WP5: “Knowledge Networks”

Deliverable D5.1: Knowledge Networks Specifications, Mechanisms, and Alpha Software Release

Status and Version:	Version 1, Draft	
Date of issue:	11.12.2006	
Distribution:	Public Deliverable	
Author(s):	Name	Partner
	Franco Zambonelli	UNIMORE
	Matthias Baumgarten	UU
	Nicola Bicocchi	UNIMORE
Checked by:		

Table of Contents

1 Introduction

1.1 Purpose and Scope

This document represents the M12 deliverable for the CASCADAS WP5 "Knowledge Networks". It includes a general introduction to knowledge networks concepts and to related work in the area, the description of the first specifications for knowledge networks, and the description of the preliminary software developed to test knowledge networks concepts and functionalities.

1.2 Document History

Version	Date	Authors	Comment
0.2	19/11/2006	Franco Zambonelli	Structure and Introductory parts
0.4	4/12/2006	Matthias Baumgarten & Nicola Bicocchi	Finalized KN specifications & software description
1	11/12/2006	Franco Zambonelli	Added application parts & integrated/harmonized parts from TI and UNIK
1.1	09/01/2006	Franco Zambonelli	Integrated all revisions and comments form partners and from BT

1.3 Document overview

The document is structured as follows. Section 2 sketches the general vision of knowledge networks and introduces some basic knowledge network definitions and concepts. Section 3 puts knowledge networks in context, by discussing relevant related works in the area. Section 4 details the preliminary specifications of knowledge networks, in terms of the structure of knowledge networks components and their relations. Section 5 details several mechanisms and examples of use of knowledge network. Section 6 describes the characteristics of the first (alpha) release of a software system for testing and experiencing with knowledge networks. Eventually, Section 7 defines a roadmap for the future activities to be performed within WP5.

2 General Vision and Basic Knowledge Networks Concepts

The capability of services to autonomously adapt to the context from which they are requested and in which they execute is necessary to achieve effective autonomic behaviour to effectively satisfy increasingly demanding users [ManZ06, CMN+06]. This however requires the technologies to capture contextual data and at the same time the ability of the system and of application services to effectively exploit this data at the best.

Much of the technology to acquire contextual information is already becoming available, and it will soon become pervasive with the increasingly frequent deployment of sensors, location systems, users and organization profiles, and run-time systems for the monitoring of computational and network resources [Est02, Phi04]. What is still in its infancy and still needs to be properly resolved, however, is the investigation of the principles and the algorithms with which this growing amount of distributed information can be properly organized, aggregated, and made more meaningful, so as to facilitate their exploitation by services [MulZ06].

In other words, we think there must be an evolution from a model of simple context-awareness, in which services are given access to isolated pieces of contextual data, to a model of "situation-awareness", in which services are given access to properly elaborated and organized information representing, in much more expressive yet still simple to be exploited ways, comprehensive knowledge related to a "situation" [DeAb01, Tum05].

This is where the idea of "knowledge networks" arises: providing models and tools to analyze and organize contextual information into sorts of structured collections of related knowledge items, so as to support application and services in reaching effective of adaptability and autonomicity.

2.1 Basic Definitions

To unambiguously frame all the concepts and ideas we will present, it is necessary to provide a few basic definitions for the key terms adopted.

Context: In general terms, the context defines the "surrounding and interrelated conditions in which something exists" (Mirriam-Webster Dictionary). In CASCADAS, the context identifies the operational environment in which a service situates, which could include network, application, social, and physical context (Cfr. Knowledge).

Contextual Information: Information related to some actual characteristics of the operational environment, i.e., to some facts occurring in it.

Context-awareness: The capability of software (i.e., as far as CASCADAS is concerned, of services) of being aware of the context in which they are invoked and/or executed, and of adapting their behaviour accordingly.

Concept of Interest. A computational model of any real world object or event (there included services and processes).

Ontology: a formal specification detailing how to express concepts of interest in a specific area. In CASCADAS, a shared ontology is expected to be defined for ACE's so as to enable them to properly represent in a semantic and inter-operable way all needed contextual information.

Knowledge: Contextual information as it can be made available to some actors (i.e., ACE's) to make them aware of some facts and reason about them. In CASCADAS, we account for: network knowledge, representing facts about the current configuration of the physical network and of the related devices; application (or ACE-level) knowledge, representing fact about the current status of (some) ACE's; social knowledge, representing facts about the human actors currently exploiting the network and its ACE-based services, and the social context in which they are doing so; physical knowledge, representing facts about the physical world. We emphasize that the difference between Contextual Information and Knowledge is really subtle, and mostly related to the observation viewpoint: the context generates contextual information which then becomes something that the agent knows, i.e., knowledge.

Knowledge Atom: A knowledge atom is a record-based data structure, expressing a single specific concept of interest, and represented according to a specific ontology. For example, one can imagine the information related to the current physical location of a person can be a knowledge atom reporting the name of that person, its location in terms of latitude and longitude, and possibly some information related to the activities currently undertaken by that person. The specifications of the identified structure for knowledge atoms follows of this document.

Situation. In general terms, a situation defines a "relative position or combination of circumstances at a certain moment" (Mirriam-Webster Dictionary). Accordingly, in CASCADAS, a situation is considered as "something that is happening in the context" and, for generalization, also something that "is likely to occur at a certain moment in the future".

