A Multi-Agent System for Building Dynamic Ontologies

Kévin Ottens, Marie-Pierre Gleizes, Pierre Glize

A Multi-Agent System for Building Dynamic Ontologies

Kévin Ottens*
IRIT, Université Paul Sabatier
118 Route de Narbonne
F-31062 TOULOUSE
ottens@irit.fr

Marie-Pierre Gleizes
IRIT, Université Paul Sabatier
118 Route de Narbonne
F-31062 TOULOUSE
gleizes@irit.fr

Pierre Glize
IRIT, Université Paul Sabatier
118 Route de Narbonne
F-31062 TOULOUSE
glize@irit.fr

ABSTRACT

Ontologies building from text is still a time-consuming task which justifies the growth of Ontology Learning. Our system named *Dynamo* is designed along this domain but following an original approach based on an adaptive multi-agent architecture. In this paper we present a distributed hierarchical clustering algorithm, core of our approach. It is evaluated and compared to a more conventional centralized algorithm. We also present how it has been improved using a multi-criteria approach. With those results in mind, we discuss the limits of our system and add as perspectives the modifications required to reach a complete ontology building solution.

General Terms

Algorithms, Experimentation

Keywords

Ontologies, Cooperation, Emergent behavior

1. INTRODUCTION

Nowadays, it is well established that ontologies are needed for semantic web, knowledge management, B2B... For knowledge management, ontologies are used to annotate documents and to enhance the information retrieval. But building an ontology manually is a slow, tedious, costly, complex and time consuming process. Currently, a real challenge lies in building them automatically or semi-automatically and keeping them up to date. It would mean creating dynamic ontologies [10] and it justifies the emergence of ontology learning techniques [14] [13].

Our research focuses on Dynamo (an acronym of DYNAMic On-tologies), a tool based on an adaptive multi-agent system to con-struct and maintain an ontology from a domain specific set of texts.

*PhD student

Our aim is not to build an exhaustive, general hierarchical 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 is using information from texts and expert interviews to design ontologies.

In the multi-agent field, ontologies generally enable agents to understand each other [12]. They're sometimes used to ease the ontology building process, in particular for collaborative contexts [3], but they rarely represent the ontology itself [16]. Most works interested in the construction of ontologies [7] propose the refinement of ontologies. This process consists in using an existing ontology and building a new one from it. This approach is different from our approach because Dynamo starts from scratch. Researchers, working on the construction of ontologies from texts, claim that the work to be automated requires external resources such as a dictionary [14], or web access [5]. In our work, we propose an interaction between the ontologist and the system, our external resource lies both in the texts and the ontologist.

This paper first presents, in section 2, the big picture of the Dynamo system. In particular the motives that led to its creation and its general architecture. Then, in section 3 we discuss the distributed clustering algorithm used in Dynamo and compare it to a more classic centralized approach. Section 4 is dedicated to some enhancement of the agents behavior that got designed by taking into account criteria ignored by clustering. And finally, in section 5, we discuss the limitations of our approach and explain how it will be addressed in further work.

2. DYNAMO OVERVIEW

2.1 Ontology as a Multi-Agent System

Dynamo aims at reducing the need for manual actions in processing the text analysis results and at suggesting a concept network kick-off in order to build ontologies more efficiently. The chosen approach is completely original to our knowledge and uses an adaptive multi-agent system. This choice comes from the qualities offered by multi-agent system: they can ease the interactive design of a system [8] (in our case, a conceptual network), they allow its incremental building by progressively taking into account new data (coming from text analysis and user interaction), and last but not least they can be easily distributed across a computer network.

Dynamo takes a syntactical and terminological analysis of texts as input. It uses several criteria based on statistics computed from the linguistic contexts of terms to create and position the concepts. As output, Dynamo provides to the analyst a hierarchical organization of concepts (the multi-agent system itself) that can be validated, refined of modified, until he/she obtains a satisfying state of

the semantic network.

An ontology can be seen as a stable map constituted of conceptual entities, represented here by agents, linked by labelled relations. Thus, our approach considers an ontology as a type of equilibrium between its concept-agents where their forces are defined by their potential relationships. The ontology modification is a perturbation of the previous equilibrium by the appearance or disappearance of agents or relationships. In this way, a dynamic ontology is a self-organizing process occurring when new texts are included into the corpus, or when the ontologist interacts with it.

