Self-Organized Control of Knowledge Generation in Pervasive Computing Systems

Gabriella Castelli
DISMI
University of Modena and Reggio Emilia
42100 Reggio Emilia, Italy
gabriella.castelli@unimore.it

Ronaldo Menezes
Computer Sciences
Florida Tech
Melbourne, Florida, USA
rmenezes@cs.fit.edu

Franco Zambonelli
DISMI
University of Modena and Reggio Emilia
42100 Reggio Emilia, Italy
franco.zambonelli@unimore.it

ABSTRACT

Pervasive computing devices (e.g., sensor networks, localization devices, cameras, etc.) are increasingly present in every aspect of our lives. These devices are able to generate enormous amounts of data, from which knowledge about situations and facts occurring in the world can be inferred; inference can also be done by combining data items and generating new (higher-level) ones. Such data and knowledge is of extreme importance for to context-aware and mobile services. However, we are left with the problem that the possibly huge amount of data and knowledge generated can be very hard to be analyzed and made usable in real-time. The core of the problem in today's pervasive environments lies between the ability to extract meaningful (useful) knowledge from the data while making sure the total amount of data does not become overwhelming to the system. This paper focus on this trade-off using (without loss of generality) the W4 model for contextual data as a case study. Starting from the basic mechanism by which the W4 model autonomously generate new knowledge, the paper shows how this can generate knowledge overflow, and propose a method to select—in a self-organizing way—what kinds of knowledge should be generated based on their importance; hence preventing knowledge overflow. Experimental results are reported to support our arguments and proposals.

Categories and Subject Descriptors

J.9 [**Mobile Applications**]: Pervasive Computing; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*

Keywords

Pervasive Computing, Self-Organization, Context-Awareness, Knowledge Engineering

1. INTRODUCTION

Pervasive computing is here to stay [10]. We already have smart phones, on-board computers in vehicles, computer-based appliances and localization devices being *pervasive* to every aspect of our lives. Very soon, sensor networks, RFID tags, cameras with intelligent vision system will complete the picture, and will make it possible to deliver a wide variety of innovative digital services related to our everyday activities. In particular, all such devices will be able to produce diverse information related to fact and events occurring in the physical and social worlds [6], which we can directly exploit to better understand and interact with the world, as well as be exploited for the development of location-based and context-aware mobile services.

Of course, the more pervasive computing devices are made in our environment, the larger the amount of data that will be generated and made available about fact and events occurring in the world. For such data to be used in real-time by users and services, it must be expressive and easy to be managed (a well-known issue in pervasive computing [5]), and its amount must controlled so that it does not become overwhelming (too much information implies no information).

Raw data generated by pervasive computing devices is often hard to be used. The standard approach is then to enriched it by relating and combining data items with each other so as to generate higher-level, more expressive and easy to be managed, knowledge that, in turn, is also represented in some standard and processable way [4]. As a trivial example, if a GPS device can generate information about your absolute location on earth, only by relating this data with the data of some mapping tool and by representing the results in, e.g., some standard format, you can understand in a meaningful way where you are.

At the same time, it is clear that a large amount of data that will be generated by increasingly dense pervasive computing devices can make it hard for services and users to extract the needed information in real time. The possibility of generating higher-level knowledge from raw data, although it can possibly facilitate access and understanding of data, does not necessarily solve the problem of having to deal with unmanageable amounts of information. Rather, it can be the case that the continuous generation of higher-level knowledge from raw data even exacerbates the problem; very large amount of higher-level knowledge pieces can complicate knowledge extraction and be of no use to anyone.

In this context, the contributions of this paper are to unfold the above issues and to explore a self-organized solution for adaptively controlling the generation of knowledge from raw data in order to limit the amount of existing knowledge (i.e., to avoid knowledge overflow). The need for a self-organizing approach to knowledge generation [2, 7, 9] is clearly required by the inherent decentralized nature of pervasive computing, which asks for a data management process that is adaptive and based on local information. In our approach, specifically, biologically-inspired self-organization is used to adaptively drive the process of knowledge generation based on the queries taking place in the system, i.e., the more some knowledge is queried, the more the generation of similar knowledge items is reinforced. The result, which we discuss and validate with the help of simulation, is that the number of new knowledge generated can be effectively controlled without any a priori information and without undermining the possibility for services and users to access the needed knowledge.

