Self Organization in Coordination Systems using a WordNet-based Ontology

Danilo Pianini*, Sascia Virruso*, Ronaldo Menezes†, Andrea Omicini* and Mirko Viroli*
* ALMA MATER STUDIORUM – Università di Bologna, Italy
{danilo.pianini,sascia.virruso}@studio.unibo.it {andrea.omicini,mirko.viroli}@unibo.it
† Florida Institute of Technology, Department of Computer Sciences, USA
rmenezes@cs.fit.edu

***Abstract*—In today's data-intensive world, the need for data organization has increased dramatically. Distributed systems are dealing with unheard amounts of data arising primarily from the popularization of pervasive computing applications and the so-called "data-in-the-cloud" paradigm. Naturally, agent-coordination systems are affected by this data-increase phenomenon as they are often used as the basis for pervasive-computing frameworks and cloud-computing systems. There have been a few works on coordination system to include data self-organization (e.g. SwarmLinda) however they generally organize their data based on naive approaches where items are either completely similar or dissimilar (1|0 approach for matching of data). Although this approach is useful, in general-purpose systems where the diversity of data items is large, data items will rarely be considered as plainly similar, leading to a situation where data does not self-organize well. In this paper we move towards a general-purpose approach to organization based on an ontology-defined concept relationship in WordNet. In our approach, data items are seen as concepts that have relation to other concepts: tuples are driven towards one-another at rates that are proportional to the strength of tuple relationship. We demonstrate that this approach leads to a good mechanism to self-organize data in data-intensive environments.**

I. INTRODUCTION

In recent years we have witnessed the birth of a new paradigm in computer science based on the idea of self organization [1]. This trend emerged out of need: computer systems are becoming so large that system designers and programmers are unable to make decisions that can handle the dynamics of the system and at the same time maintain reasonable performance. Self-organization is proposed as a mechanism in which the system itself chooses, based on local conditions, what is the best decision to be taken leading to a near optimal solution to the problem at hand. The term "organization" generally refers to a situation in which the system tends to organize itself without a centralized decision maker. The literature is full of examples which include: ant-based heuristics for optimization [2], brood-sorting models for clustering [3], [4], and bird-flocking algorithms [5], [6].

One issue that has received increasingly more attention is data self-organization. We are living in a data-intensive era in which the amount of stored data in the world (in bytes) is approximating the Avogadro constant. Data is being collected for nearly all phenomena that one can think of. But for data to be useful it requires both organization and processing so that patterns can be found at a macro level; whereas the former relates to the subject of this paper, the latter falls in the realm of data modeling and visualization. Given the scale of these systems and the amount of data in question, standard policy-based approaches to data organization are just not adequate as the definition of the data-organization policies for large scale systems is beyond anyone's intellectual capability. The number of issues to consider, and the number of variables to define, make data organization in large scale an instance of a complex systems [7].

Coordination systems deal with synchronization and communication of processes/agents. In this paper we focus on coordination systems based on the Linda model [8], which were fruitfully used in the context of distributed, pervasive, and self-adapting scenarios – such as TuCSoN [9], TOTA [10], SwarmLinda [11], and EgoSpaces [12]. These models use a distributed associative memory as the sole form of communication and synchronization. Hence, data placement in the network nodes is essential to the systems efficiency (we briefly overview this problem in Section II).

We propose a self-organized approach for data that leads to an improvement to such coordination systems. As a first mechanism (Section III), we introduce a matching criterion (contrasting standard 1|0 match of Linda) in which tuple matching yields a value in between 0 and 1, based on a WordNet ontology for relating concepts (synonyms, hyponyms, hypernyms, and so on). Secondly, we introduce a nature-inspired pheromone/organization management of tuple location (Section IV): tuple requests are abstracted as pheromones to promote clustering of similar tuples near to where they are more frequently required, and organization-level considerations are used to provide self-adaptation: it both guarantees clustering even without requests and it avoids over-clustering [13]. The proposed approach has been implemented on top of TuCSoN coordination infrastructure [9], and was subjected to experimentation as described in Section V. The simulation results demonstrates conclusively that overall system tends to organize itself, as tuples get ordered by forming clusters, and that such clusters tend to be formed near nodes where those tuples are actually required, hence facilitating tuple retrieval.

However, it is important to note that we are not proposing a new approach to create tuples spaces but simply a way to organize tuples (data values stored in tuples spaces) in Linda-like systems.

II. Data Organization in Coordination Systems (Related Work)

Coordination systems have come a long way since their early days [14], [8]. Yet, one aspect of coordination has always caught the attention of researchers working on bringing the idea of Linda to a large audience: performance. When discussing performance of Linda systems, a major point of concern is scalability. Linda employs an indirect communication model in which processes do not communicate directly but rather via associative memories storing tuples—agents indirectly interact by inserting and retrieving those tuples. The same memories are used to store data associated with these processes in the form of tuples. Clearly, tuples are central to the workings of Linda, so their organization affects the scalability of the entire system. In very large systems, the location of tuples is extremely important to facilitate their retrieval, hence to promote performance of applications.

Coupled with the need for organizing data efficiently, we are faced with the problem of scale. When the amount of data is large, it is unrealistic to think that anyone can consistently decide what is the best organizational mechanism for tuples. The number of factors involved in this decision makes it a complex process. Hence large amounts of data requires a fundamentally different approach to organizing tuples in Linda-based systems. Self-organization has recently emerged as a science field offering a different approach to the understanding of complex processes [15]. Self-organization has also been used in many areas of computing including optimization [2], robotics [16], and even in coordination models [11] which are the topic of this paper.

The problem of defining "self-organizing coordination" has been addressed in [17], where it is framed as the problem of finding coordination laws (e.g. managing tuples) which are local, possibly timed and probabilistic, and which make some space-time patterns emerging at the global level. This approach has been the subject of several works. In [18], [19], self-organization approaches inspired to ant's brood sorting are exploited to replace tuples: the model we present here proceeds a step further, with a pheromone/organization approach solving intrinsic problems of the above approaches such as over-clustering. A different approach is developed in [10], where replacement of tuples is aimed at defining a *computational field* of tuples, used by agents to retrieve each other in mobile environments—hence it is physics-oriented rather than bio-inspired. In [20], chemical-like coordination rules are used to evolve and diffuse the population of tuples so as to mimic ecology-like behaviors. The WordNet approach we discuss in this paper can be used in all the above approaches, which in fact assume and require advanced forms of matching. Finally, note that several works are addressing the problem of semantically matching data, mostly exploiting Web Ontologies like OWL [21], [22], [23]: the work in this paper is complementary to them, and can ultimately lead to a more complete semantic matching for tuple space systems.