To support the needed flexibility of such a system we use a self-organizing multi-agent system based on a cooperative approach [9]. We followed the ADELFE method [4] proposed to drive the design of this kind of multi-agent system. It justifies how we designed some of the rules used by our agents in order to maximize the cooperation degree within Dynamo's multi-agent system.

2.2 Proposed Architecture

In this section, we present our system architecture. It addresses the needs of Knowledge Engineering in the context of dynamic ontology management and maintenance when the ontology is linked to a document collection.

The Dynamo system consists of three parts (cf. figure 1):

- a term network, obtained thanks to a term extraction tool used to preprocess the textual corpus,
- a multi-agent system which uses the term network to make a hierarchical clustering in order to obtain a taxonomy of concepts,
- an interface allowing the ontologist to visualize and control the clustering process.

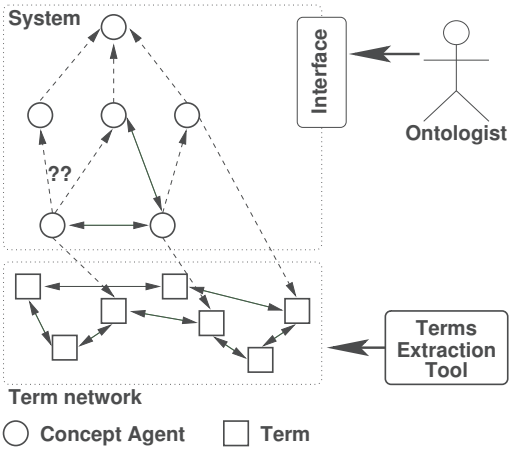

Figure 1: System architecture

The term extractor we use is *Syntex*, a software that has efficiently been used for ontology building tasks [11]. We mainly selected it because of its robustness and the great amount of information extracted. In particular, it creates a "Head-Expansion" network which has already proven to be interesting for a clustering system [1]. In such a network, each term is linked to its head term[1] and its expansion term[2], and also to all the terms for which it is a head or an expansion term. For example, "knowledge engineering from text" has "knowledge engineering" as head term and "text" as expansion term. Moreover, "knowledge engineering" is composed of "knowledge" as head term and "engineering" as expansion term.

With Dynamo, the term network obtained as the output of the extractor is stored in a database. For each term pair, we assume that it is possible to compute a similarity value in order to make a clustering [6] [1]. Because of the nature of the data, we are only focusing on similarity computation between objects described thanks to binary variables, that means that each item is described by the presence or absence of a characteristic set [15]. In the case of terms we are generally dealing with their usage contexts. With Syntex, those contexts are identified by terms and characterized by some syntactic relations.

The Dynamo multi-agent system implements the distributed clustering algorithm described in detail in section 3 and the rules described in section 4. It is designed to be both the system producing the resulting structure and the structure itself. It means that each agent represent a class in the taxonomy. Then, the system output is the organization obtained from the interaction between agents, while taking into account feedback coming from the ontologist when he/she modifies the taxonomy given his needs or expertise.