The remainder of this paper is organized as follows. Section 2 introduces a scenario that is used as a running example throughout the paper. Section 3 presents the W4 system, a simple and efficient framework for the representation and management of pervasive information, which we use to test our approach in a concrete setting. Section 4 provides details the knowledge overflow issue as well as the self-organizing approach to knowledge generation that we propose. We end this paper in Section 5 with a discussion of our simulation results, followed by the proposed future work, in Section 6.

2. RUNNING EXAMPLE

University campuses are excellent test-bed scenarios to large-scale pervasive systems. They are densely populated by individuals carrying a variety of pervasive devices (i.e. mobile phones, PDAs, etc.). Moreover, existing network and security infrastructures facilitate people to be always connected and able to get additional information about what is happening in their surroundings. Such information can be stored and made available for the use of context-aware services.

Consider the scenario in which Patricia, a university student, has a Wi-Fi enabled PDA that collects information about about herself, share it with other users on campus, and can also access other users' information. Please note that campus infrastructures themselves can collect data about Patricia and make it available, e.g. when Patricia use his/her badge to get into the library the system gets the information about her location.

Monday morning Patricia is walking on a campus park. While in the park, her position is continuously updated by her PDA's GPS receiver. When she enters the Computer Science lab for her classes, her position can be inferred both by the Wi-Fi signal triangulation, the building sensor network, and a RFID tagging system (depending on her/his PDA equipment). Moreover, if tools are available to analyze this information and extract higher-level knowledge, one could combine it with the university's class schedule coming from Internet sites to infer what class(es) she is attending. Or, by combining this information with information related to other students, it may be possible to infer that, e.g., some social activity is taking place sometime somewhere.

Concerning the usage of such data and knowledge, imagine that Patricia does not see her friend Tom in class. Then, she can find where he is by interacting with the pervasive system. The PDA then, by querying the system, can discover that Tom is in his room and, moreover, by inferring from a combination of other data items, the system can also suggests that he overslept. After class, Patricia can perform contextual queries to know which friends are near the Computer Science building and join them for lunch.

In general, the user exploits classical contextual services and takes advantages of the knowledge management and generation features in the system to retrieve the desired information. In fact, the system itself can continuously analyze data coming from pervasive devices in the attempt to infer and extract new knowledge, and inject it again in the system. In this way, when Patricia access the system it is more likely that she finds the information she wants.

Just thinking to the tens of thousand students that a medium campus may have, and understanding that data is also coming form pervasive sensor networks densely distributed in the campus, it is quite clear that the overall system may lead to the creation of an overwhelming amount of data and inferred knowledge (indeed, the more data available the more the knowledge that can be inferred from it), even when this inference is not useful or never requested by anyone. Therefore, while tools for knowledge generation from raw data are necessary to make the system more useful (we discuss these in Section 3), tools for handling the problem of knowledge overflow are needed too (which we present in Section 4).

3. W4 SYSTEM

In this paper we use the W4 System to illustrate our self-organizing approach for knowledge management in pervasive computing. The W4 system is a simple and suitable model for contextual data, enabling flexible general-purpose management of contextual knowledge by pervasive services. However, the proposed approach in this paper is a general one, can can be easily applied to other models of contextual data.

In the W4 model, every piece of contextual knowledge is represented as a simple, yet expressive, four-field tuple *(Who, What, Where, When)*: someone or something (*Who*) does/did some activity (*What*) in a certain place (*Where*) at a specific time (*When*). A W4 tuple represents the elementary information of a fact that occurs in the world (which we also call a piece data or knowledge atom).

Despite its simplicity, the model may handle information coming from sources as diverse as embedded devices, cameras, users, or Web 2.0 sites, and can account for adaptation to context and incomplete information (i.e., some of the four fields being unspecified). W4 tuples can be stored in suitable shared data spaces, whatever distributed and implemented. Users and services, located anywhere, can retrieve W4 tuples via a simple API, based on Linda-like pattern-matching query mechanisms [1]. In addition, the simple W4 structure supports general distributed algorithms for data aggregation and manipulation. This makes it possible to link W4 tuples with each other, to generate (higher-level) W4 tuples, and to build multiple application-specific views of data.

Indeed, we have implemented a prototype of an infrastructure for pervasive services (integrating sensor networks, RFID tags, and a friendly map-based interface), and a number of exemplary services, fully relying on the W4 model. W4 tuples can be stored in a set of distributed W4 tuple spaces, which we have implemented by customizing the LighTS system [3]. We have now also integrated features for enabling self-organized knowledge generation (as described in the following sections). More details about the services and the implemented infrastructure can be found in [4, 5].