Situation-awareness. In general terms, situation-awareness relates to the capability of being aware and of adapting behaviour to situations other than to context (Cfr. Context-awareness). While components and services (i.e., ACE's) are situated in a context and can perceive contextual information in the form of knowledge atoms to become context-aware, perceiving situations (present and future) and becoming situation-aware implies a higher degree of understanding. In particular, it requires properly acquiring all the needed knowledge about "combinations of circumstances".

Knowledge Network: A network of knowledge is an ontology-based structured collection of knowledge atoms, describing specific situations, and built in order to facilitate ACE's in acquiring high degrees of situation-awareness in an efficient way. This is not to be confused with "network knowledge", intended as the information available about the status of a network.

Knowledge Container: As it will appear clearer in the following of this document, the structuring of knowledge atoms in networks may also imply the need to create higher-level structures aggregating existing knowledge atoms into a component which, besides being a knowledge atom in itself, aggregate a set of related knowledge atoms into a composite. Besides the basic definitions, the question arises in CASCADAS of what actually implies structuring knowledge in networks, and what types of architecture can properly support knowledge networks. The next section will try to somewhat identify some preliminary directions.

2.2 An Abstract Architectural Perspective

Once we have absorbed the general idea of "knowledge networks", the question then arises of how such networks could actually look, and what an abstract reference architecture for knowledge networks could be.

Obviously, the construction of a single knowledge network capable of mirroring the universal situational knowledge is illusionary. On the one hand, when considering that even relatively small network scenarios can (due to the availability of several sensors and devices) generate enormous amount of knowledge, it is necessary that knowledge network can provide – other than for correlating knowledge – for properly pruning it and making it manageable. On the other hand, different kinds of services may have different needs in terms of type of knowledge required and in terms of the relations that must be outlined on this knowledge.

Accordingly, one should necessarily consider the possibility of a multiplicity of knowledge networks to co-exist within a globally accessible knowledge space where each network is limited by clearly defined knowledge boundaries in order to serve application-specific and/or service-specific goals. Although the context is the same for all situations (and thus the basic contextual information is the same) the way this has to be perceived and elaborated by ACE's in terms of properly organized knowledge may depend on the specific type of service one has to enforce. In other word, the context may be in need to be perceived by ACE's as a variety of situations, and one should thus consider that several "dimensions" according to which knowledge atoms can be networked with each other exists.

Although it is generally impossible to identify all possible dimensions around which one can think at organizing knowledge, a few of them are likely to be recurrent and exploited in several applications.

First, we have a purely semantic dimension, in which knowledge atoms related to a situation network with each other according to the relations institutionalized in (or inferred from) some shared ontology. This can be the case of knowledge facilitating and supporting spontaneous interoperability in pervasive computing and service-oriented computing [HuhS05].

Second, we may have a spatial dimension, in which knowledge atoms related to a local fact network to knowledge atoms at different location (or distribute/replicate themselves in different locations). This can be of use to express some distributed situation (as in the case of computational fields or pheromones), in which spatiality actually refers to physical spatiality, and which can be of great use for pervasive computing applications. Also, we could conceive any class of spatially distributed P2P structures to distribute knowledge across a network and to facilitate access to knowledge (as in the case of networks of knowledge brokers) [Rat01, AndS04].

Third, we may have a temporal dimension, in which knowledge atoms express facts occurred (or already to occur) at different times. This can be the case of elaborating knowledge for predictive purposes: starting from the knowledge available about the situation at current time, analyze and extract new knowledge in the form of a knowledge network expressing the most likely future situation.

Figure 1 tries to somehow summarize these considerations into a sort of conceptual reference architecture. Figure 2 tries to exemplify the concept via an example in the area of pervasive computing, where a situation as simple as that related to the position ("location") of a user can tolerate both a semantic (vertical) knowledge network and a spatial (horizontal) knowledge network.

In addition to that, we may also have any number of application-specific dimensions on which to rely to network knowledge atoms in variously shaped knowledge networks serving different purposes, and possibly overlapping with each other.

