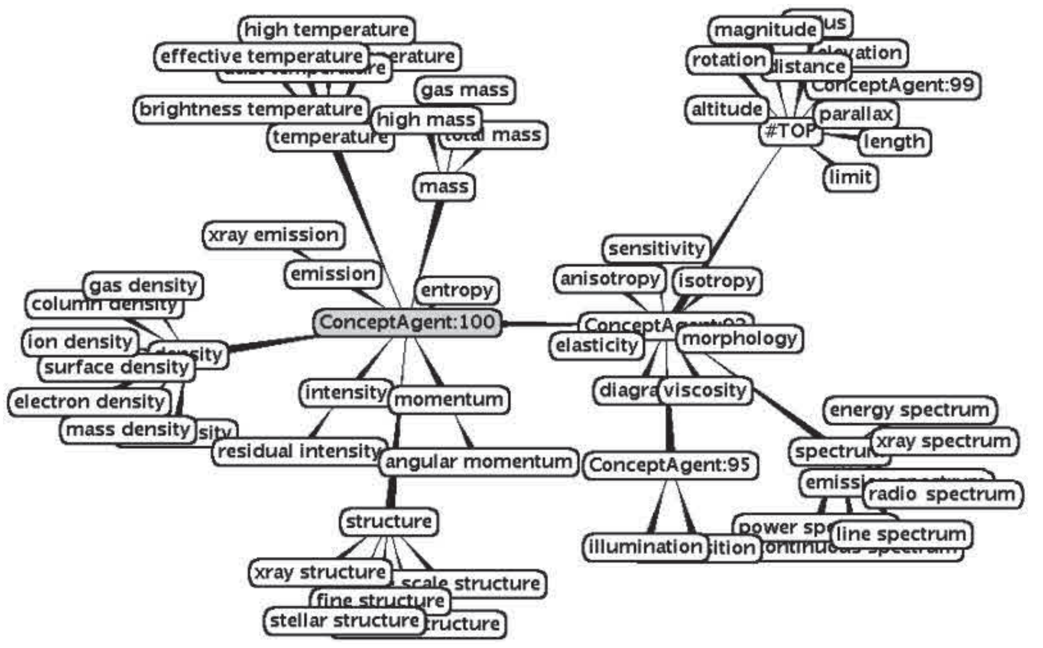

FIGURE 18. System stabilization after the second modification (extract displaying the "mechanical and thermodynamic" branch)

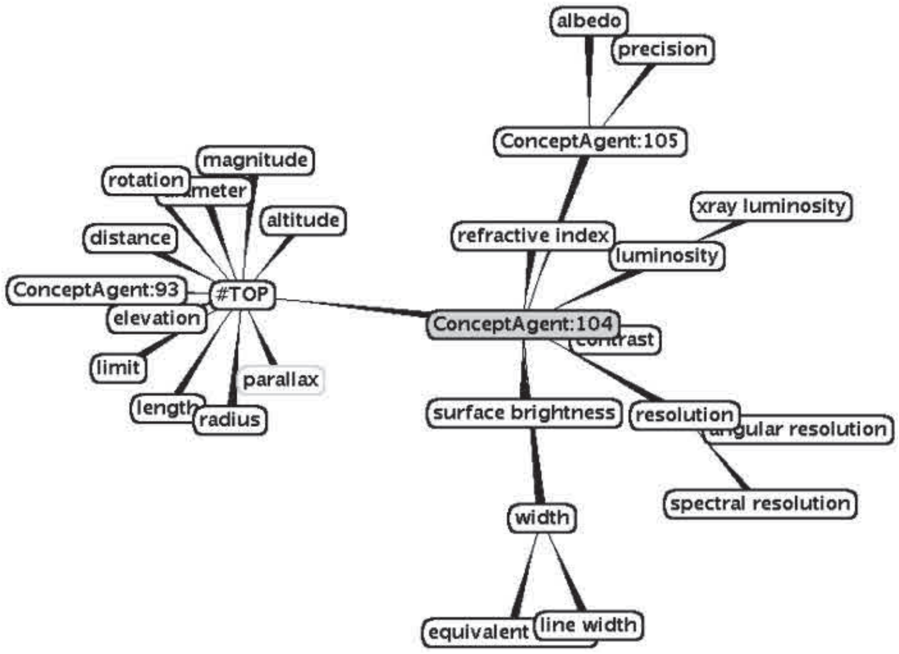

FIGURE 19. System stabilization after the third modification (extract displaying the 'optical' branch)