3.1 Data Representation

Each of the four fields (*Who*, *What*, *Where*, *When*) of the W4 data model describes a different aspect of a contextual fact.

- The *Who* field associates a subject to a fact, and may represent a human (e.g., a username), an animal, or an inanimate part of the context acting as a data source (e.g., the ID of an RFID tag). The *Who* field is represented by a *type:value* pair, in the form of a string, with an associated namespace that defines the *type* of the entity that is represented. E.g., *student: Patricia*.

- The *What* field describes the activity performed by the subject. This information can either come directly from the data source (e.g., a sensor is reading a temperature value), or be inferred from other context parameters (e.g., an accelerometer on a PDA can reveal that the user is running), or it can be explicitly supplied by the user. This field is represented as a string containing a *predicate:complement* statement. For example, valid entries for the *What* field are: *attending:Computer Foundations class*, *read:temperature=23*.

- The *Where* field associates a location to the fact. In our model the location may be a physical point represented by its coordinates (longitude, latitude), a geographic region (described as a bounding box), or it can also be a logical place. In addition, context-dependent spatial expressions like *here* or *within:300m* can be used for context-aware querying.

- The *When* field associates a time or a time range to a fact. This may be an exact time/time range (e.g., *2008/ 07/19:09.00 am–2008/07/19:10.00 am*) or context-dependent expressions (e.g., *now*, *yesterday*, *Tuesday*). The field is also important for context-dependent querying.

The way it structures and organizes information makes the W4 data model general enough to represent data coming from many heterogeneous sources, but also simple enough to promote easy data management and processing (although we are perfectly aware that it cannot capture each and every aspect of context, as freshness of data, reliability, access control, etc).

3.2 Data Generation and Data Access

In the W4 model, we rely on the reasonable assumption that software drivers (or, more generally, software agents) are associated with data sources, and are in charge of creating W4 tuples and inserting them in data spaces. In the end, any data source must be somehow associated with some software agent/driver to gather and store data items, W4 agents have the additional goal of collecting all the necessary information to produce a W4 tuple which is as accurate and complete as possible. This occurs by sensing and inferring information from all the devices and sources available (e.g., RFID tags, GPS devices, Web services), and by combining them into a W4 tuple.

Since all knowledge atoms are stored in the form of W4 tuples in a shared data space (or in multiple data spaces), we took inspiration from tuple-space approaches [1] to define the following API to access W4 tuples:

```
void inject(KnowledgeAtom a);
KnowledgeAtom[] read(KnowledgeAtom a);
```

The *inject* operation is equivalent to a tuple space *out* operation: an agent accesses the shared data space to store a W4 tuple. The *read* operation is used to retrieve tuples from the data space via querying. A query is in turn represented as a W4 tuple with some unspecified or only partly specified values (i.e., a template tuple). Upon invocation, the read operation triggers a pattern-matching procedure between the template and the W4 tuples that already populate the data space. In the W4 system the pattern-matching operation works differently from the traditional tuple-space model since it exploits diverse mechanisms for the various W4 fields and it also enforce context-aware queries.

3.3 W4 Knowledge Networks

Although pattern-matching techniques proves rather flexible to retrieve context information, the key idea is to exploit the W4 structure to perform some semantic data organization, aggregation and pruning in order to extract meaningful knowledge from existing data. In this way the tuple space acts as a sort of ”live layer” turning raw *data atoms* into expressive *knowledge atoms* representing a situation from a higher-level perspective.

In particular, new information could be produced by navigating the space of W4 tuples, linking together existing tuples (to create something we may call knowledge networks), and consequently combine and aggregating existing tuples into W4 tuples. Such new knowledge could also arise from the temporal analysis of the historical context (e.g., the location where a person spends 8 hours every day could be his workplace) or from a wide analysis of the state the whole W4 repository (e.g., If nobody go to work on 2007/12/25, it could be an holiday).