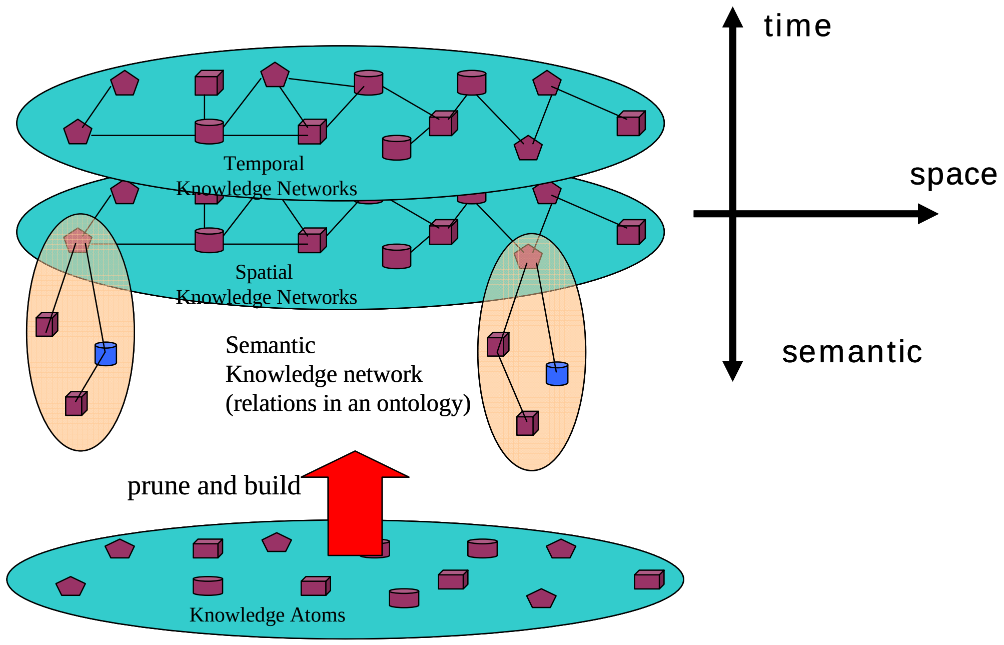

time
space
semantic
Temporal Knowledge Networks
Spatial Knowledge Networks
Semantic Knowledge network (relations in an ontology)
prune and build
Knowledge Atoms

Figure 1: A Conceptual architecture for knowledge networks: knowledge atoms expressing contextual fact can be processed and elaborated to produce different knowledge networks according to different conceptual dimensions. Such a reference architecture can consider the presence of multiple knowledge networks for each conceptual dimensions, each serving application specific purposes.

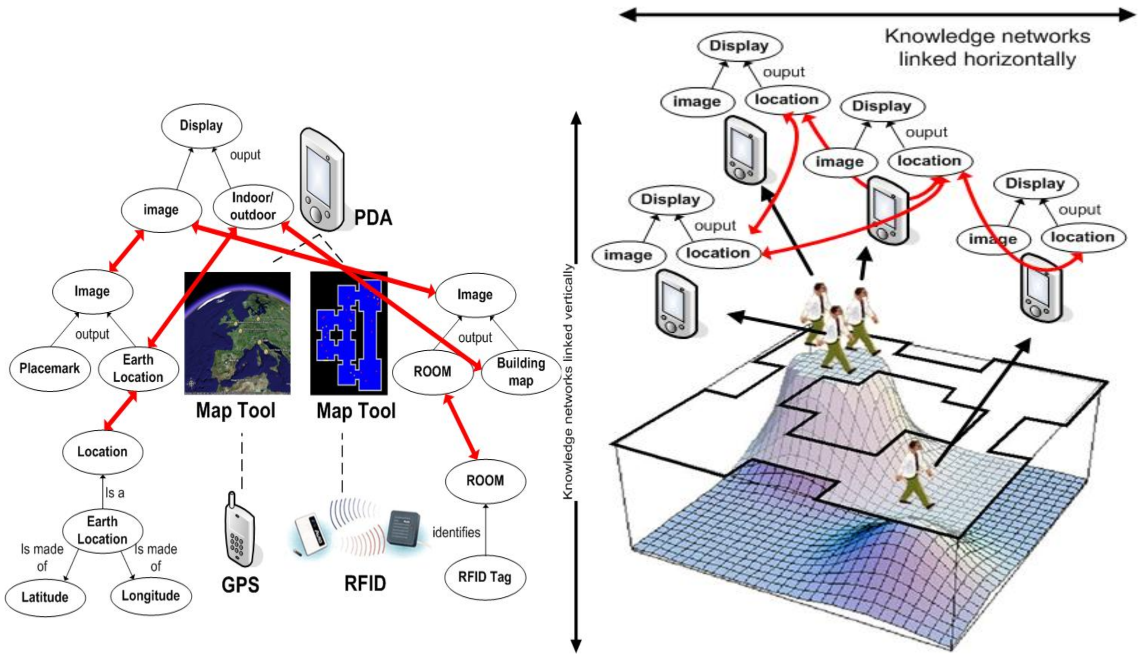


Figure 2: Vertical (semantic) vs. Horizontal (spatial dimension) in a simple example related to the "location" situation. On the left, we can see how an ontology for the concept and the mechanisms of location could be. On the right, we could see how the location per se can lead to a spatially distributed structure expressing where specific persons are (as a sort of overlay computational field – see also Section 5).

2.3 Knowledge Networks and Autonomic ComponentWare

The above introduced abstract reference architecture does not say anything about how knowledge networks should actually be implemented. Indeed, the concepts can and should be as general and implementation-independent as possible. However, in the context of the CASCADAS project and in respect of the general CASCADAS principle of "autonomic componentware" [ManZ06], knowledge networks can and should be provided as sorts of services to be realized by means of ACE's.

One the one hand, one could have dedicated ACE's in charge of storing knowledge and, by interacting and aggregating with each other, in charge of building knowledge networks in the form of networks of ACE's. In other words, there will be special-purpose ACE's acting as knowledge atoms and knowledge containers. These ACE's will be part of a sort of "middle-level", making available knowledge networks as if it were a "middleware" service make available to other (application-level) ACE's. For the sake of simplicity, and to enforce a better separation of concern, activities have focused till now on such a perspective.