III. Semantic Tuple Matching

Tuples are ordered lists of typed elements, defined as $T(t_1, .., t_n)$, where T is the name of the tuple and $t_1, .., t_n$ defines a list of n arguments: for the sake of conciseness, we will use $\mathbf{t}_n$ to represent the list $t_1, .., t_n$, hence a tuple is written as $T(\mathbf{t}_n)$. The matching of tuples in coordination systems has traditionally used a discrete approach in which two tuples either match perfectly or not. A robust data organization, however, requires a softer matching approach in which the value of the matching varies continuously from 0 (completely different) to 1 (exactly the same). The reason is simple: data organization should consider the fact that tuples may not match but they are related. The relation between tuples may be given by many factors such as: being required by the same process, having the same geo-tag (if available) or, as proposed here, being semantically related. The assumption is simple and yet very general. If a process requires tuples of a particular kind, it is more likely to require tuples that have similar semantical relationship than tuples that do not. As an example: if a process is consuming tuples dealing to feline data (cats, tigers), it is more likely to become interested in other animals than in tuples related to computing terms because processes typically do not have wide scopes—so, they do not deal with many kinds of tuples. The assumption is crucial in the definition of the organization because it implicitly leads to an ontology for the relationship of terms—where, for instance, the concept DOG is more related to CAT than the concept MOTHERBOARD.

Indeed, in order to establish a correct semantic relationship between two concepts, the system bases itself on a *a priori* ontology which defines the meaning of a particular word. Generally speaking, an ontology defines what a concept is, how it could be described and the relations that it has with other concepts or the categories it belongs to.

In this paper we are dealing with tuples that may have multiple values. As mentioned before, tuples in general are sets of data which could be heterogeneous both in type and in value. They can be represented by a tree in which at each level the root is the functor of the tuple and the children are the arguments—which could be other tuples. So, in general, it is not so simple to map an ontology over this kind of model because the heterogeneity of data does not give us the global perspective of the relations that data have and how we can use them to describe a concept. For example, in a tuple *PERSON(John Smith, 40, teacher)* it may be clear to a human that this is tuple describing a person "John Smith" who is "40" years old, and works as a "teacher". But the mechanism we use to relate the terms is not explicit neither in the tuple nor in the ontology. "40" could represent his teacher-number in the school, not the age of the person. To avoid this kind of problem we decided to use *first-order tuples* in which every argument is a *constant* (term with arity zero) and introducing a default relation rule, such that every argument simply describes the main concept expressed by the name of the tuple.

A. WordNet as an Ontology

WordNet [24] is a lexical English database developed at Princeton University. In WordNet nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. WordNet defines every word according to the meaning(s) that it has in the English dictionary thus being an agreed-upon standard.

The adoption of this ontology in our approach is due to the fact that the network created by the structure of the database links explicitly words and concepts which are related semantically. According to the notion of tuple described before, if every element of a tuple follows this ontology, it is possible to establish which semantical relation exists between two terms giving us the possibility to compare tuples.

There are several APIs available to interface the access to WordNet. We opted to use MIT's Java WordNet Interface (JWI) [25] because it is extensively documented and connects the WordNet database to Java (used in our implementation).

B. Tuple Organization via Soft Matching

The approach proposed here entails a matching mechanism used solely for the purpose of tuple organization and movement. The approach does not affect how tuples are matched from the point of view of their retrieval by external agents. As we have pointed out, we would like the system to be self-organized so that once an agent requires the tuple, related ones should be in close proximity.

Our proposal on the relatedness of two tuples is to use what we call here *semantic matching*. The semantic matching of two tuples, $A(\mathbf{a}_n)$ and $B(\mathbf{b}_m)$, denoted by $\Delta(A(\mathbf{a}_n), B(\mathbf{b}_m))$ is the measure of how related these tuples are according to the adopted ontology. Here we assume that $\Delta(A(\mathbf{a}_n), B(\mathbf{b}_m)) \geq 0$ meaning that the more two tuples $A(\mathbf{a}_n)$ and $B(\mathbf{b}_m)$ are semantically related, the higher the value of Δ.

The mechanism used to compare two tuples starts from the premise that the value of the semantic relationship between tuples depends on the semantic relationship between the words (terms/concepts) that exist as fields of each tuple. We use this approach in our everyday lives; indeed, when people try to explain a concept, they normally base their explanation on an aggregation of related words that give more information about the concept requiring explanation. For example, when defining the animal *cat*, a person could add the information *animal*, *breed*, *mammal*, and so on. The mechanism we use in this paper attempts to extract this idea and quantify it in a numeric form.

Given tuples $A(\mathbf{a}_n)$ and $B(\mathbf{b}_m)$ with arities *n* and *m*, the algorithm makes the word-by-word comparison of all the possible combinations between every pieces of information in the tuple $A(\mathbf{a}_n)$ with the ones in $B(\mathbf{b}_m)$, including both the tuple name and the tuple arguments. For example given a tuple *CAT(animal)* and *DOG(animal)*, there are four possible combinations that are: *CAT-DOG*, *CAT-animal*, *DOG-animal* and *animal-animal*.

In tuple-based systems, the name of the tuple is more important as a definition of the tuple than its arguments, therefore, when comparing tuples, one should try to avoid giving the same importance to every combination of elements in the tuple because what we want is to compare "main concepts" explained by the tuple. There are then three different kinds of relationships: *name-to-name*, *name-to-argument* and *argument-to-argument*. Assuming that the concept expressed by a tuple is defined by the name of the tuple while the arguments are additional information that may help the explanation of a concept, it is obvious that the *name-to-name* relationship is the most important one. Then to evaluate the semantic relation that exists between two tuples, *argument-to-argument* relation is less important because if two tuples have the same name but different arguments it means that tuples are explaining the same concept but in a different way, so the weight to assign to the comparison between arguments must be lower than the one between the names. The last kind of relationship is the *name-to-argument*. This is the least important because in that case we are trying to compare words that have different roles: one is the concept explained by the tuple, while the other one is a further information which is something like a description. Even if this comparison is the least relevant, we want to use it in order to capture the semantic relationship that could exist between the description of a concept and its name.

Based on the argument above and after some empirical experiments, we decided to give a weight of 1.0 for the *name-to-name* (henceforth called *NN*) relationship, weight 0.8 for the *argument-to-argument* (henceforth called *AA*) relationship and weight 0.5 for the *name-to-argument* (henceforth called *NA*) relationship. After defining the weights, and denoting by $\delta(w_1, w_2)$ the semantical relationship between two words w_1, w_2, we can define the semantic match between two tuples $A(\mathbf{a}_n), B(\mathbf{b}_m)$ as follows:

$$\begin{aligned}\Delta(A(\mathbf{a}_n), B(\mathbf{b}_m)) = &\mathit{NN} \times \delta(A, B) + \\ &\mathit{NA} \times (\delta(\mathbf{a}_n, B) + \delta(A, \mathbf{b}_m)) + \\ &\mathit{AA} \times \delta(\mathbf{a}_n, \mathbf{b}_m)\end{aligned} \tag{1}$$

[illegible]

Note that $\delta(\mathbf{a}_n, B)$ is the sum of all $\delta(a_i, B)$ $(i \leq n)$, and $\delta(\mathbf{a}_n, \mathbf{b}_m)$ is the sum of all $\delta(a_i, b_j)$ $(i \leq n, j \leq m)$.