The W4 knowledge networks approach is based on the consideration that a relationship between knowledge atoms can be detected by a relationship (a pattern-matching) between the information contained in the atoms fields. In particular, for the W4 model, we can identify two types of pattern matching correlations between knowledge atoms:

Same value – same field : We can link together W4 tuples belonging to the same user, about the same place, activity or time (or, more in general, those W4 tuples in which the values in the same field match according to some pattern-matching function). Matching two or more *same value – same field* relationships, we can render complex concepts related to groups of W4 tuples, e.g. *All students (same subject) who are attending a class (same activity) at the same room (same location).*

Same value – different field : We can link atoms in which the same information appears in different fields. This kind of pattern matching can be used for augmenting the expressive level of the information contained in the W4 tuples. For example, a knowledge atom having *When: 18/09/2008* can be linked with another atom like *Who: Fall Class Begin* , *When: 18/09/2008* to add semantic information to that date.

Exploiting those correlations makes it possible to find all relationships between one particular W4 tuple with all other tuples in the data space which may then be used as the basis for more elaborated inference and reasoning, even for eventually creating new W4 tuples. In fact, the links between W4 tuples usually enables the building of higher-level, more expressive tuples. For example, suppose that Patricia's PDA, at a certain time, creates the following tuple: *(student:Patricia,−,room:Computer Science Lab,01/09/2008 10:05 am)*, where − means an empty field. Algorithms in the knowledge network continuously analyze the network and find a correlation with an atom stating that a Computer Foundation class takes place in the Computer Science Lab at 01/09/2008 from 10:00-12:00 am *(class: Computer Foundation,−,room:Computer Science Lab,01/09/2008 10:00-12:00 am)*. A new tuple carrying higher level logical information may be created, stating that Patricia was attending the Computer Foundation class: *(student:Patricia, attending:Computer Foundation,room:Computer Science Lab,01/09/2008 10:05 am).*

4. SELF-ORGANIZED CONTROL OF KNOWLEDGE GENERATION

The introduced W4 model and its knowledge networks approach can produce an overwhelming amount of inferred data by continuously generating new W4 tuples that can be difficult to be managed and may never be requested by services that access the framework.

Of course, the same problem can affect any other data management system having the goal of managing pervasive data and knowledge. In order to make the W4 system (or any other system with similar goals) more realistic and efficient, the knowledge inference must be controlled so as to lead to the generation of tuples that are likely to be useful in the system.

In this section we first show that the knowledge overflow can be a problem, and then we introduce a self-organizing approach to control the knowledge generation. The argumentation is supported by some experiments performed by simulating the generation of W4 tuples in the W4 LighTS infrastructure already described at the beginning of Section 3.

4.1 The Knowledge Overflow Problem

The W4 system and most of the other systems assume a naïve approach to knowledge inference. In essence, it constantly generates knowledge from current atoms (raw data) independently of the knowledge being required at a particular point in time. Whenever a W4 tuples are linked together, the system tries to generate a new tuple expressing the higher-level concept derived from the merging of the tuples. The result could be an overflow of tuples inferred by the system which are not guaranteed to be needed in the future. In fact, with this approach, while useful tuples are indeed generated, they are only part of the entire of corpus of tuples being generated; the system generates useful and "useless" knowledge. In the worst case scenario, the knowledge inference process produces a number of tuples as specified by equation 1, which indicates that the number of atoms (tuples) at time $t+1$ is the accumulation of the number of atoms in the previous step t plus the number of new tuples that can be inferred (knowledge inference) .

$$N(a)_{t+1} = N(a)_t + \Delta N(a) \tag{1}$$

where

$$\Delta N(a) = \left[\frac{N(a)_t \times (N(a)_t - 1)}{2} \right] \tag{2}$$

Obviously, the equation 2 assumes that all current atoms can be combined among themselves; it accounts for all pairs of tuples being considered in the generation of a new tuple (inferred knowledge), which may not be always possible.

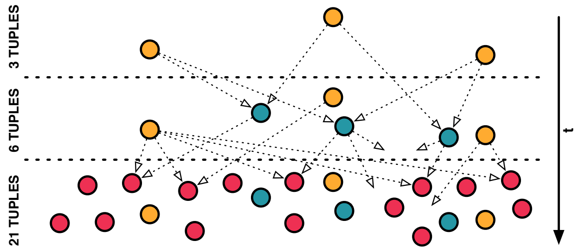

Figure 1: This figure depicts the worst-case scenario of knowledge generation. At every time iteration, all tuples participate in the process of knowledge inference. The result is an exponential growth on the number of tuples.

Figure 1 shows tuples being generated as part of a naïve inference mechanism as implement in W4 currently. As one can see the number of tuples in the system grows very rapidly. [1] This redundant generation not only is unnecessary but, if not corrected, can cause the W4 system to loose efficiency because it has to deal with a very large number of tuples.