On the other hand, one could also think at avoiding any distinction between ACE's, and at enabling any kind of ACE's to contribute to the forming of application-specific knowledge networks and at storing pieces of such knowledge networks, depending on specific attributed and roles (i.e., with reference to the ACE model, by having ACE's expose their self models and states as knowledge). In later stages of the CASCADAS project we will account for this perspective.

In any case, nothing prevents making the two perspective co-exists: special-purpose middle-level ACE's and application-level ACE's could cooperate together for the building of application-specific knowledge networks, each making available its own capability and knowledge. Possibly, such a perspective is the most flexible one, and also avoid introducing strict layerings in the overall architecture, with the consequent reduction in flexibility. In any case, the presence of primitive-level ACE's to abstract the presence of information sources and to make available atomic items of contextual information (i.e., knowledge atoms), seems unavoidable.

Whenever an ACE's is devoted to store knowledge and to interact with other ACE's for the forming of knowledge networks, such ACE will have to include the necessary capability to participate in knowledge management activities. Also, it will have to provide some standard interface to interact with other ACE's and to provide access to knowledge and updating of knowledge. It is expected that relying on ACE's to implement knowledge networks by ACE's will facilitate a flexible and adaptive building of complex knowledge networks structure, and an ease composition of knowledge networks.

We forward to the WP1 State of the Art report for a detailed analysis of component models, and to section 4 of this document for an analysis of how the current ACE model is suited for the implementation of knowledge atoms and knowledge containers.

2.4 Self-similarity and Semantic Self-organization

CASCADAS is centered around four key scientific principles: situation-awareness, autonomic componentware, self-similarity, and semantic self-organization. While the principle of situation-awareness is at the very core of knowledge networks, the previous sub-section has also outlined the relations with the principle of autonomic componentware.

As far as the principle of self-similarity is concerned, we consider that knowledge networks should be made accessible by services at different levels of observations. In other words, and depending on the application needs, services (i.e., ACE's implementing them) should be allowed to access both very detailed information about facts occurring in a context, as well as more coarse-grained aggregated information, as if they were "observing" the system from farther. Such possibility requires that whether a service accesses individual knowledge atoms directly connected with the information sources or, instead, aggregated knowledge contained in some "knowledge container", they can adopt the same mechanisms.

It is worth emphasizing that the term "self-similarity have been recently over-exploited in the context of social networks and technological networks (e.g., the Internet and the Web), and have been mostly related to the idea "scale-free" topological structures [AlbB02, Doy05], characterized by the small-world phenomenon [Wat98]. We consider that the above properties could be very important for representing evolving distributed knowledge in a robust way, and for enabling a scalable way with which to structure and compose knowledge. Thus, it will be interesting to explore how to structure knowledge networks into scale-free structures, so as to reflect the structure of the social and technological networks they support and to support robust adaptive evolution. Moreover, the study of such network structure could be of use to support scalability of network structure and, possibly even more important, to better support scale-free composability and self-similar multi-level perception of knowledge at different scales of observation. However, our concept of self-similarity does not reduce to the topological structure of knowledge network but is mostly concerned with the possibility of enforcing different level of observations, independently of whether this is enforced via scale-free topologies or via hierarchical aggregation of knowledge.

As far as semantic self-organization is concerned, it has been observed that global self-organizing and self-adapting behaviour can be made emerge in systems of a large number of lightweight agents that indirectly interact via the mediation of an environment [Par97, BonDT99, Bab05, HWBM02]. Agents, by depositing and by sensing "pheromones" [ParBS04] or fields [MamZL04] in an environment, and by having the environment properly diffuse pheromones according to specific laws, can – to most extent unconsciously – self-organize their global activities into robust and adaptive patterns.

Knowledge networks could potentially act as a sort of computational environment via which indirect, stigmergic interactions, may take place to promote self-organization and self-adaptation of activities. Still, this requires leveraging the traditional concept of stigmergy into a concept of cognitive stigmergy. Self-organizing and self-adaptive coordinated activities at both the network and the application level should be enforced not simply by reacting to a local concentration of meaningless pheromones. Rather, they should be driven by the actual meaning of the knowledge represented within knowledge networks. Clearly, to preserve the advantages of swarm intelligence approaches, this should occur without requiring ants to become heavyweight agents, and a proper trade-off between the purely reactive behaviours promoted by traditional stigmergy and the purely cognitive behaviour promoted by artificial intelligence approaches have to be found. However, as far as we know, this is a largely unexplored research area, and only a few "position papers" exists claiming the need for such kinds of semantic self-organization models [Tum05, Zam06].

All of this said, and beside the clear need to stay up-to-date with the continuous scientific advances in the area of complex networks [AlbB02] and self-organization [Bab05], researches in WP5 have firstly to pay a careful attention at the most pragmatic issues related to: gathering and representing knowledge, building knowledge networks via proper mechanisms, and identifying how to exploit knowledge networks in applications and services. These issues, which contribute to the definition of the knowledge network specifications, are analyzed in the following sections.

3 Grounding Related Work

We shortly report here an overview of the basic models and technologies that somehow relate to our concept of knowledge networks, which have already played some role in the earlier knowledge network specifications, and/or will play some role in future researches in knowledge networks. This is of relevance here to have the reader get a more general clue of our general vision on knowledge networks.

A more extended overview of the state of the art in the area, detailing the characteristics of several systems that are here only mentioned, can be found in the "M4 State of the Art Report" of WP5.