In order to normalize the value of Δ (to make it independent of the length of the tuples), we have to divide it by the max semantic matching value Δ_{max} which can be reached when the relationship that exists between every word is the strongest we consider (synonym) with a weight of 1.0 and can be calculated as

$$\begin{aligned}\Delta_{max}(A(\mathbf{a}_n), B(\mathbf{b}_m)) = &\mathit{NN} + \mathit{NN} \times (n + m) + \\ &\mathit{AA} \times (n \times m),\end{aligned} \tag{2}$$

where *n* and *m* represent the arities of the two tuples.

In this way we obtain the normalized version of semantic match, expressed as:

$$\hat{\Delta}(A(\mathbf{a}_n), B(\mathbf{b}_m)) = \frac{\Delta(A(\mathbf{a}_n), B(\mathbf{b}_m))}{\Delta_{max}(A(\mathbf{a}_n), B(\mathbf{b}_m))}. \tag{3}$$

The explanation above describes the process, however we have not defined yet how the δ function works—that is, its meaning. In order to correctly understand the mechanism we proposed, we need to further discuss the semantic relations between words.

C. Semantic Relationships

WordNet groups nouns, verbs, adjectives and adverbs into sets of cognitive synonyms (or *synsets*), each expressing a distinct concept and interlinked by different semantic and lexical relations. The relationships we use in our matching are:

- A *synonym* is a word with same basic meaning of the original word. This relation could be explained by the *"same as"* relationship. Examples of synonym pairs are *(picture,photo)* and *(table,mesa)*.
- A *hyponym* is a word or phrase whose semantic range is included within that of another word, its hypernym. Often in computer science this term can be explained using the *"is a"* relationship. For example, *bike is a hyponym of vehicle*.
- A *hypernym* of a word is another word that explains a more general concept. It's the opposite of Hyponym. Hence *vehicle is a hypernym of bike*.
- A *meronym* denotes a constituent part of, or a member of something. In knowledge representation languages, meronym is often expressed as *"part of"* relationship. For instance, *eye is a meronym of face*.
- A *holonym* denotes the whole part. It is the opposite of Meronym. It could be explained by the *"contains"* relationship. Similarly to the example before, *face is a holonym of eye*.

Figure 1 shows a simple example in WordNet and the relation between the terms. We show only the relations above since they are the only ones used in our algorithm and experiments.

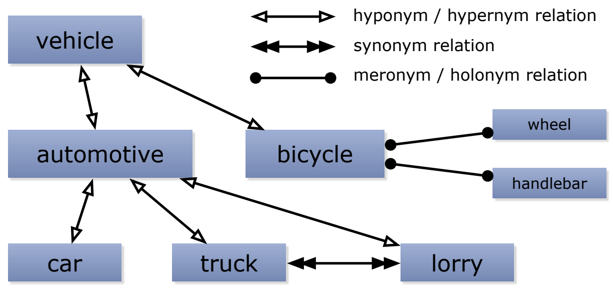


Fig. 1. Simple network demonstrating the various kids of relationships present in WordNet.

We use WordNet as a mechanism to define an ontology describing the semantic relationship of two given terms in a tuple. The mechanism we devised is loosely based on the work of Castano et al. [26]. We start by systematically searching whether two given terms/words in a tuple are synonyms, then hypernyms or hyponyms, and later if they are meronyms or holonyms. The motivation is that the *"same of"* relation between terms is a stronger relation than the *"is a"*, which in turn is a stronger than the *"part of"* or *"contains"*. All these relationships are mutually exclusive, so when the algorithm encounters one relationship, it stops without looking for the existence of others. Again similarly to the work of Castano et al., we attribute a weight to the type of relationships varying in the range $[0.0, 1.0]$ where 1.0 denotes the strongest relationship. We use 1.0 for the synonyms, 0.8 for hypernyms and hyponyms, 0.5 for meronyms and holonyms.

However, differently from the work of Castano et al., our algorithm does not create a thesaurus to store the terms to compare, but tries instead to find the relationships at run time. We do not mean to compare ontologies represented by graphs but concepts represented by tuples. In our algorithm, every piece of information has a different weight according to the role it has in the tuple (name or argument) and the kind of relationship (*NN*, *AA* or *NA*) involved. Our algorithm aims at finding a number which represents the level of relatedness between two concepts.

If two words are not directly related, the algorithm tries to find recursively the relations that may exist, so the result given by the δ function is the weight of the relationship found multiplied the depth of the recursion on the WordNet, that is, how distant in the network the words are apart. Given two words, the algorithm keeps one fixed and tries to search if the other occurs in the list of synonyms (or meronyms or hypernyms respectively) of the other one. Obviously, this mechanism is repeated twice exchanging the role of the words. In case of different values – due to the asymmetry of WordNet – we take the largest one since it represents the closest relationship between the two words.

Given the size of the WordNet and its incompleteness (it is nearly impossible to represent all relationships between every pair of words), we found that only in few instances we will get two words that are semantically related in a direct way as described before. As an example: if one thinks about *dog* and *cat* as concepts, it is clear that they are related in some way even if they do not represent the same concept or do not to fall in any of the categories listed earlier in this section. Indeed, they are both hyponyms of another concept, *animal*. Our approach needs then to take this into account so that the two terms are considered to be more related between themselves than two terms like, say, *computer* and *banana*. Along this line, we opted to find the *Lowest Common Hypernym (LCH)* of two words. In this case, after trying to find the straight relationships between two words, if there is no relation, we follow both two paths from every word with his hypernyms to find the common word that is the a hypernym of both words according to the WordNet. The algorithm is guaranteed to stop given the hierarchical structure of WordNet where the term *entity* is a hypernym of every other.

For instance, using WordNet 3.0 and searching for the first hypernyms ($\hookrightarrow$) of the word *dog* we find:

$$\begin{aligned} dog &= \textit{domestic dog}, \textit{canis familiaris} \\ &\hookrightarrow \textit{canine}, \textit{canid} \\ &\hookrightarrow \textit{carnivore} \\ &\hookrightarrow \ldots \end{aligned}$$

Searching for the first hypernyms of *cat* we find:

$$\begin{aligned} cat &= \textit{true cat} \\ &\hookrightarrow \textit{feline}, \textit{felid} \\ &\hookrightarrow \textit{carnivore} \\ &\hookrightarrow \ldots \end{aligned}$$

In the example above, we see that *carnivore* is the first hypernym that both *cat* and *dog* have in common. It is not a direct hypernym but one that is distant 2 steps from the words themselves. So the total path between the given words is made of 4 steps. In this case $\delta(dog, cat) = 0.8^4 = 0.4096$. Note that we use the sum of the distances between the words and the hypernym because we want the value of $\delta(dog, carnivore)$ – the direct comparison of a word and a hypernym – to give us a better similarity/matching. In general given two words w_1 and w_2 and defining l_1 and l_2 the distances between the words and their LCH in term of steps, the δ function is

$$\delta(w_1, w_2) = 0.8^{l_1+l_2}. \tag{4}$$

D. Semantic and Lexical Matching

Given Equation (3), it is possible to see that two tuples must be completely made by synonyms to obtain 100% of semantic match. This is correct, because the more two tuples try to explain the same concept in the same way and using similar words, the more the semantic match must be closer to 1.0. Indeed, two tuples that represent the same concept described with synonyms in the arguments lead to a perfect 1.0 semantic match.