[1]Note that for clarity, not all arrows are shown in the transition from the second to the third step.

The alternative to this, as proposed in Section 4.2 is to have a mechanism that uses self-organization to drive the inference of tuples. In essence one can assume that at every time step, not all tuples are used in the inference process. Which means that that the $\Delta N(a)$ in Equation 1 is changed to consider only some of the tuples in the space rather than all of them. The number of tuples used in each time iteration t does not have to be fixed and may depend on W4 system factors.

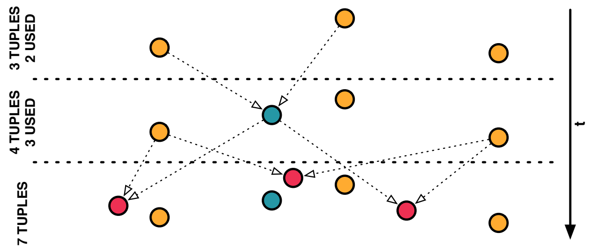

Figure 2: This depicts a case where just some of the tuples participate in the knowledge inference at every step. The controlled self-organization achieves this by having a probabilistic knowledge inference based on the queries that exist in the system (as described in Section 4.2.

In Figure 2 we see the effect of having just a few tuples being used in the generation of new knowledge. In the first step, out of 4 tuples, 2 were used to generate knowledge, and in the second step we considered 3 tuples out of the 5 that were present in the system. The obvious outcome is that significantly less tuples are present in the end (step 3). Note that we are not considering the quality of the information generated. Some quality is achieved based on the self-organized mechanism that can consider the type of knowledge that is needed in the system but for now we want to concentrate on a mechanism to avoid the unnecessary generation of too many tuples.

We have run some experiments in W4 to demonstrate that what we describe above does indeed happen in practice. We have simulated what happens in a standard day at the Engineering Campus of UNIMORE (Reggio Emilia site). The Engineering campus is supposed to be tagged with a number of RFID tags, each tag containing a W4 tuple describing a point of interest or a location at different levels of granularity such as offices, class rooms, etc. People in the campus (students, professors, staff) are supposed to carry a PDA equipped with a short-range RFID tag reader, a GPS and a WI-FI connection, as the system described in [4]. The PDA is in charge of periodically (in the simulation every 5 minutes) creating a W4 tuple describing the status of each user. The W4 tuple is then sent to a W4 remote tuple space. W4 tuples come also from agents that analyze the faculty websites and create tuples with class schedules, buildings information, departmental events such as meetings and so on. In the experiments the W4 knowledge network continuously scans the tuple space to retrieve tuples that:

- are in the campus area (based on GPS coordinates in the *where* field)
- are in the campus area (based on a logic description in the *where* field)

- are in a building (based on a logic description in the *where* field)
- refer to attending a class (based on the *where* and *when* fields)
- happens in a day of the week (based on the *when* field)

The engineering campus we simulated is made up of 3 buildings, with 10 classrooms, 8 laboratories, 20 offices. The average number of classes in one day is 15 for 125 students and 20 professors.

We have simulated a campus environment by keeping track of the tuple space size at the end of each W4 knowledge networks iteration. Figure 3 shows the cumulative value for the tuple space. During the first iterations the number of tuples increase quickly, then it continues to increase (as new tuples are continuously injected and the knowledge networks act on those tuples) but at lower rate.

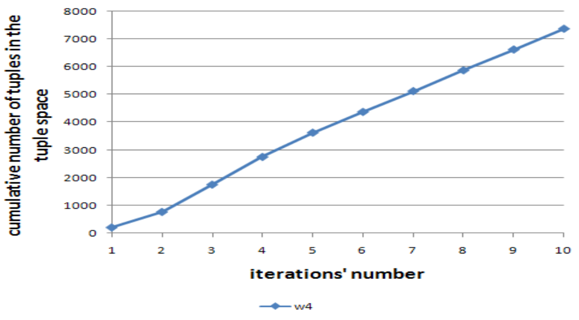

Figure 3: The knowledge overflow effect on the W4 system: cumulative values

Figure 4 shows the rate of generation of tuples for the same data depicted in Figure 3. It is clearly visible that the generation of tuples is growing fast in the beginning, and then it adjusts to a constant trend. It has to be noted that the knowledge overflow effect is clearly visible even in a small-scale scenario as the campus we simulated. In a bigger campus, i.e. a campus with more students and more infrastructures, it should be even more prominent.