3.1 Gathering and Representing Knowledge

A great amount of current researches in context-aware systems tend to focus on the context of users, based on contextual information such as temperature, humidity, light intensity, spatial and temporal location that can be provided by available sensors and algorithms, to adapt services to the current situations in which users exploit services. In CASCADAS, we require a much broader notion of context. In fact, the context required for autonomic and robust behaviour of services has also to strictly relate to the status of computer systems and of the exploited networks. Also, the context should include information related to the current status surrounding/interacting ACE's. In addition, context may refer to the social context from which services are exploited by users. In any case, as far as the issues of gathering and representing such information are concerned, we argue that there is no big difference for algorithms regarding the different context information. In fact, provided that sources for contextual information exist (e.g., sensors, user profiles, etc.), it makes usually no difference if context information that is represented by a number describes the current temperature, or the current CPU load. Indeed, as described in [BaDR04], contextual information of any kind can always be thought of as being provided by "sensor abstractions", which may include physical sensors (e.g., sensor networks), virtual sensors (e.g., producing information by browsing existing digital information) or logical sensors (capable of somewhat merging and synthesizing information from a variety of sensors). In our knowledge networks researches, the idea that all kind of contextual information can be represented as a "knowledge atom" is – in the end – a specific instantiation of the concept of sensor abstraction.

The issue of how information is gathered from its actual source and made available to application/services is also deeply analyzed in the literature [Win01, Chen04]. One can consider that: applications access sensors directly without any mediation [HPSH00, LaFI94, WHFG92]; or by exploiting the APIs of a middleware infrastructure [NSNT96, Dey00, DeAb01]; or by accessing some sort of "context server" in the form of a network service [HoLa01, Chen04, Peri02] or context blackboard [Cab03]. Because CASCADAS adopts the unifying ACE model for implementing all types of tools, including services to access and organize contextual information, we think that a solution that provides direct access by application service ACE's to knowledge in the form of direct access to a virtual sensor (as defined by the general concept of knowledge atom) is the one to be preferred as a starting point. In any case, this does not exclude the possibility of organizing sensor ACE's into proper network structures, so as promote context brokering and enable them to act as a sort of network service for knowledge provisioning. Nor does it exclude the possibility of defining (when needed) specific ACE's that can act as sorts of blackboard to mediate access to other sensor ACE's. However, we think CASCADAS should not rely on pre-defined complex and heavy-weight middleware and network services to enable gathering of contextual information.

The issue of representing contextual information, i.e., identifying suitable models to facilitate the understanding of such information by software and services is also extensively analyzed [BaDR04, BrLe04]. Apart from graphical representations [HelR03, ShBe05] of limited interest for CASCADAS, context information can be represented in terms of simple key-value pairs, via a mark-up language such as XML [StPo04], in object-oriented terms or in logic one [HSPL02]. For the CASCADAS project, key-value models may appear too simplistic to represent significant contextual information (although these may be appropriate for resource-constrained devices and small sensors). Mark-up languages such as XML may be very effective to provide open and easy to process representations, and we indeed commit to such a representation. However, one has also to consider that some inspiration from object oriented models will be taken, which is in accord with the goal of representing and providing access to knowledge via ACE's.

3.2 Mechanisms for Networking Knowledge

The high level goal of knowledge networks can be summarized as the provision of a vehicle capable of creating, storing, propagating and discovering information in a light-weight, scale free and multi-view environment (which makes knowledge networks notably differ from the "knowledge plane" approach [Cla03] and alike, which also charge the knowledge level with the duty of understanding knowledge and taking actions to ensure the proper functioning of the application levels).

The structural requirements for such a vehicle can be broken down into two main building blocks. Firstly, an autonomous knowledge entity is required capable to encapsulate and transport knowledge independent of the environment. Secondly, designated and dynamically maintainable relations have to be overlaid upon those entities connecting them to a purpose-built network based structure. While the former component is practically provided through the concept of ACE's as envisioned by the CASCADAS project, the latter concept requires the provision of advanced network based *ontologies* and flexible *overlay* structures that allow for *ad hoc* reconfiguration of the overall knowledge network and for the construction of purpose-built views of any sub-part(s) thereof.

An ontology is generally defined as an "explicit specification of a conceptualization" [Gruber, Usc96, BreO04], and as such is capable of representing relevant objects, concepts or other entities of interest and all their relation in an explicit and formal manner [GeNi87]. Within this definition, ontologies are not simply a way of representing concepts, but are indeed a way to put concepts related to contextual information in relation with each other and facilitate access to them. That is, ontologies can be considered, to most extents, knowledge networks (with reference to the reference architecture of Figure 1, ontologies provide for organizing knowledge around the semantic dimension). Clearly, ontologies should (and do) provide a general independent of programming language, underlying operating system or middleware. Other knowledge 'consumers' in the network must be able to access and use the ontological formalisms developed. Accessing information stored in a network of distributed contextual knowledge requires the specification of information locators, e.g., in the form of an addressing scheme as well as request routing procedures. In the literature, a very large amount of ontology proposals can be found. These include CoOL [Str03a, Str03b], CONON [Wan04], CoBrA [Che03], each of which is specialized to a specific area of applications. Of great interest are proposals such as CYC [LeGu90], OWL [OWL], ConceptNet [LiuS06], which also have the advantage of being extensible. Thus, new concepts and new relations can be defined by sub-classing from existing ones and to make the ontology better suit specific application areas. In CASCADAS, we have started our research in knowledge networks by assuming the presence of a basic shared ontology for knowledge network concepts, i.e., we start by simply assuming that specific terms have a well assessed meaning. While this is sufficient to identify the basic knowledge network specifications, later on in the project it will be necessary to extend the approach towards the exploitation of full-fledged, flexible, and extensible ontologies.