3. DISTRIBUTED CLUSTERING

This section presents the distributed clustering algorithm used in Dynamo. For the sake of understanding, and because of its evaluation in section 3.1, we recall the basic centralized algorithm used for a hierarchical ascending clustering in a non metric space, when a symmetrical similarity measure is available [15] (which is the case of the measures used in our 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 ← 0;
    A ← nil;
    B ← nil;
    for i ← 1 to length(L) do
        I ← L[i];
        for j ← i + 1 to length(L) do
            J ← L[j];
            sim ← similarity(I, J);
            if sim > max then
                max ← sim;
                A ← I;
                B ← J;
            end
        end
    end
    remove(A, L);
    remove(B, L);
    append((A, B), L);
end
R ← L[1];
```

In algorithm 1, for each clustering step, the pair of the most similar elements is determined. Those two elements are grouped in a cluster, and the resulting class is appended to the list of remaining elements. This algorithm stops when the list has only one element left.

[1] *i.e.* the maximum sub-phrase located as head of the term

[2] *i.e.* the maximum sub-phrase located as tail of the term

The hierarchy resulting from algorithm 1 is always a binary tree because of the way grouping is done. Moreover grouping the most similar elements is equivalent to moving them away from the least similar ones. Our distributed algorithm is designed relying on those two facts. It is executed concurrently in each of the agents of the system.

Note that, in the following of this paper, we used for both algorithms an Anderberg similarity (with $\alpha = 0.75$) and an average link clustering strategy [15]. Those choices have an impact on the resulting tree, but they impact neither the global execution of the algorithm nor its complexity.

We now present the distributed algorithm used in our system. It is bootstrapped in the following way:

- a *TOP* agent having no parent is created, it will be the root of the resulting taxonomy,
- an agent is created for each term to be positioned in the taxonomy, they all have *TOP* as parent.

Once this basic structure is set, the algorithm runs until it reaches equilibrium and then provides the resulting taxonomy.

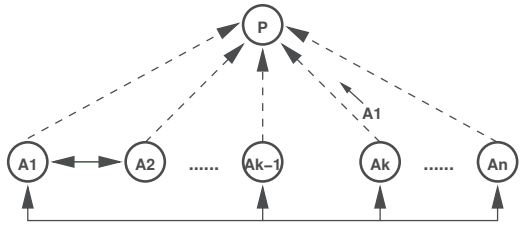

Figure 2: Distributed classification: Step 1

The process first step (figure 2) is triggered when an agent (here A_k) has more than one brother (since we want to obtain a binary tree). Then it sends a message to its parent P indicating its most dissimilar brother (here A_1). Then P receives the same kind of message from each of its children. In the following, this kind of message will be called a "vote".

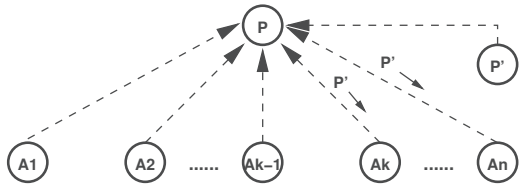

Figure 3: Distributed clustering: Step 2

Next, when P has got messages from all its children, it starts the second step (figure 3). Thanks to the received messages indicating the preferences of its children, P can determine three sub-groups among its children:

- the child which got the most "votes" by its brothers, that is the child being the most dissimilar from the greatest number of its brothers. In case of a draw, one of the winners is chosen randomly (here A_1),
- the children that allowed the "election" of the first group, that is the agents which chose their brother of the first group as being the most dissimilar one (here A_k to A_n),
- the remaining children (here A_2 to A_{k-1}).

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

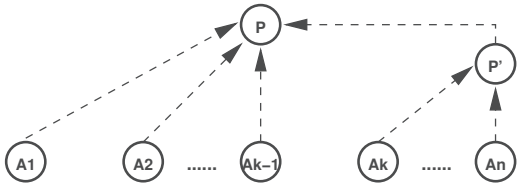

Figure 4: Distributed clustering: Step 3

Finally, step 3 (figure 4) is trivial. The children rejected by P (here agent A_2 to A_n) take its message into account and choose P' as their new parent. The hierarchy just created a new intermediate level.

Note that this algorithm generally converges, since the number of brothers of an agent drops. When an agent has only one remaining brother, its activity stops (although it keeps processing messages coming from its children). However in a few cases we can reach a "circular conflict" in the voting procedure when for example A votes against B, B against C and C against A. With the current system no decision can be taken. The current procedure should be improved to address this, probably using a ranked voting method.

3.1 Quantitative Evaluation

Now, we evaluate the properties of our distributed algorithm. It requires to begin with a quantitative evaluation, based on its complexity, while comparing it with the algorithm 1 from the previous section.

Its theoretical complexity is calculated for the worst case, by considering the similarity computation operation as elementary. For the distributed algorithm, the worst case means that for each run, only a two-item group can be created. Under those conditions, for a given dataset of n items, we can determine the amount of similarity computations.