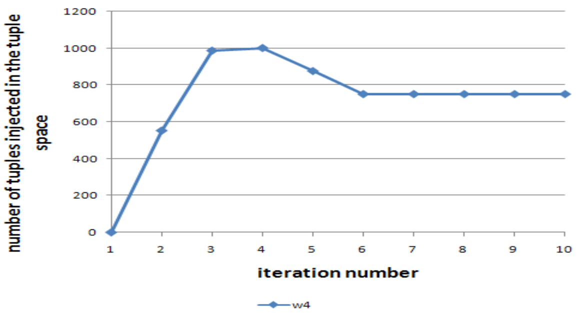

Figure 4: The knowledge overflow effect on the W4 System: the generation rate

4.2 A Self-Organizing Approach to Controlling Overflow

The simple example described in Section 4.1 shows that, if no action is taken to curtail the generation of knowledge, pervasive systems are likely to become impractical since the amount of data present at any given point can be overwhelming at least for real-time exploitation of such data by pervasive context-aware services—it is clear that big data centers can perfectly manage such amounts of data in an off-line fashion. This problem is even more pertinent if we assume that pervasive environments are made of many devices with limited computational power. Therefore one has to control the influx of data from knowledge generation as well as improve the visibility of data that is required in the system.

The question then is how to control data generation and visibility in pervasive systems. Given the sheer scale of such environments, it is impossible to have a centralized process in charge of making decisions about what should and should not be generated. The proposed approach introduced in this paper is to let the system self-organize in response to knowledge requests being made. In other words, the knowledge generation process of W4 should be driven by system activity and by user needs for certain types of knowledge.

When one talks about self-organization it has to identify the particular metaphor that is being used. There are many metaphors inspired by a diverse set of biological and physical phenomena [7]. In our case, we have adopted an approach based on ant foraging. As described in [7] this approach uses a marker-based approach where individuals stochastically explore possibilities and self-organize their activities of searching for food.

In particular, we maintain markers about the need for tuples based on queries. The system then stochastically decides to generate knowledge based on these markers. In essence this means that the self-organized process in the system will tend to infer knowledge that are needed in the system (based on current queries). Obviously, as in most self-organized approaches a second force needs to drive the self-organized behavior. The queries act as positive feedback that is used to drive knowledge generation of a particular format, but the markers used to drive the generation need to have a time-to-live otherwise a burst of queries of a certain format can forever drive the system in that direction. Accordingly, the markers are updated using a known rule proposed in ant-colony systems, as described in Equation 3, which states that the amount of a marker for a given query format $M(qf)$ is decreased by a rate ρ and increased by an amount $\Delta M(qf)$ accumulated in the last iteration of the system.

$$M(qf)_{t+1} = \rho M(qf)_t + (1-\rho)\Delta M(qf) \qquad (3)$$

Our self-organized approach to tuple generation with overflow control uses all the markers for all query types to decide which type of knowledge is *currently* needed in the system. This decision is stochastically and proportional to the amount of a particular marker in relation to all the markers in the system.

5. EXPERIMENTAL RESULTS

To evaluate the improvement introduced by our self-organized approach, we performed simulations based on the scenario introduced in Section 4.1. We used a query generator submitting non-destructive queries to the system every 250 milliseconds. For the sake of implementation we considered markers of the following kind:

```
<student:id, - , - , - >
```

where the marker is identified by the value of who field. To analyze the performance and the reactiveness of our approach, we considered two non-overlapping pools of tuples: poolA and poolB (both are made up of three tuples). The query generator works accordingly the following pattern:

- phase 1: 90% of queries are about students in the poolA, while the other 10% are uniformly distributed among students that are not in poolA.
- phase 2 (transition phase): 60% of queries are about students in the poolB, 30% of queries about students in the poolA, 10% of queries uniformly distributed among students that are not in poolB.
- phase 3: 90% of queries are about students in the poolB, while the other 10% are uniformly distributed among students that are not in poolB.