Overlay networks, within a general networked infrastructure such as the internet, provide an abstract view on such networked environment, tailored to specific needs, and without the necessity to know or care about underlying real network infrastructures [Cast02, BBM+02, CSW+00, GaSt04, GFC00, CMK+99, LCH+05, BCA01, Keah02, Korp01, Ratn02]. In overlay architectures, a set of nodes (servers, services, end-user equipment etc.) and virtual links, not directly related to an underlying topology, are involved in specific applications. The overlay traffic traverses through the overlay nodes and virtual links. Therefore an overlay network can be seen to act as a specialized middle layer between an application and an underlying topology of entities. The general advantage of overlays is that they can be customized for a single service or a group of services, thus creating a variety of overlays that allow for hierarchical structures. This is perfectly in line with our view (as sketched in the reference architecture) of building on a variety of different knowledge networks that, starting from raw contextual-information, can provide different views to applications/services. Among several studies in the area of overlays, peer-to-peer overlay networks for content-sharing, such as CAN [ABA+03], Chord [Stoic01], Freenet [CSW+00], Gnutella [Gnut01, LRS02, KlMa02, Stoke03], Pastry [RoDr01], SkipNet [GFC00], SWAN [BH02], which have received a great attention in the past few years [MTT03], are particularly interesting because they are capable of self-organizing their structures and of self-healing. These features are highly relevant to our efforts of making knowledge networks self-organizing and self-adaptive, and this document indeed reports early experiences in that directions. Additional inspirations might come from recent work on self-adaptive peer-to-peer structures [HalA06, CKG04], semantic peer-to-peer overlay networks [LWS+03, GaCr03], and from general frameworks for the creation of overlays (such as Opus [BKR+02] and JXTA [SUN05].

The two concepts of ontologies and overlays as outlined above provide the two main building blocks of knowledge networks in a way that: (a) individual knowledge components are linked together through high-level ontologies, thus providing structured knowledge at different levels of granularity; and (b) the concept of overlay networks is exploited to provide highly dynamic, purpose-built and *ad hoc* constructed views of any (sub-)part thereof, independent of the circumstances of where the knowledge resides or through whatever means it has to be accessed. In tandem, both concepts provide a vehicle to represent, maintain and provide knowledge yet neither provide mechanisms that are capable to efficiently identify or track knowledge components in a global, highly-distributed environment.

3.3 Models and Mechanisms for Component Coordination

In CASCADAS, the idea of a knowledge repository to store contextual and situational information is provided through the concept of knowledge networks. However, considering the fact that knowledge seldom resides where it is consumed and that knowledge is normally too complex to be represented as a single structural element, the effective collection of distinct parts of individual knowledge (i.e., knowledge atoms) and provision thereof to services is an important aspect. The mechanisms for relating knowledge atoms via ontologies and overlay network have been identified. Still, proper models and mechanism for rapid correlation and distribution of knowledge components in a network environment and upon which to rely for the actual construction of ontological relations and overlay structures have to be identified. In this context, biologically-inspired and socially-inspired approaches may be – and indeed have been – of use.

First, coordination among individual knowledge entities of a more globally structured knowledge network could occur e.g., via stigmergic mechanisms [Holl96, Par97, PBS04]. As already outlined in Section 2, the presence of a distributed network of knowledge, to be accessed for sensing and effecting by both network and application level components, can act as the computational environment to enforce stigmergic self-organization by ACE's via knowledge networks. However stigmergy can also be considered and exploited as a mechanism to actually build and maintain knowledge networks: knowledge atoms themselves could sense and act on knowledge and self-organize in a robust and adaptive way. In contrast to overlay networks in peer-to-peer environments, knowledge networks should not simply transport data and messages but also support their own nodes to adapt. Not by means of heavy-weight autonomous agents (as in the knowledge plan approach [Cla03]) but rather via mechanisms of simple knowledge-mediated reactive adaptation.

Clearly, stigmergy can be considered as an instance of the more general perspective of swarm intelligence [BoTh00], i.e., the property of a system whereby the collective behaviors of (unsophisticated) agents interacting locally with their environment cause coherent functional global patterns to emerge. We feel that several examples of swarm intelligence can provide useful inspiration for identifying mechanisms for building and maintaining knowledge networks. An example of swarm intelligent ant-based behavior of interest for the building of knowledge network is that of collective sorting, which can be used as in [PWC+05] to cluster large amount of documents (i.e., in our case, large amounts of distributed knowledge atoms. As another example, the concept of a pulse monitor, explore in [SGMH04] to reproduce the fault tolerant heartbeat monitor mechanism could be exploited to realize a "heartbeat" mechanism into any knowledge atoms, such that at given intervals relevant health based information are sent to all other entities and / or to a central monitoring facility. The specific example of swarm intelligence that we have already actually experienced to aggregate distributed sensors (each abstracted as a knowledge atom) will be described in the following.