In standard Linda-based matching, the exact matching of words is the sole comparison between terms: a lexical matching. If on the one hand the semantic match offers the possibility to reach the maximum with tuples made all by synonyms, on the other hand we want to also capture lexicographic match, which takes in account the word-by-word equality of every tuple as in the original Linda. The idea is to achieve a higher value in the matching for the case where the matching is perfect not only because of synonyms but also because the terms are exactly the same. In order to capture such a kind of relationship, we introduce the Φ function, which is a weighted average of a word-by-word lexicographic match

$$\Phi(A(\mathbf{a}_n), B(\mathbf{b}_m)) = \begin{cases} 0, & \text{if } n \neq m, \\ \frac{1}{n+1}\sum_{i=0}^{n} \phi(a_i, b_i), & \text{if } n = m \end{cases} \tag{5}$$

where a_0 and b_0 are used in the equation above to denote the names of tuples $A(\mathbf{a}_n)$ and $B(\mathbf{b}_m)$, respectively. The definition of equality function $\phi(a, b)$ is:

$$\phi(a, b) = \begin{cases} 1, & \text{if } a = b, \\ 0, & \text{if } a \neq b \end{cases} \tag{6}$$

In this function the equal (=) symbol has the lexicographical meaning. To be more precise, it means that two words have to be written in exactly the same way to be considered equal.

In conclusion, using (3) and (5) we combine semantic and lexical matching using α as a parameter that controls their relative importance for the specific system. This matching is represented by Equation 7.

$$\begin{aligned} \Lambda(A(\mathbf{a}_n), B(\mathbf{b}_m)) = \alpha\hat{\Delta}(A(\mathbf{a}_n), B(\mathbf{b}_m)) + \\ (1-\alpha)\Phi(A(\mathbf{a}_n), B(\mathbf{b}_m)) \end{aligned} \tag{7}$$

In our case, we found experimentally that $\alpha = 0.75$ provides a good balance between lexical and semantic matching of tuples. The following example shows a comparison between tuple *animal(dog)* and *animal(cat)*.

```
Comparing animal(dog) - animal(cat)
NN animal-animal val: 1.0
NA animal-cat val: 0.105
NA dog-animal val: 0.162
AA dog-cat val: 0.328
SEMANTIC matching is 0.569318582857143
LEXICAL  matching is 0.5
Calculating: 0.75 * 0.569318582857143 + 0.25 * 0.5
Tot: 0.5519889371428572
```

IV. Pheromone and Organization of Tuples

We now consider a tuple space architecture deployed into a network of nodes. The standard behavior would be to let tuples reside in the node where an agent inject them, while requests wander the network looking for the desired tuple.

In order to get a scalable self-organizing system, we design a different mechanism inspired by biological systems. In particular, we use a pheromone-based approach which – exploiting the two base mechanisms of diffusion and evaporation – makes each node able to excite the neighbors and to have a sort of memory of the actions. The pheromone is emitted every time a tuple is searched, thus ensuring that the data is attracted by the searches as soon as they start. In order to avoid the creation of multiple pheromones with the same "meaning" which would have lead to performance issues, every time the system needs to emit pheromone the tuple is broken and a different pheromone for every part is emitted. For example, the search for a tuple $A(b, c, d)$ is supposed to be semantically the same of $A(d, b, c)$. With the adopted system, four pheromones are generated: *pheromone*(A, i), *pheromone*(b, j), *pheromone*(c, j), *pheromone*(d, j) instead of a pheromone for each tuple, where i and j are the amount of pheromone for each value.

The amount of pheromone used is constant in every performed search. However, such an amount (100 units in our experiments) is equally divided among all the arguments in the tuple except for the name of the tuple, which gets twice the amount used for the arguments. The idea here is to capture the fact that if a tuple $A(\mathbf{a}_n)$ has n arguments and a tuple $B(\mathbf{b}_m)$ has m arguments and $n \ll m$, the amount of pheromone, τ for each argument is such that $\tau(a_i) \gg \tau(b_i), \forall i$ in $\mathbf{a}_n, \mathbf{b}_m$, and $\tau(A) = 2 \times \tau(a_i), \forall i$ in $\mathbf{a}_n$. Note that this means that tuples with a smaller number of arguments get proportionally more pheromone per argument, in fact each argument of $A(\mathbf{a}_n)$ will get $1/(n+1)$ of the total amount of pheromone being used with the name of the tuple getting $2/(n+1)$ of the total.

Every fixed time interval (5 seconds in our experiments), each node checks for the presence of pheromones. If some is found, the spread and evaporation process starts. For every kind of pheromone found, $\tau(t_i)$, the evaporation process is applied. This is similar to the pheromone update rule found in ACO algorithms [2], which states that $\tau(t_i) = \rho\tau(t_i)$, where ρ is the evaporation rate—e.g. set at 0.9. The pheromone left is spread equally among the n neighboring nodes: each node receives $(1-\rho) \times \tau(t_i)/(n+1)$ and the remaining quantity evaporates.

Pheromone systems enable the clustering of tuples around nodes that search for the same kind of tuple, however they do not cover two issues: over-clustering and tuple organization in a situation where no searches occur. In order to solve these problems we introduced a system able to dynamically detect the maximum number of tuples allowed for each tuple space, and which is able to excite and move the tuples which are not related to the others present in the tuple space.

First of all, we defined a measure of local organization of a tuple space based on the work of Casadei et al. [18]. Given a tuple space with N tuples, denoting with T_i the i-th tuple, the local organization, $S_\ell(k)$, of a tuple space k is given by:

$$S_\ell(k) = 1 - \frac{\sum_{i,j} \Lambda(T_i, T_j)}{N(N-1)}. \tag{8}$$

Given the equation above, one can obtain the level of organization of the entire system, S_e as an average of the local organization levels:

$$S_e = \frac{\sum_k S_\ell(k)}{K}. \tag{9}$$

Similar to the concept of entropy, the equations above assume that the lower values reflect high organization. Nodes use the organization level as a way to decide how many tuples they can store. Our approach is that if a node is poorly organized it becomes "excited" and the system tries to move them towards the neighbors. The tuple space k, with local organization level $S_\ell(k)$, periodically selects a tuple to move to a neighbor. The periodicity of the movement is based on the organization level, the higher the order, the slower the tuples are selected to move. As the organization increases the system becomes less active, hence leading to a convergence where tuples are rarely selected to move. This mechanism of selecting tuples to move enables tuple clustering even if there are no nodes searching.

The choice of the tuple to move is random. The destination is chosen stochastically from the neighbors based on the amount and kind of the pheromones each neighbor contains. We use pheromones instead of the actual neighbor tuple information because the idea here is that if there is a short path to a node that wants the tuple, the path may include nodes that may not be interested in the tuple but contain high pheromone and hence the tuple should flow through these neighboring nodes following the pheromone trail. Furthermore, when the node receive a tuple which is different from theirs, their organization decreases, leading to a reduction of the time between the tuple movements and increasing the probability that the tuple is moved again.