The W4 tuple space engine has been modified to keep track of the most relevant markers. This has been done introducing a system hash table that keeps the number of queries for each queried marker, and modifying the read operations to update the hash table every time a query is submitted to the system. Every time that a tuple is inferred by the system, the thread that is in charge of managing that new tuple will inject the new tuple in the system. Inferred tuples relating to highly queried markers are likely to be generated and inferred tuples relating to low queried markers are not likely to be actual generated. To better evaluate the improvement introduced by the self-organizing generation of new knowledge we considered the particular case in which the W4 knowledge networks scans the tuple space to retrieve tuples to meet one single relationship. We chosen the relationship referring to individuals attending a class. The figure that follow from here are all relative to this particular case.

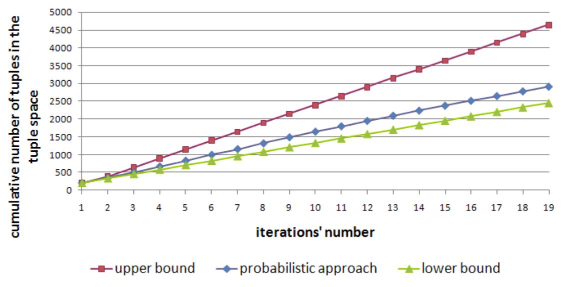

Figure 5: Cumulative number of tuples in the tuple space when the W4 knowledge network scans the systems to meet a single relationship.

Figure 5 shows the cumulative value for the tuple space and compares the value with the the upper bound (i.e. inferred tuples are always injected in the system, depicted in Figure 3) and with the lower bound (i.e. inferred tuples are never injected in the system). Figure 5 clearly shows that the amount of inferred tuples is way lower than the upper bound and it actually approximates the lower bound.

Figure 6 shows the number of tuples generated and injected by each iteration. The graph shows that after the initial peak the number of tuples grows based on the query pattern. In particular it is clear that the number of tuples grows a little at the first iterations of each query phase when there is not a highly preferred marker and all markers have a low generation rate.

The previous graphs show the situation regarding to the whole tuple space. We also analyzed the trend of tuples in poolA regarding to tuples in poolB.

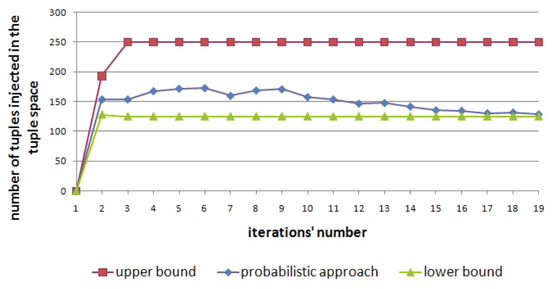

Figure 6: Number of tuples generated and injected in the iteration time when the W4 knowledge network scan the systems to meet a single relationship.

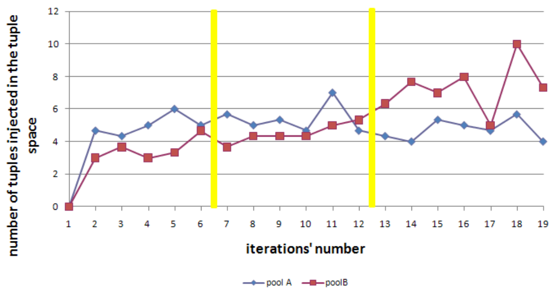

Figure 7: Number of tuples generated and injected in the iteration time. tuples in the poolA Vs tuples in the poolB. The yellow lines indicate the beginning of a new queries' phase: 90% of queries are about students in the poolA, the second phase is a transition phase and in the third phase 90% of queries are about students in the poolB. When the system has memory, its response is slow.

Figure 7 shows the number of tuples generated and injected in the iteration time for tuples in poolA and tuples in poolB. In those simulations the system has memory, i.e. the value ρ in 3 is zero. The yellow lines indicate the beginning of a new query phase, as explained at the beginning of this section. The number of tuples in poolA is greater in both the first and the second phases, while in the third phase the number of tuples in poolB increase quickly. This system has a slow response as we would like the number of tuples in poolB increasing since phase 2.

Figure 8 considers a ρ value equal to 0.4, that means that the systems forget the 40% of what happened in the past at each iteration. It clearly shows that while the line for poolA is more or less on a constant trend, the line for poolB increases during the second phase; this approach provides a quicker response, which means that the system is able to generate meaningful knowledge more efficiently.

The previous figures show the situation about the particular case of one single relationship taken into account by the W4 knowledge network. Figure 9 shows the cumulative values for the tuple space when the W4 knowledge network scan the tuple space to retrieve tuples to meet all relationships described in Section 3.3. The figure is here to confirm that the observation of the gain of the self-organized approach is true to the general case also.