For algorithm 1, we note $l = length(L)$, then the most enclosed "for" loop is run $l - i$ times. And its body has the only similarity computation, so its cost is $l-i$. The second "for" loop is ran l times for i ranging from 1 to l. Then its cost is $\sum_{i=1}^{l}(l-i)$ which can be simplified in $\frac{l\times(l-1)}{2}$. Finally for each run of the "while" loop, l is decreased from n to 1 which gives us $t_1(n)$ as the amount of similarity computations for algorithm 1:

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

For the distributed algorithm, at a given step, each one of the l agents evaluates the similarity with its $l-1$ brothers. So each steps has a $l \times (l-1)$ cost. Then, groups are created and another vote occurs with l decreased by one (since we assume worst case, only groups of size 2 or $l-1$ are built). Since l is equal to n on first run, we obtain $t_{dist}(n)$ as the amount of similarity computations for the distributed algorithm:

$$t_{dist}(n) = \sum_{l=1}^{n} l \times (l-1) \tag{2}$$

Both algorithms then have an $O(n^3)$ complexity. But in the worst case, the distributed algorithm does twice the number of el-

ementary operations done by the centralized algorithm. This gap comes from the local decision making in each agent. Because of this, the similarity computations are done twice for each agent pair. We could conceive that an agent sends its computation result to its peer. But, it would simply move the problem by generating more communication in the system.

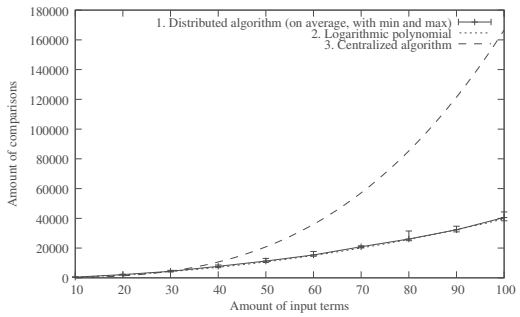

Figure 5: Experimental results

In a second step, the average complexity of the algorithm has been determined by experiments. The multi-agent system has been executed with randomly generated input data sets ranging from ten to one hundred terms. The given value is the average of comparisons made for one hundred of runs without any user interaction. It results in the plots of figure 5. The algorithm is then more efficient on average than the centralized algorithm, and its average complexity is below the worst case. It can be explained by the low probability that a data set forces the system to create only minimal groups (two items) or maximal ($n-1$ elements) for each step of reasoning. Curve number 2 represents the logarithmic polynomial minimizing the error with curve number 1. The highest degree term of this polynomial is in $n^2log(n)$, then our distributed algorithm has a $O(n^2log(n))$ complexity on average. Finally, let's note the reduced variation of the average performances with the maximum and the minimum. In the worst case for 100 terms, the variation is of 1,960.75 for an average of 40,550.10 (around 5%) which shows the good stability of the system.

3.2 Qualitative Evaluation

Although the quantitative results are interesting, the real advantage of this approach comes from more qualitative characteristics that we will present in this section. All are advantages obtained thanks to the use of an adaptive multi-agent system.

The main advantage to the use of a multi-agent system for a clustering task is to introduce dynamic in such a system. The ontologist can make modifications and the hierarchy adapts depending on the request. It is particularly interesting in a knowledge engineering context. Indeed, the hierarchy created by the system is meant to be modified by the ontologist since it is the result of a statistic computation. During the necessary look at the texts to examine the usage contexts of terms [2], the ontologist will be able to interpret the real content and to revise the system proposal. It is extremely difficult to realize this with a centralized "black-box" approach. In most cases, one has to find which reasoning step generated the error and to manually modify the resulting class. Unfortunately, in this case, all the reasoning steps that occurred after the creation of the modified class are lost and must be recalculated by taking the modification into account. That is why a system like ASIUM [6] tries to soften the problem with a system-user collaboration by showing to the ontologist the created classes after *each* step of reasoning. But, the ontologist can make a mistake, and become aware of it too late.

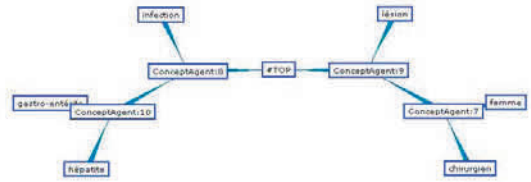

Figure 6: Concept agent tree after autonomous stabilization of the system

In order to illustrate our claims, we present an example thanks to a few screenshots from the working prototype tested on a medical related corpus. By using test data and letting the system work by itself, we obtain the hierarchy from figure 6 after stabilization. It is clear that the concept described by the term "lésion" (lesion) is misplaced. It happens that 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 statistics criteria.