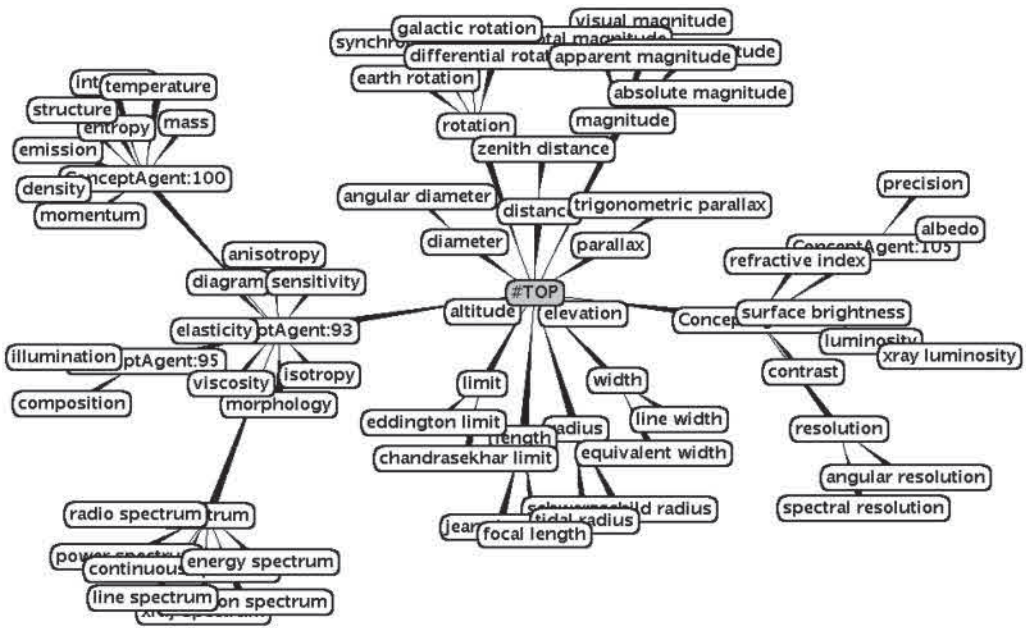

FIGURE 20. System stabilization after the fourth modification

level. Moreover, the concepts in question are sufficiently generic to be kept close to more specific terms as those of thermodynamics or optics. We decide therefore to keep this modification.

3.4.4 Fourth modification

We are now close to the final result. We have three coherent sets of concepts under '#TOP': the concepts linked to the root, a group under 'ConceptAgent:93' and a group under 'ConceptAgent:104'. Although we are not trying to create a third subbranch under '#TOP', it is clear that 'width' is not in its place under the branch of optical properties. Our last modification will therefore be to move this concept to the root.

After stabilization, we obtain the structure in Figure 20. Three coherent subgroups can be observed: on the left mechanical and thermodynamic properties, in the centre geometrical properties and on the right optical properties. The total time spent by the user from starting the system to obtaining the result was around 3 h. Most of the time was spent thinking about the actions to take, but especially observing the structure and modifying its visualization in order to make his choices.

3.4.5 Discussion

Despite the difficulties to read the displayed ontology, a result which is close to the reference ontology was obtained in a relatively short time. The taxonomy produced is even slightly more refined which explains the low values of our metric.