6. CONCLUSIONS AND FUTURE WORK

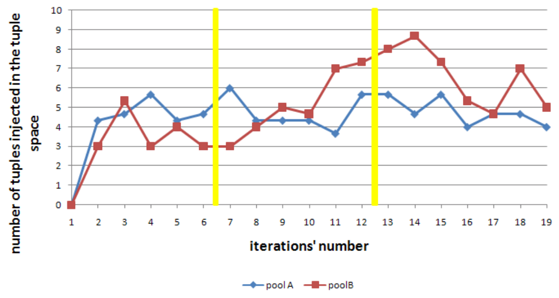

Figure 8: Number of tuples generated and injected in the iteration time. The ρ parameter improves the system's speed.

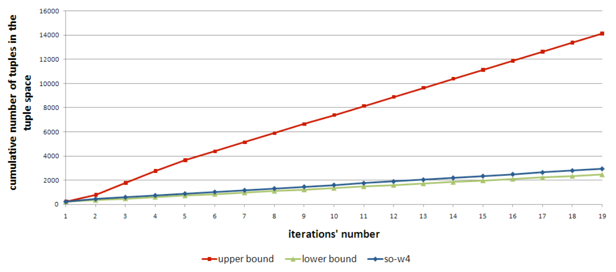

Figure 9: Cumulative number of tuples in the tuple space, when the W4 knowledge networks consider all the relationships introduced in 3.3.

In this paper we discussed the knowledge management issues that may affect pervasive computing systems, in which high-level information is combined and inferred from raw data, thus possibly leading to knowledge overflow. We have proposed a self-organized approach to control the knowledge generation process and provided some promising evaluation results for a specific pervasive framework based on the W4 contextual data model. In particular, we have shown that a self-organization approach have proven to significantly reduce the number of tuples (i.e., exactly those not very useful) that are being generated as part of the knowledge inference process. We have also argued that the need for a self-organized approach spawns from the fact that the scale of pervasive applications does not allow for centralized entities to be able to make good decisions.

However, in our system, the number of tuples present in the system is still non-decreasing except for tuples that are removed as part of explicit destructive queries. For long-running systems, this can represent a problem. Accordingly, we plan to experiment with a concept of knowledge tuple fading, as introduced in [8], to W4. There, tuples would decrease in importance and hence be less likely to be considered in the knowledge inference process. The more a tuple is required in the system, the more visible it is, which means that tuples that are not being used become less visible (they fade). We are currently working on the implementation of this concept in W4 and we are confident this will improve the systems ability to infer knowledge that is required by processes in a true self-organized fashion.

Acknowledgments

Work partially supported by the integrated project CASCADAS (IST-027807), funded under the FET Program of the European Commission.

7. REFERENCES

[1] S. Ahuja, N. Carriero, and D. Gelernter. Linda and friends. *Computer*, 19(8-9):26–34, 1986.

[2] O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, and T. Urnes. Design patterns from biology for distributed computing. *ACM Trans. Auton. Adapt. Syst.*, 1(1):26–66, 2006.

[3] D. Balzarotti, P. Costa, and G. P. Picco. The lights tuple space framework and its customization for context-aware applications. *International Journal on Web Intelligence and Agent Systems*, 50(1-2):36–50, 2007.

[4] G. Castelli, M. Mamei, and F. Zambonelli. Engineering contextual knowledge for autonomic pervasive services. *International Journal of Information and Software Technology*, 52(8-9):443–460, 2008.

[5] G. Castelli, A. Rosi, M. Mamei, and F. Zambonelli. A simple model and infrastructure for context-aware browsing of the world. *Pervasive Computing and Communications, 2007.*, pages 229–238, 19-23 March 2007.

[6] R. Jain. Eventweb: Developing a human-centered computing system. *Computer*, 41(2):42–50, 2008.

[7] M. Mamei, R. Menezes, R. Tolksdorf, and F. Zambonelli. Case studies for self-organization in computer science. *Journal of Systems Architecture*, 52(8-9):443–460, 2006.

[8] R. Menezes and A. Wood. The *fading* concept in tuple-space systems. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, pages 440–444, Dijon, France, 2006. ACM, ACM Press.

[9] H. V. D. Parunak. 'go to the ant': Engineering principles from natural agent systems. *Annals of Operations Research*, 75:69–101, 1997.

[10] M. Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–84, 1993.