Finally, the area of game theory also offers a number of concepts that promise to be potentially useful in knowledge networks [PlatoGT, Gint00], in particular for the adaptive and robust coordination between knowledge components. However, we still have not investigated this issue in detail.

3.4 Predictive Knowledge Networks

Predictive knowledge networks may be seen as the next step in the evolution of the knowledge network approach envisioned by CASCADAS. The ultimate goal can be summarized as the provision of accurate, real-time predictions of any kind about individual objects, entities, relations or higher concepts that are embraced by the knowledge system.

In general terms, predicting situations may take place by analyzing existing knowledge, extracting relevant patterns of knowledge, reasoning about it, and learn from it. In CASCADAS, mechanisms for the extraction of relevant data patterns from available knowledge data (i.e., knowledge atoms) are of particular interest, in that such patterns can be used as the basic ground upon which to rely for the prediction of short-term as well as long-term behavioral patterns. Types of patterns that can be identified include: (a) associative-patterns, i.e., associations which are capable of representing relationships among objects of a set-orientated structure, where the order of objects is irrelevant. [AIS93, AgSi94, BMU+97, PCY97, Toi96, Zaki00, HPY00, HaZa03]; (b) sequential patterns, which are similar to associative patterns but incorporate the additional dimension of time [EO98], where the order of items is relevant and cannot be ignored. In sequential patterns, the discovery of sequences can be thought of as the discovery of associations' over temporal data [Zaki01, AgSi95] and which can therefore be useful to predict future events based on past events, as in [AgSi95, MCP98, HPM+00, HPY00, PHM+01, Zaki01]. As from the abstract reference architecture of Figure 1, the identification of such patterns among knowledge components implies building a knowledge networks along the temporal aspect.

Once knowledge has been discovered it needs to be represented in a flexible and efficient way in order to utilize it for other purposes. Accordingly to our choice of adopting XML for the representation of knowledge, we consider it very promising to investigate the adoption of PMML (Predictive Modelling Mark-up Language) an XML-based standard for the representation of predictive data mining models. PMML provides a machine-understandable standardized representation that is adhered to by all the major data mining vendors, comprising different standards that maintain high semantic integrity and coherence for the data and knowledge that is derived through designated knowledge discovery algorithms [GBR+99]. The current knowledge networks specifications, presented in the following, have also been conceived to easily support the future integration of predictive technologies.

4 Building Knowledge Networks: Specifications

In this section, we present a more operational architecture for knowledge networks, and detail the early structural specifications we have defined for knowledge network components, that is: knowledge atoms, knowledge containers, and their possible network organization. The building blocks required to construct a knowledge network can be broken down into two main components. Firstly an autonomous knowledge component is required capable to encapsulate and transport knowledge independent of the environment. Some aspects of this component will be designed and implemented within WP5, whereas others such as the transport of knowledge or a dedicated communication interface are more relevant to the concept of ACE's as envisioned through WP1 of the project. Secondly, designated and dynamically maintainable relations have to be overlaid upon those entities connecting them to a purpose built network based structure. This requires the provision of advanced network based ontologies and flexible overlay structures that allow for ad-hoc reconfiguration of the overall knowledge network and for the construction of purpose build views of any sub-part(s) thereof.

4.1 A Note on the Process of Identifying Specifications

Before going into the details of the operational architecture and of the specifications, it is worth spending a few words about the process by which we arrived at them. In fact, the description of this process may tell a lot about the rationale of the outcome and its generality.

We started our activities in WP5 with a shared perspective on the abstract reference architecture (Figure 1). Then, we analyzed the possibility of (i) using as a building block a single class of atomic elements (i.e., knowledge atoms) to structure knowledge networks and then (ii) identifying a simple set of basic mechanisms and algorithms (e.g., by exploiting the various models and mechanisms analyzed in the previous section) via which to build any type of knowledge networks (i.e., along any of the semantic, spatial, and temporal dimension, and suited for any class of application scenarios), with proper behavioural intelligence (i.e., autonomic capabilities)..

With regard to the former point, what we found out is that the need to flexibly consider a large amount of diverse models around which to network knowledge atoms together (there included the need to aggregate knowledge atoms into high-level structures and produce new atoms to compactly represent the aggregated knowledge of multiple atoms) could hardly be accommodated by exploiting a single class of element. From the viewpoint of ACE's, nothing prevents at perceiving and accessing a knowledge network as if it were composed of a single class of elements. However, as far as designing and developing knowledge components is concerned, it is much more useful to clearly distinguish between knowledge components that are indeed atomic (i.e., contain a single logical unity of knowledge) and components that, instead, provide for networking components with each other and for containing aggregated knowledge items. Indeed, the specifications we report here reflects this perspective and consider two classes of knowledge components, namely knowledge atoms and knowledge containers, both of which exposing the same interface but internally structured in a different way.