Briefly, internal organization evaluation makes the system excited when it is far from a desirable configuration, while the pheromone subsystem tends to let the system evolve towards higher organization: the two subsystems work closely together.

V. Experiments and Results

A. Network Topology

As a first issue, we describe topological aspects of the proposed approach and experiments. We note that it is important to use network topologies that are likely to appear in the real world. In our case, we have an open network in which nodes should be able to join and leave execution at any time. One may argue that a fully connected network would provide a good solution but the cost of such network may be prohibitive and, most importantly, very few real networks are actually fully connected. A better approach is to use those results that demonstrate that real networks – Web, Internet, friendship – are scale free [27], meaning that the probability of a node having k connections, $P(k)$, is given by $k^{-\lambda}$. In order to build a network with these characteristics we use the BA model [27] in which nodes are added to network according to a preferential attachment. Each node, when it tries to connect to another one, chooses n neighbors based on a probability distribution driven by the number of connections the nodes already have in the network—the standard approach is $n = 1$ but here we choose $n = 2$ to improve the network tolerance to failures.

For the purpose of our experiments we use a preferential attachment variation which considers more than just the number of degrees as proposed in the BA model. When a new node is to be added in the network, it considers *degree of mobility*, *numbers of connection* and *location* of the nodes already existing in the network.

- The *degree of mobility* is an indication of how often a node changes its physical position. High values indicate high mobility (e.g. mobile and smart phone) while low numbers indicate low mobility (e.g. a desktop PC).
- The *degree* is the number of link that a node has in the network.
- The *location* represents the GPS coordinates of the node.

The parameters above are used to decide the locations a new node will connect to according to equation

$$P(new, j) = \frac{\frac{n_j m_j}{d(j,new)}}{\sum_{i=1}^{n} \frac{n_i m_i}{d(i,new)}}, \tag{10}$$

where $P(new, j)$ is the probability that the node new will connect to a reachable node j. This probability depends on the number of connections that j has, n_j; its degree of mobility, m_j; and the Euclidean distance between j and new, $d(j, new)$. Equation 10 is used for every existing node j in the graph, allowing us to choose the two connecting nodes using a *Fitness Proportionate Selection* (aka *Roulette Wheel*). The equation

TABLE I
SEMANTIC MATCHING VALUES OF EACH COUPLE OF WORDS ROUNDED WITH 3 DECIMAL PLACES

Λ	Dog	Cat	Bird	Turtle	Animal	Banana	Orange	Apple	Pear	Food	Plaza	Highway	Road	Street	Avenue
Dog	1.000	0.410	0.174	0.207	0.323	0.168	0.118	0.097	0.097	0.189	0.118	0.110	0.138	0.107	0.097
Cat	0.410	1.000	0.295	0.410	0.210	0.148	0.121	0.097	0.097	0.189	0.101	0.127	0.159	0.134	0.139
Bird	0.174	0.295	1.000	0.328	0.512	0.328	0.198	0.328	0.328	0.640	0.172	0.118	0.148	0.151	0.172
Turtle	0.207	0.410	0.328	1.000	0.198	0.210	0.121	0.107	0.107	0.210	0.101	0.127	0.159	0.110	0.139
Animal	0.323	0.210	0.512	0.198	1.000	0.328	0.151	0.151	0.168	0.269	0.121	0.151	0.189	0.121	0.159
Banana	0.168	0.148	0.328	0.210	0.328	1.000	0.512	0.640	0.640	0.512	0.077	0.097	0.121	0.088	0.097
Orange	0.118	0.121	0.198	0.121	0.151	0.512	1.000	0.512	0.512	0.410	0.151	0.132	0.165	0.086	0.069
Apple	0.097	0.097	0.328	0.107	0.151	0.640	0.512	1.000	0.640	0.512	0.110	0.107	0.134	0.097	0.110
Pear	0.097	0.097	0.328	0.107	0.168	0.640	0.512	0.640	1.000	0.512	0.110	0.107	0.134	0.097	0.110
Food	0.189	0.189	0.640	0.210	0.269	0.512	0.410	0.512	0.512	1.000	0.215	0.148	0.185	0.189	0.215
Plaza	0.118	0.101	0.172	0.101	0.121	0.077	0.151	0.110	0.110	0.215	1.000	0.159	0.198	0.248	0.165
Highway	0.110	0.127	0.118	0.127	0.151	0.097	0.132	0.107	0.107	0.148	0.159	1.000	0.800	0.340	0.258
Road	0.138	0.159	0.148	0.159	0.189	0.121	0.165	0.134	0.134	0.185	0.198	0.800	1.000	0.640	0.290
Street	0.107	0.134	0.151	0.110	0.121	0.088	0.086	0.097	0.097	0.189	0.248	0.340	0.640	1.000	0.800
Avenue	0.097	0.139	0.172	0.139	0.159	0.097	0.069	0.110	0.110	0.215	0.165	0.258	0.290	0.800	1.000

makes nodes prefer to connect to others with low mobility, high number of connection, and that are physically near.

B. Framework Architecture

The chosen architecture is made of three layers, incorporating the ability of generating the above topologies. The lower level is called *Node Research Manager* (NoRMan) and manages the low-level aspects related to the real network structure, ensuring that each node has an up-to-date list of all the physically reachable nodes, each containing information useful to the upper layers such as position (in terms of longitude and latitude), degree of mobility and number of active connections. In order to accomplish its task, NoRMan automatically scans all IP addresses starting from the nearest to the one in use. Once a node is found, its information is spread among the nodes in the network. NoRMan supports geo-localization through a specific interface. An external agent stub is used for supporting implementation of GPS geo-localization. Currently, the agent only performs a table lookup in a IP address to location database. If the lookup fails, it generates random position.

The *Topology Abstractor* (TopA) runs over NoRMan, and is composed of a server and a *Barabasi Network Builder* (BaNBu). Its target is to exploit the informations which NoRMan carries to realize and keep updated a virtual Barabasi Network implementing the algorithm explained in section V-A.

The third layer is the core system, which exploits the TuCSoN [9] 1.9.1 framework in order to build distributed tuple spaces and to implement our self-organization mechanisms. To this end, the intrinsic ability of TuCSoN to program the behavior of tuple spaces is exploited—declarative transition rules can be injected in the space which intercept communication events and accordingly update the tuple set. In fact, our framework aims at being a self-organized, fault-tolerant virtual Barabasi-Albert network where each node represents a tuple space, and where the tuples are able to move within this network without any user intervention. According to the mechanisms in Section IV, tuples that are semantically related must form a cluster, making sure that clusters are formed and maintained in close proximity to where users need them. More precisely, what our experiments demonstrate is that if someone is searching for data of a certain kind, all related data move toward that user; the speed of such movement – how fast it converges – is proportional to the relatedness of the data.

C. Simulation Results

All the experiments in this paper were performed on a hardware system composed of three Dell Optiplex GX280, each with an Intel Pentium 4 3.00GHz processor and 1GB RAM running Sabayon Linux 4.2G and four iMacs. In addition we connected to the network two notebooks: an Acer 5920G (Intel Core 2 Duo 2.00GHz, 4GB RAM) with Sabayon Linux 4.2G and a Macbook Pro (Intel Core Duo 2.00GHz, 1.5GB RAM) with MacOS X.