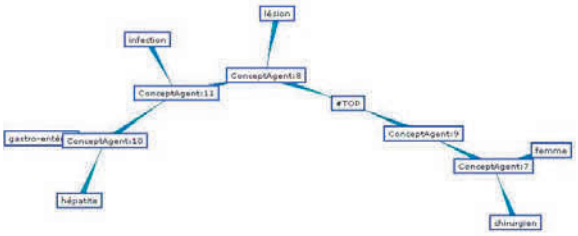

Figure 7: Concept agent tree after ontologist modification

Then, the ontologist replaces the concept in the right branch, by affecting "ConceptAgent:8" as its new parent. The name "ConceptAgent:X" is automatically given to a concept agent that is not described by a term. The system reacts by itself and refines the clustering hierarchy to obtain a binary tree by creating "ConceptAgent:11". The new stable state if the one of figure 7.

This system-user coupling is necessary to build an ontology, but no particular adjustment to the distributed algorithm principle is needed since each agent does an autonomous local processing and communicates with its neighborhood by messages.

Moreover, this algorithm can *de facto* be distributed on a computer network. The communication between agents is then done by sending messages and each one keeps its decision autonomy. Then, a system modification to make it run networked would not require to adjust the algorithm. On the contrary, it would only require to rework the communication layer and the agent creation process since in our current implementation those are not networked.

4. MULTI-CRITERIA HIERARCHY

In the previous sections, we assumed that similarity can be computed for any term pair. But, as soon as one uses real data this property is not verified anymore. Some terms do not have any similarity value with any extracted term. Moreover for leaf nodes it is sometimes interesting to use other means to position them in the hierarchy. For this low level structuring, ontologists generally base

their choices on simple heuristics. Using this observation, we built a new set of rules, which are not based on similarity to support low level structuring.

4.1 Adding Head Coverage Rules

In this case, agents can act with a very local point of view simply by looking at the parent/child relation. 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 concept agent X and $head(T_X)$ the set of all the terms that are head of at least one element of T_X. Thanks to those two notations we can describe the parent adequacy function $a(P, C)$ between a parent P and a child C:

$$a(P, C) = \frac{|T_P \cap head(T_C)|}{|T_P \cup head(T_C)|} \quad (3)$$

Then, the best parent for C is the P agent that maximizes $a(P, C)$. An agent unsatisfied by its parent can then try to find a better one by evaluating adequacy with candidates. We designed a complementary algorithm to drive this search:

When an agent C is unsatisfied by its parent P, it evaluates $a(B_i, C)$ with all its brothers (noted B_i) the one maximizing $a(B_i, C)$ is then chosen as the new parent.

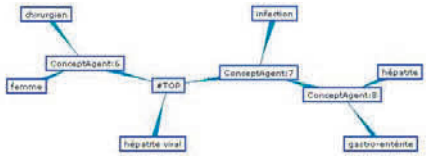

Figure 8: Concept agent tree after autonomous stabilization of the system without head coverage rule

We now illustrate this rule behavior with an example. Figure 8 shows the state of the system after stabilization on test data. We can notice that "hépatite viral" (viral hepatitis) is still linked to the taxonomy root. It is caused by the fact that there is no similarity value between the "viral hepatitis" term and any of the term of the other concept agents.

Figure 9: Concept agent tree after activation of the head coverage rule

After activating the head coverage rule and letting the system stabilize again we obtain figure 9. We can see that "viral hepatitis" slipped through the branch leading to "hepatitis" and chose it as its new parent. It is a sensible default choice since "viral hepatitis" is a more specific term than "hepatitis".

This rule tends to push agents described by a set of term to become leafs of the concept tree. It addresses our concern to improve the low level structuring of our taxonomy. But obviously our agents lack a way to backtrack in case of modifications in the taxonomy which would make them be located in the wrong branch. That is one of the point where our system still has to be improved by adding another set of rules.

4.2 On Using Several Criteria

In the previous sections and examples, we only used one algorithm at a time. The distributed clustering algorithm tends to introduce new layers in the taxonomy, while the head coverage algorithm tends to push some of the agents toward the leafs of the taxonomy. It obviously raises the question on how to deal with multiple criteria in our taxonomy building, and how agents determine their priorities at a given time.

The solution we chose came from the search for minimizing non cooperation within the system in accordance with the ADELFE method. 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:

$$\mu(A) = max(\mu_H(A), \mu_B(A), \mu_M(A)) \quad (4)$$

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 we detailed in the previous subsection