As such, the process requires reduced intervention by the ontologist. The reported experiment has been carried out in only four steps. Each intervention by the user led to the movement of agents in the hierarchy. For the first two, about 30 agents changed parent at each step; the last two interventions each generated only half a dozen movements. It is interesting to note therefore that the system converges

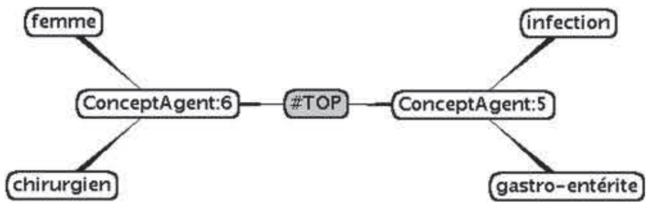

FIGURE 21. Concept agent tree after autonomous stabilization of the system

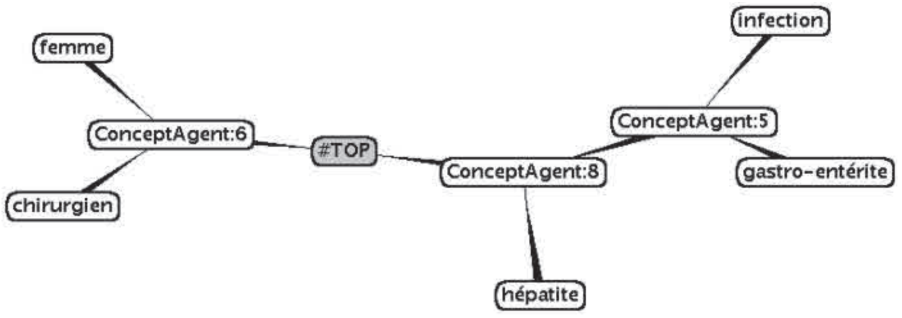

FIGURE 22. Concept agent tree after taking 'hepatitis' into account

and that the number of interventions by the user progressively decreases. The ontologist can then concentrate on the broad lines of the draft he wants to produce, as the system adapts the low-level concepts.

3.5 *Current limitation of our approach*

Currently, the main drawback of our algorithm is that the result are dependent on the order in which data are inserted in the system. When the system works by itself on a fixed set of data, provided at startup, the result is equivalent to what could be obtained with a centralized approach (except that we gained dynamicity, and the ability to take more criteria into account). Unfortunately, adding a new item after a first stabilization has an impact on the final result.

To illustrate our claim, we present another example of the behaviour of our system. Since this issue is a legacy to our classification roots, this experiment is conducted only with the core behaviour of our system (presented in Section 2.3). We use a small test data set and let the system work by itself. After stabilization, we obtain the hierarchy of Figure 21.

The user then intervenes and adds a new concept represented by the term 'hepatitis' under the root of the hierarchy. The system then reacts and stabilizes itself. We then obtain the organization shown in Figure 22. 'hepatitis' is placed in the correct branch, but we do not obtain the desired organization of Figure 7.

On-going developments in Dynamo aim at adding new rules that will allow a concept to have more freedom to travel along branches. Adding such a rule will make it more difficult to compare our system with traditional classification algorithms.

4 Related Work

Although ontologies are defined to as stable and consensual knowledge representations, their use on the Semantic Web context makes it necessary to regularly update them. Major works about ontology evolution bear on the propagation of changes and ensure the ontology consistency after an evolution. For instances, [24] propose a process that analyses the causes and the consequences of changes and ensures the consistency of the ontology after resolving these changes. This process can be customized according to evolution strategies and by suggesting the user some changes according to the analysis of the ontology and its use in applications. Other work raise the issue of collaborative and distributed evolution of ontologies [19]. Their solution stores changes together with the ontology as annotations on its components, which makes versioning and back-tracking easier. In Dynamo, changes are propagated straightforward and not tracked. But undoing a change is made easy through the interface. Flouris [9] proposes a classification of ontology changes happening during ontology evolution, versioning and merging. Changes in Dynamo refer to ontology evolution (only evolutions related to concepts and the is-a hierarchical relation) and consistency is ensured thanks to the agents' cooperative behaviour.

Thanks to the maturity reached by Natural Language Processing over the past 10 years, engineering ontologies from text is now acknowledged as a powerful means to efficiently learn or populate ontologies [4]. Dynamo belongs to this trend as its input are data extracted from text by a term extractor. Text2Onto [6], one of the best known method and framework for learning ontologies from text, considers this learning process as a continuous evolution driven by the changes detected from text analysis. Dynamo promotes the same perspective where initial design and evolution are carried out within the same framework.