Two experiments were made in order to verify the self-organizing properties of the framework. Both experiments use the same set of tuples and the same initial configuration. As soon as it starts, the framework builds a scale-free network using the BA model explained above. Each node starts containing the same 75 tuples. The values are organized in 3 categories. The tuple spaces (nodes) are initialized with 25 tuples of each category divided equally amongst the values available. The categories are: C_1 (*animal*, *dog*, *cat*, *bird*, *turtle*), C_2 (*apple*, *banana*, *pear*, *orange*, *food*), and C_3 (*avenue*, *highway*, *street*, *plaza*, *road*). Both the experiments were run on the nine nodes described above.

The Λ for each pair of tuples is computed locally in each tuple space—the inherent cost ranges from few milliseconds to few hundred milliseconds depending on the tuple size. Results of matching are cached for future usages, so that the cost of matching quickly becomes negligible. Table I shows the values calculated between each pair (using Equation 7).

In the first experiment each node starts searching for a specific kind of tuple, as described in Table II, last column. What we expect is to see the tuples move and reach the node which is searching for them while avoiding over-clustering. We ran the experiment three times, recording for each node the local organization level and their contents. The number associated to each node is a unique identifier (e.g. the IP address). For the first run we reached the values in Table

TABLE II

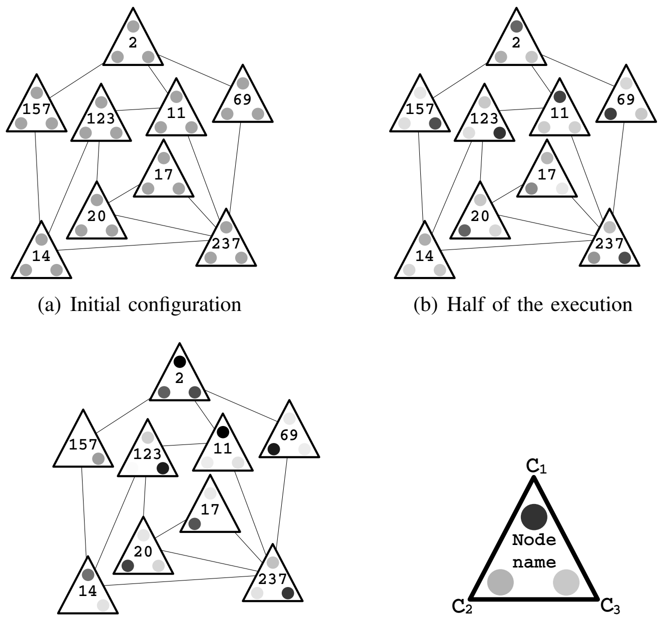
EVOLUTION OF CLUSTERING AT THREE DIFFERENT TIME POINTS: BEGINNING, HALFWAY THROUGH THE EXECUTION AND AT THE END OF THE EXECUTION. THE SYSTEM WAS LEFT EXECUTING AND AFTER STARTING FROM A UNIFORM DISTRIBUTION OF TUPLES IN EACH LOCATION WE SEE AN INCREASE IN THE ORGANIZATION LEVELS SINCE LOCATIONS SEEM TO CLUSTER NODES OF SPECIFIC SIDES. FIGURE 2 SHOWS A PICTORIAL VIEW OF THIS TABLE.

	initial			Half			Final			
Node	C_1	C_2	C_3	C_1	C_2	C_3	C_1	C_2	C_3	Search
2	25	25	25	**42**	16	22	**70**	49	44	Bird
11	25	25	25	**52**	13	14	**67**	8	5	Cat
14	25	25	25	**22**	15	11	**39**	8	0	Turtle
17	25	25	25	20	7	**32**	6	0	**47**	Avenue
20	25	25	25	15	11	**43**	7	11	**52**	Highway
69	25	25	25	11	15	**53**	6	5	**63**	Road
123	25	25	25	15	**51**	9	13	**62**	1	Pear
157	25	25	25	7	**48**	9	0	**27**	0	Apple
237	25	25	25	18	**49**	29	17	**55**	8	Orange

II and the graphs in Figure 2. We started from disorganized (unclustered) state in Figure 2(a) where all nodes have the same number of tuples from each category, and we reached the point in Figure 2(c) where clusters are formed for each category. At the end of the experiment all the nodes but node 2 have formed clusters of similar tuples. Looking at Table II, we can conclude from the high number of tuples of category C_2 and C_3 that node 2 is subject to high traffic. Additionally, in Figure 3(a) we can see that the level of organization of the whole system increases rapidly with time. Looking carefully at Figure 3(a), which is related to the first run, we can notice that there are some nodes with a very high organization level and others that remain highly disorganized. Such disorganized nodes are the most connected ones: they have to support the highest amount of traffic and as a consequence they often contain a more diverse set of tuples. Figures 3(b) and 3(c) respectively refer to other runs in the system. Their behavior is similar but not identical due to the stochastic nature of the self-organized process. However they clearly show S_e decreasing.

In the second experiment, a single node searches continuously for the tuple *dog*. What we expect to see is that the tuples of kind *dog* and the related ones cluster on the node where the search is being performed. We also expect that the neighbor nodes contain tuples similar to each other but not related to *dog*. This is a good test for the over-clustering avoidance algorithm: if it would not work well, we will see a single big cluster containing all categories. In order to evaluate the effect of the pheromone approach, we chose to perform the search in nodes with lower connectivity (node 157 for the first run and node 237 for the second and third runs). In order to measure how much the cluster is related with the searched tuple, we used the concept of matching between the searched tuple and all other tuples in the tuple space (node).

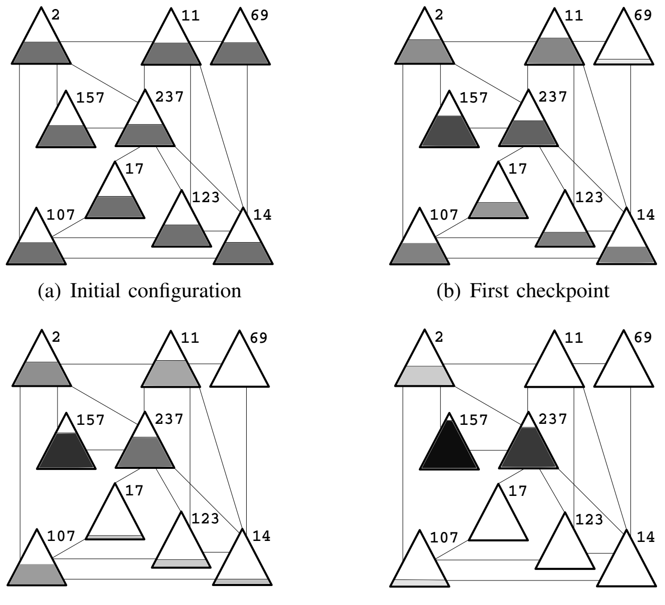
In the Figure 4 we can see the affinity value for each node. The node where the search is being performed is shown in black in Figures 4(a), 4(b) and 4(c). Figures 5, 6, 7 show the clusters formation in the actual graph. There, the more full is the triangle the larger the number of tuples in that node, while gray saturation indicates the how similar the tuples on that node are to the tuple *dog*. The figures show the state of



(a) Initial configuration (b) Half of the execution

(c) Final configuration

Fig. 2. Visualization of II. Each circle inside the triangle represent a category of tuple and the gray level the amount of tuples in that category.