- if $\mu(A) = \mu_B(A)$ then A will use the distributed clustering algorithm (see section 3)
- if $\mu(A) = \mu_M(A)$ then A will process M_k immediately in order to help its sender

Those three cases summarize the current activities of our agents: they have to find the best parent for them ($\mu(A) = \mu_H(A)$), improve the structuring through clustering ($\mu(A) = \mu_B(A)$) and process other agent messages ($\mu(A) = \mu_M(A)$) in order to help them fulfill their own goals.

4.3 Experimental Complexity Revisited

We evaluated the experimental complexity of the whole multi-agent system when all the rules are activated. In this case, the metric used is the number of messages exchanged in the system. Once again the system has been executed with input data sets ranging from ten to one hundred terms. The given value is the average of message amount sent in the system as a whole for one hundred runs without user interaction. It results in the plots of figure 10.

Curve number 1 represents the average of the value obtained. Curve number 2 represents the average of the value obtained when only the distributed clustering algorithm is activated, not the full rule set. Curve number 3 represents the polynomial minimizing the error with curve number 1. The highest degree term of this polynomial is in n^3, then our multi-agent system has a $O(n^3)$ complexity

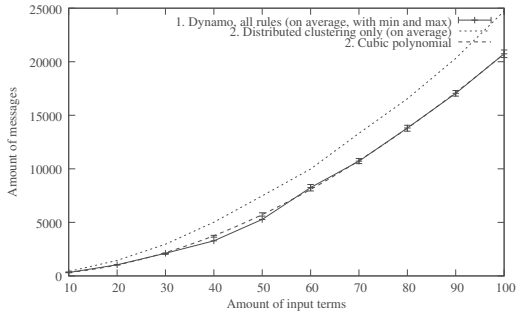

Figure 10: Experimental results

on average. Moreover, let's note the very small variation of the average performances with the maximum and the minimum. In the worst case for 100 terms, the variation is of 126.73 for an average of 20,737.03 (around 0.6%) which proves the excellent stability of the system.

Finally the extra head coverage rules are a real improvement on the distributed algorithm alone. They introduce more constraints and stability point is reached with less interactions and decision making by the agents. It means that less messages are exchanged in the system while obtaining a tree of higher quality for the ontologist.

5. DISCUSSION & PERSPECTIVES

5.1 Current Limitation of our Approach

The most important limitation of our current algorithm is that the result depends on the order the data gets added. When the system works by itself on a fixed data set given during initialization, the final result is equivalent to what we could obtain with a centralized algorithm. On the contrary, adding a new item after a first stabilization has an impact on the final result.

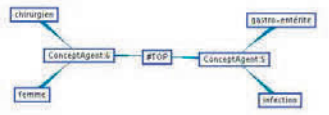

Figure 11: Concept agent tree after autonomous stabilization of the system

To illustrate our claims, we present another example of the working system. By using test data and letting the system work by itself, we obtain the hierarchy of figure 11 after stabilization.

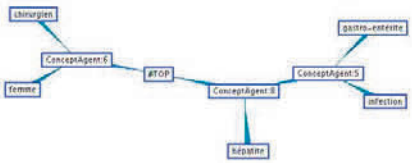

Figure 12: Concept agent tree after taking in account "hepatitis"

Then, the ontologist interacts with the system and adds a new concept described by the term "hepatitis" and linked to the root. The system reacts and stabilizes, we then obtain figure 12 as a result. "hepatitis" is located in the right branch, but we have not obtained the same organization as the figure 6 of the previous example. We need to improve our distributed algorithm to allow a concept to move along a branch. We are currently working on the required rules, but the comparison with centralized algorithm will become very difficult. In particular since they will take into account criteria ignored by the centralized algorithm.

5.2 Pruning for Ontologies Building

In section 3, we presented the distributed clustering algorithm used in the Dynamo system. Since this work was first based on this algorithm, it introduced a clear bias toward binary trees as a result. But we have to keep in mind that we are trying to obtain taxonomies which are more refined and concise. Although the head coverage rule is an improvement because it is based on how the ontologists generally work, it only addresses low level structuring but not the intermediate levels of the tree.

By looking at figure 7, it is clear that some pruning could be done in the taxonomy. In particular, since "lésion" moved, "ConceptAgent:9" could be removed, it is not needed anymore. Moreover the branch starting with "ConceptAgent:8" clearly respects the constraint to make a binary tree, but it would be more useful to the user in a more compact and meaningful form. In this case "ConceptAgent:10" and "ConceptAgent:11" could probably be merged.

Currently, our system has the necessary rules to create intermediate levels in the taxonomy, or to have concepts shifting towards the leaf. As we pointed, it is not enough, so new rules are needed to allow removing nodes from the tree, or move them toward the root. Most of the work needed to develop those rules consists in finding the relevant statistic information that will support the ontologist.