In the multi-agent field, ontologies generally enable agents to understand each other [13]. Exchanges between several software agents require an agreement about a shared conceptualization in order to reach a mutual understanding. Usually agents do not have any capability to create or maintain them. From our knowledge, multi-agent systems have not been used before to build or maintain an ontology. They are sometimes used to facilitate the ontology building process, in particular for collaboration [3], but as said in [21] they rarely represent the ontology itself. The closest contribution to Dynamo can be found in [22]. They present a multi-agent system for ontology generation from hierarchical conceptual clustering. In their system, each node of the hierarchy is an agent, created when new data are encountered in the entry flow of the system. Then, each autonomous agent can decide either to join with a brother agent or to promote a son agent as a new brother. This hierachical reorganisation process is based on a homogeneity measure computed by each agent by taking into account their sons. In our Dynamo system, the multi-agent system itself is the resulting hierarchy.

5 Conclusion

The Dynamo System can make an ontology a 'living material'. It enables the construction and evolution of an ontology according to text analysis and to interactions with a user. Dynamo implements such dynamic ontologies using adaptive multi-agent systems. It makes it easier to add new constraints for the management of the ontology. Adaptive agents have a core behaviour for representing and organizing the concepts of the ontology, but their behaviour can be incrementally enriched by adding new rules.

We have carried out an evaluation experiment of Dynamo with promising results. However, the resulting ontology lacks high-level concepts as, for the moment, Dynamo only builds domain-specific taxonomies. This first result is nevertheless interesting since in some fields, like information retrieval, no formal ontologies are necessary. We plan to improve Dynamo by reusing top level ontologies such as DOLCE, and 'teaching' agents how to position themselves in such ontologies. We are also thinking of a way to create non-taxonomic relations. A pattern-based approach such as the one presented in [11] could provide agents with data from which they could learn semantic relations. We will also consider the work by Völker and colleagues [25] who suggest tracks for relation and axiom extraction from text. Another limitation of Dynamo is that the interface is not optimal. More work will be done to display to the ontologist a large ontology which dynamically changes depending on interactions.

Since the first evaluation showed that taxonomies are easily created with Dynamo, we are currently working on embedding such a system in an end-user information system. This perspective will make it possible to annotate documents with an ontology which captures knowledge linked both to the text collection and to the system users. The dynamic aspects of Dynamo will be fully exploited as new texts are constantly added to the information system and as the user's knowledge evolves with time.

References

[1] H. Assadi. Construction of a regional ontology from text and its use within a documentary system. In *Proceedings of the International Conference on Formal Ontology and Information Systems - FOIS'98*, pp. 236–249. Trento, Italy, 1998.

[2] N. Aussenac-Gilles and D. Sörgel. Text analysis for ontology and terminology engineering. *Journal of Applied Ontology*, **1**, 35–46, 2005.

[3] J. Bao and V. Honavar. Collaborative ontology building with wiki@nt. In *Proceedings of the Third International Workshop on Evaluation of Ontology-Based Tools (EON2004)*, Hiroshima, Japan, 2004. http://km.aifb.unikarlsruhe. de/ws/eon2004/EON2004 Proceedings.pdf.

[4] P. Buitelaar, P. Cimiano, and B. Magnini. *Ontology Learning From Text: Methods, Evaluation and Applications*. IOS Press, 2005.

[5] C. Brewster, F. Ciravegna, and Y. Wilks. Background and foreground knowledge in dynamic ontology construction. In *Proceedings of the Semantic Web Work- shop, Toronto (Can), August 2003. SIGIR, 2003*.

[6] P. Cimiano and J. Völker. Text2onto - a framework for ontology learning and data-driven change discovery. In *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB'2005), June 2005*, Vol. 3513, pp. 227–238. Springer Verlag, Alicante, ESPAGNE, 2005.

[7] N. de Condorcet. *Essai sur l'application de l'analyse la probabilitž des dcisions rendues à la pluralité des voix*. Paris, 1785.

[8] D. Faure and C. Nedellec. A corpus-based conceptual clustering method for verb frames and ontology acquisition. In *LREC Workshop on Adapting Lexical and Corpus Resources to Sublanguages and Applications*, pp. 5–12. Granada, Spain, 1998.