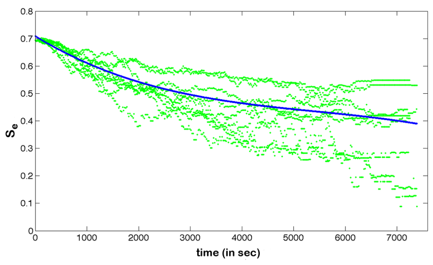
(a) Initial configuration (b) First checkpoint

(c) Second checkpoint (d) Final Configuration

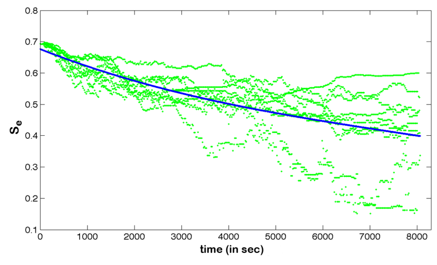
Fig. 5. Relation of tuple *dog* with all the nodes in the network (first run).

the network at 4 different execution points. One can clearly see that in all the cases there are clusters being formed, with a few nodes left empty and one or two nodes containing most of the nodes related to the tuple *dog*. The nodes containing more nodes related to *dog* are the ones in which the search is being executed while other nodes contain tuples unrelated with the searched one.

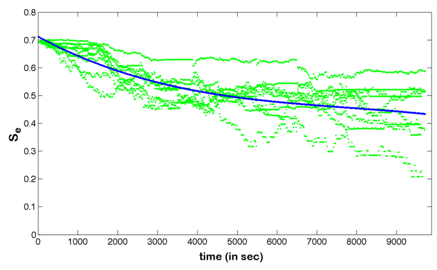
By analyzing the results we can assert that, while the second and third runs have a similar behavior featuring a



(a) Run 1

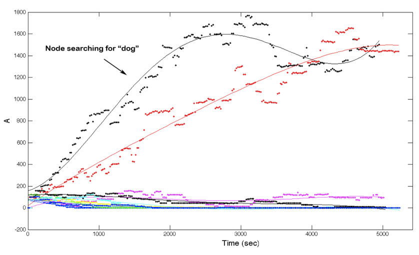


(b) Run 2

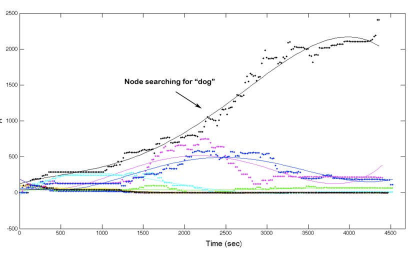


(c) Run 3

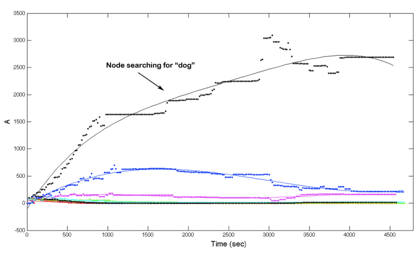
Fig. 3. System organization level E, S_e (shown as a solid), consistently decreases over time. This means that the system is self-organizing and that the proposed approach lead to an organization of tuples based solely on the system activities. The other lines refer to $S_\ell(k)$ for each node k in the network.



(a) Run 1

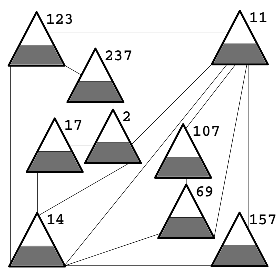


(b) Run 2

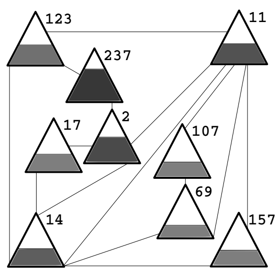


(c) Run 3

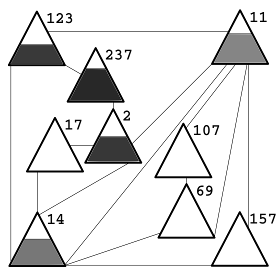
Fig. 4. Affinity with tuple *dog* in three runs of the second type experiment. The lines show how similar the tuples in each node are to the tuple *dog*.



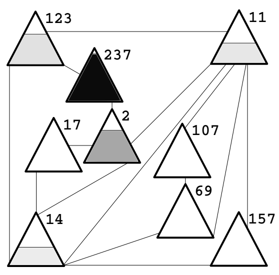
(a) Initial configuration



(b) First checkpoint

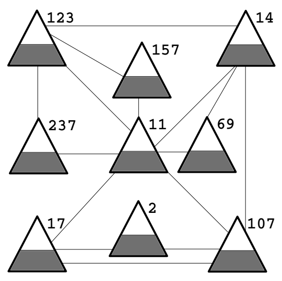


(c) Second checkpoint

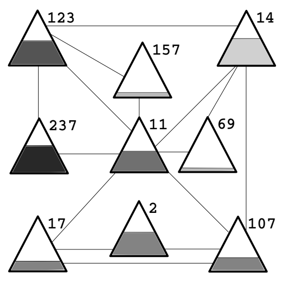


(d) Final Configuration

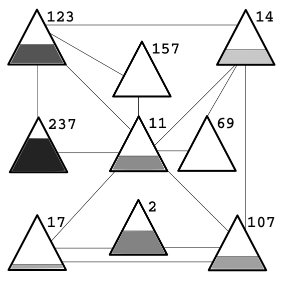
Fig. 6. Relation of tuple *dog* with all the nodes in the network (second run).



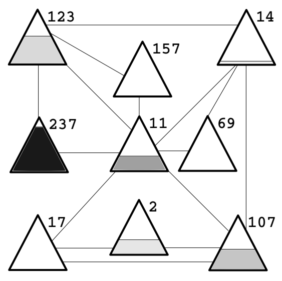
(a) Initial configuration



(b) First checkpoint



(c) Second checkpoint



(d) Final configuration

Fig. 7. Relation of tuple *dog* with all the nodes in the network (third run).

single cluster, in the first run there are two clusters instead. The reason of this behavior is the proximity of node 237, which is very connected to node 157 where the search is being performed. This means that the node 237 has an high amount of pheromone (because of its proximity) and high traffic (because of the number of connections it has). These two factors cause the formation of a cluster which is quite related with the searched tuple. Yet, it is possible to see from Figure 5 that the node executing the search still has a higher relation to the tuple *dog*. Also, one can see that other clusters of tuples not related to *dog* are formed. For instance, in 7(c) we see other clusters in nodes 123 and node 107.