6. CONCLUSION

After being presented as a promising solution, ensuring model quality and their terminological richness, ontology building from textual corpus analysis is difficult and costly. It requires analyst supervising and taking in account the ontology aim. Using natural languages processing tools ease the knowledge localization in texts through language uses. That said, those tools produce a huge amount of lexical or grammatical data which is not trivial to examine in order to define conceptual elements. Our contribution lies in this step of the modeling process from texts, before any attempts to normalize or formalize the result.

We proposed an approach based on an adaptive multi-agent system to provide the ontologist with a first taxonomic structure of concepts. Our system makes use of a terminological network resulting from an analysis made by Syntex. The current state of our software allows to produce simple structures, to propose them to the ontologist and to make them evolve depending on the modifications he made. Performances of the system are interesting and some aspects are even comparable to their centralized counterpart. Its strengths are mostly qualitative since it allows more subtle user interactions and a progressive adaptation to new linguistic based information.

From the point of view of ontology building, this work is a first step showing the relevance of our approach. It must continue, both to ensure a better robustness during classification, and to obtain richer structures semantic wise than simple trees. From this improvements we are mostly focusing on the pruning to obtain better taxonomies. We're currently working on the criterion to trigger the complementary actions of the structure changes applied by our clustering algorithm. In other words this algorithm introduces in-

termediate levels, and we need to be able to remove them if necessary, in order to reach a dynamic equilibrium.

Also from the multi-agent engineering point of view, their use in a dynamic ontology context has shown its relevance. This dynamic ontologies can be seen as complex problem solving, in such a case self-organization through cooperation has been an efficient solution. And, more generally it's likely to be interesting for other design related tasks, even if we're focusing only on knowledge engineering in this paper. Of course, our system still requires more evaluation and validation work to accurately determine the advantages and flaws of this approach. We're planning to work on such benchmarking in the near future.

7. REFERENCES

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

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

[3] J. Bao and V. Honavar. Collaborative ontology building with wiki@nt. *Proceedings of the Workshop on Evaluation of Ontology-Based Tools (EON2004)*, 2004.

[4] C. Bernon, V. Camps, M.-P. Gleizes, and G. Picard. *Agent-Oriented Methodologies*, chapter 7. Engineering Self-Adaptive Multi-Agent Systems : the ADELFE Methodology, pages 172–202. Idea Group Publishing, 2005.

[5] C. Brewster, F. Ciravegna, and Y. Wilks. Background and foreground knowledge in dynamic ontology construction. *Semantic Web Workshop, SIGIR'03*, August 2003.

[6] D. Faure and C. Nedellec. A corpus-based conceptual clustering method for verb frames and ontology acquisition. *LREC workshop on adapting lexical and corpus resources to sublanguages and applications*, 1998.

[7] F. Gandon. *Ontology Engineering: a Survey and a Return on Experience*. INRIA, 2002.

[8] J.-P. Georgé, G. Picard, M.-P. Gleizes, and P. Glize. Living Design for Open Computational Systems. *12th IEEE International Workshops on Enabling Technologies, Infrastructure for Collaborative Enterprises*, pages 389–394, June 2003.

[9] M.-P. Gleizes, V. Camps, and P. Glize. A Theory of emergent computation based on cooperative self-organization for adaptive artificial systems. *Fourth European Congress of Systems Science*, September 1999.

[10] J. Heflin and J. Hendler. Dynamic ontologies on the web. *American Association for Artificial Intelligence Conference*, 2000.

[11] S. Le Moigno, J. Charlet, D. Bourigault, and M.-C. Jaulent. Terminology extraction from text to build an ontology in surgical intensive care. *Proceedings of the AMIA 2002 annual symposium*, 2002.

[12] K. Lister, L. Sterling, and K. Taveter. Reconciling Ontological Differences by Assistant Agents. *AAMAS'06*, May 2006.

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

[14] A. Maedche and S. Staab. Mining Ontologies from Text. *EKAW 2000*, pages 189–202, 2000.

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

[16] H. V. D. 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*, August 2006.