[9] G. Flouris, D. Plexousakis, and G. Antoniou. A classification of ontology change. In *SWAP 2006 – Semantic Web Applications and Perspectives, Proceedings of the 3rd Italian Semantic Web Workshop, Scuola Normale Superiore, Pisa, Italy, 18–20 December, 2006*, G. Tummarello, P. Bouquet, and O. Signore, eds, Vol. 201 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[10] J.-P. Georgé, G. Picard, M.-P. Gleizes, and P. Glize. Living Design for Open Computational Systems. In *12th IEEE International Workshops on Enabling Technologies, Infrastructure for Collaborative Enterprises, Linz (Austria)*, IEEE CS, pp. 389–394. June 2003.

[11] M. Hearst. Automatic acquisition of hyponyms froms large text corpora. *13th international Conference On Computational Linguistics (COLING)*, pp. 539–545. Nantes, France, 1992.

[12] J. Heflin and J. Hendler. Dynamic ontologies on the web. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pp. 443–449. AAAI/MIT Press, Menlo Park, CA (USA), 2000.

[13] K. Lister, L. Sterling, and K. Taveter. Reconciling Ontological Differences by Assistant Agents. In *AAMAS'06, Proceedings of the International Joint Conference on Autonomous Agents and MultiAgent Systems*, pp. 943–945. Hokkaido, Japan, May 2006.

[14] A. Maedche. *Ontology Learning for the Semantic Web*. Kluwer Academic Publisher, 2002.

[15] A. Maedche and S. Staab. Semi-automatic engineering of ontologies from text. In *Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering* (*SEKE'2000*), pp. 231–239. 2000.

[16] A. Maedche and S. Staab. Measuring Similarity between Ontologies. In *EKAW 2002* Gómez-Pérez and Benjamins, eds, Vol. 2473 of *Lecture Notes in Computer Science*. pp. 251–263. Springer, 2002.

[17] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts, 1999.

[18] J. Mothe and N. Hernandez. Data Mining with Ontologies: Implementations, Findings, and Frameworks, In *Mining a thesaurus and texts to build and update a domain ontology, Chapter VII*. H. O. Nigro, S. G. C. Csaro, and D. Xodo, eds, pp. 123–144. Information Science Reference, October 2007.

[19] N. F. Noy, A. Chugh, W. Liu, and M. A. Musen. A framework for ontology evolution in collaborative environments. In *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, Proceedings*, Vol. 4273 of *Lecture Notes in Computer Science*, pp 544—558. Springer, Berlin, 2006.

[20] K. Ottens, M.-P. Gleizes, and P. Glize. A Multi-Agent System for Building Dynamic Ontologies. In *AAMAS 2007, Proceedings of the International Joint Conference on Autonomous Agents and Multi Agent Systems*, pp. 1278—1284. ACM Press, Honolulu (USA), May 2007.

[21] H. Van Dyhe Parunak, R. Rohwer, T. C. Belding, and S. Brueckner. Dynamic decentralized any-time hierarchical clustering. *29th Annual International ACM SIGIR Conference on Research & Development on Information Retrieval*, Springer Berlin, Heidelberg, 2006.

[22] H. Van Dyhe. Parunak, R. Rohwer, T. C. Belding, and S. Brueckner. Dynamic decentralized any-time hierarchical clustering. In *Engineering Self-Organising Systems*, Vol. 4335 of *Lecture Notes in Computer Science*, pp. 66–81. Springer Berlin, Heidelberg, 2006.

[23] G. Picard and P. Glize. Model and analysis of local decision based on cooperative self-organization for problem solving. *Multiagent and Grid Systems*, **2**, 253–265, 2006.

[24] L. Stojanovic, A. Maedche, B. Motik, and N. Stojanovic. User-driven ontology evolution management. In *EKAW2002*, A. Gómez-Pérez and V. R. Benjamins, eds, Vol. 2473 of *Lecture Notes in Computer Science*, pp. 285–300. Springer, 2002.

[25] J. Völker, P. Hitzler, and P. Cimiano. Acquisition of owl dl axioms from lexical resources. In *ESWC2007*, E. Franconi, M. Kifer, and W. May, eds, Vol. 4519 of *Lecture Notes in Computer Science*, pp. 670–685. Springer, 2007.