VI. Conclusion and Future Work

We explored a new way to match tuples, according to a semantic model contrasting the classical $0|1$ that is typically adopted. Using this kind of algorithm, we were able to build a general purpose self-organizing coordination infrastructure which, relying on pheromone and organization levels evaluation mechanisms, makes similar tuples cluster near to where they are typically required. The experimental results show that the whole system tends to organize itself over time, confirming the appropriateness of our approach.

In the future, we plan to move towards experiments with more complex tuples in real large scale networks, which can also imply a deep study on the overhead introduced by the framework and consequently the system performance optimization. Other improvements could regard the possibility of making tuples diffuse and not only simply move, according to the scope of the application, and automatically detect the tuple "life time" and consequently remove the obsolete ones. The improvements related to the semantic match involves the inclusion of adjectives and verbs, the improvement of the mechanisms for searching the LCH – because finding hypernyms of hypernyms is different than search all hypernyms of a word – the improvement of the lexical matching, and finally the integration with semantic approaches based on Description Logics [22], as typically happens in the context of the Semantic Web.

The concept of similarity depends on the ontology. Here we assume that we are dealing with concepts whose similarity depends on how similar the concepts are according to a general idea defined in WordNet. However, one could envision a system in which the ontology itself could be defined to handle cases in which the similarity is defined in other terms. For instance, if we assume similarity defined by the number of overlapping letters, one could say that the concept DOG and CAT are not similar because they do not have letters in common while DOG and MOTHERBOARD are similar because the letters D and O are common. The study of this approach is left as future work.

References

[1] M. Mamei, R. Menezes, R. Tolksdorf, and F. Zambonelli, "Case studies for self-organization in computer science," *Journal of Systems Architecture*, vol. 52, no. 8-9, pp. 443–460, 2006.

[2] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, vol. 26, no. 1, pp. 29–41, 1996.

[3] J.-L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chretien, "The dynamic of collective sorting robot-like ants and ant-like robots," in *Proceedings of the First International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3*. Cambridge, MA: MIT Press, 1991, pp. 356–365.

[4] E. D. Lumer and B. Faieta, "Diversity and adaptation in populations of clustering ants," in *SAB94: Proceedings of the third international conference on Simulation of adaptive behavior : from animals to animats 3*. Cambridge, MA, USA: MIT Press, 1994, pp. 501–508.

[5] J. Toner and Y. Tu, "Flocks, herds and schools: A quantitative theory of flocking," *Physical Review E.*, vol. 58, pp. 4828–4858, 1999.

[6] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *Automatic Control, IEEE Transactions on*, vol. 51, no. 3, pp. 401–420, march 2006.

[7] M. Mitchell, *Complexity: A Guided Tour*. Oxford University Press, 2009.

[8] D. Gelernter, "Multiple tuple spaces in linda," in *Proceedings of the Parallel Architectures and Languages Europe*, vol. II: Parallel Languages. London, UK: Springer-Verlag, 1989, pp. 20–27.

[9] A. Omicini and F. Zambonelli, "Coordination for Internet application development," *Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 3, pp. 251–269, Sept. 1999.

[10] M. Mamei and F. Zambonelli, "Programming pervasive and mobile computing applications: the tota approach," *ACM Trans. Software Engineering and Methodology*, vol. 18, no. 4, 2009.

[11] R. Tolksdorf and R. Menezes, "Using swarm intelligence in linda systems," in *Proceedings of the Fourth International Workshop Engineering Societies in the Agents World (ESAW'03)*, ser. Lecture Notes in Artificial Intelligence, A. Omicini, P. Petta, and J. Pitt, Eds., vol. 3071. Springer Verlag, Oct. 2003.

[12] C. Julien and G.-C. Roman, "Egospaces: Facilitating rapid development of context-aware mobile applications," *IEEE Trans. Software Eng.*, vol. 32, no. 5, pp. 281–298, 2006.

[13] M. Casadei, R. Menezes, R. Tolksdorf, and M. Viroli, "On the problem of over-clustering in tuple-based coordination systems," in *Proceedings of the First International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE Computer Society, 2007, pp. 303–306.

[14] D. Gelernter, "Generative communication in Linda," *ACM transactions in Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112, 1985.

[15] S. Kauffman, *At Home in the Universe: The Search for the Laws of Self-Organization and Complexity*. Oxford University Press, 1996.

[16] R. Pfeifer, M. Lungarella, and F. Iida, "Self-organization, embodiment, and biologically inspired robotics," *Science*, vol. 318, no. 5853, pp. 1088–1093, Nov. 2007.

[17] M. Viroli, M. Casadei, and A. Omicini, "A framework for modelling and implementing self-organising coordination," in *24th Annual ACM Symposium on Applied Computing (SAC 2009)*, S. Y. Shin, S. Ossowski, R. Menezes, and M. Viroli, Eds., vol. III. Honolulu, Hawai'i, USA: ACM, 8–12 Mar. 2009, pp. 1353–1360.

[18] M. Casadei, R. Menezes, M. Viroli, and R. Tolksdorf, "A self-organizing approach to tuple distribution in large-scale tuple-space systems," in *Second International Workshop on Self Organizing Systems*, ser. Lecture Notes in Computer Science, vol. 4725. Springer Verlag, Sept. 2007, pp. 146–160.

[19] M. Casadei and M. Viroli, "Applying self-organizing coordination to emergent tuple organization in distributed networks," in *Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO'08)*, S. Brueckner, P. Roberson, and U. Bellur, Eds. Venice, Italy: IEEE Computer Society, 20–24 Oct. 2008, pp. 213–222.

[20] M. Viroli and M. Casadei, "Biochemical tuple spaces for self-organising coordination," in *Coordination Languages and Models*, ser. LNCS, J. Field and V. T. Vasconcelos, Eds. Springer-Verlag, June 2009, vol. 5521, pp. 143–162, 11th International Conference (COORDINATION 2009), Lisbon, Portugal, June 2009. Proceedings.

[21] K. Fujii and T. Suda, "Semantics-based dynamic web service composition," *Int. J. Cooperative Inf. Syst.*, vol. 15, no. 3, pp. 293–324, 2006.

[22] E. Nardini, M. Viroli, and E. Panzavolta, "Coordination in open and dynamic environments with tucson semantic tuple centres," in *25th Annual ACM Symposium on Applied Computing (SAC 2010)*, S. Y. Shin, S. Ossowski, M. Schumacher, M. Palakal, C.-C. Hung, and D. Shin, Eds., vol. III. Sierre, Switzerland: ACM, 22–26 Mar. 2010, pp. 2037–2044, awarded as Best Paper.

[23] R. Tolksdorf, L. J. B. Nixon, and E. P. B. Simperl, "Towards a tuplespace-based middleware for the Semantic Web," *Web Intelligence and Agent Systems*, vol. 6, no. 3, pp. 235–251, 2008.

[24] C. Fellbaum, Ed., *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[25] M. A. Finlayson, "MIT Java Wordnet Interface," http://projects.csail.mit.edu/jwi/.

[26] S. Castano, A. Ferrara, and S. Montanelli, "Matching ontologies in open networked systems: Techniques and applications," *Journal on Data Semantics (JoDS)*, vol. V, 2005.

[27] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.