With regard to the latter point, identifying a limited number of mechanisms turned out to be impossible, due to the vast amount of diverse needs that users and services in different scenarios exhibit when having to become "situation-aware". Accordingly, we decided to adopt a radically different approach, based on a "hand-on" analysis of the actual needs of applications. In particular, we focused on developing (either with "pencil and papers" or with some simulation experiment) a set of application examples with extensive needs of situation-awareness, and at devising for them the more appropriate mechanisms for knowledge provisioning and, therefore, for the building of application-specific knowledge networks. Such an exercise, without having the ambition of being exhaustive, has definitely provided us with good insights on several typical mechanisms that might be used in knowledge networks and also resulted in useful feedbacks for refining knowledge network specifications.

Overall, the process resulted in a sort of "co-evolution" of structural knowledge network specification and identification of behavioural mechanisms for knowledge networks. The structural knowledge network specifications here reported have been conceived to accommodate the needs of several diverse application scenarios and, vice versa, mechanisms and application needs have been used to verified the suitability of knowledge network specification as they were being developed.

4.2 Operational Architecture for Knowledge Networks

The abstract reference architecture of Figure 1 considers the presence of some sorts of knowledge atoms (i.e., abstract sensors) and the possibility of operating on these to produce specific knowledge organizations around several possible dimensions.

From a more operational perspective, knowledge networks can be organized around a knowledge provisioning pyramid, as depicted in Figure 3: A knowledge network has to connect to some sort of data layer that exist, from a knowledge provisioning point of view, below a knowledge network. On the other hand a dedicated knowledge request layer is required to create temporal views of individual parts of knowledge without changing any parts of the knowledge network itself. This is necessary to provide request based and ad-hoc created knowledge structures to knowledge "requesters" which are at the top of the knowledge provisioning pyramid. While the former concept requires the implementation of intelligent methods capable to access a multitude of factual and virtual based data sources e.g. sensors, repositories, smart environments, etc., the latter requires advanced knowledge search as well as knowledge matching mechanism that, ideally, are embedded within the structure of the knowledge network itself. Finally, an organisational layer is required that actually represents the core of a knowledge network in which "all" knowledge registered through the data layer is pre-processed and organised to be served, via the request layer, to individual services and applications. For this layer dedicated and sophisticated knowledge management facilities have to be developed that allow flexible aggregation of knowledge, advanced quality management as well as sophisticated access control.

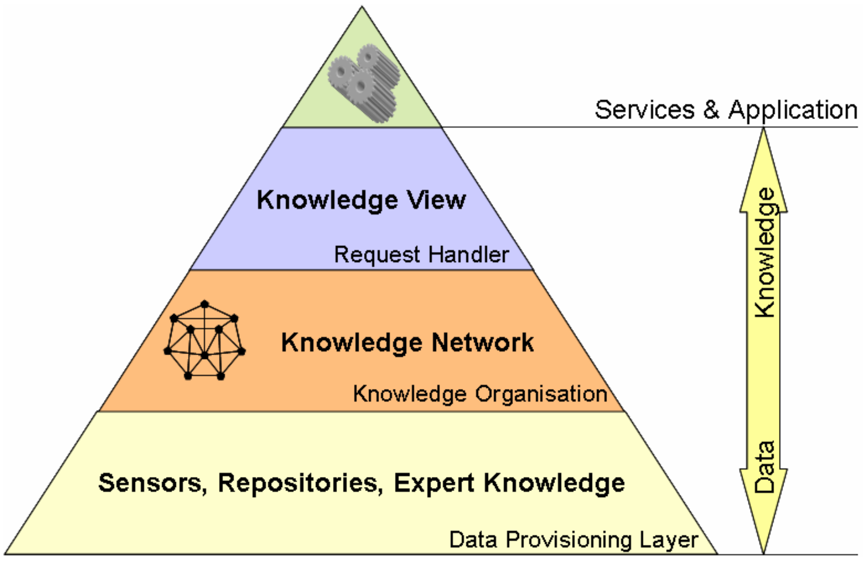


Figure 3: Knowledge Provisioning Pyramid.

The design and development of individual components of knowledge networks and their interactions with each other will be based on the knowledge provisioning pyramid depicted below where the three layers relevant for knowledge networks can be summarized as follows:

- **Data Provisioning Layer**

  The data provisioning layer represents the actual data layer where information resides or is collected from. For instance, this layer could include all the information produced by the sensor of a sensor network devoted to measure information. However, it could also include data coming form network monitoring tools, from users profiling tools, knowledge repositories or other kind of data sources. In order to access such information sources and to introduce the knowledge they provide to the next layer a dedicated component will be developed which provides a generic interface that can be extended to accommodate for different types of information sources. This component is referred to as a knowledge atom and is discussed and specified in Section 4.4.

- **Knowledge Organization Layer**

  The organisation layer could be seen as a central yet distributed information sink which contains all the knowledge generated by the lower data level in a properly represented form. Note that at this level we speak of knowledge rather than data. That is because of the fact that the information from the data layer is, at this level, properly represented and generically accessible and as such has a higher value. Once data are introduced, via the concept of knowledge atoms, into the "space" of the knowledge network, the goal is to organise them based on different characteristics, such as time, space, purpose, semantic etc. For that another dedicated component will be developed, namely a knowledge container as discussed in Section 4.4. Simplified, the rational of a knowledge container or KC is to enable specialised aggregation of knowledge sources stemming from either knowledge atoms and / or knowledge containers themselves.

- **Knowledge Request Layer**

This layer implements a dedicated knowledge view concept and is devoted to host temporal views of individual parts of the knowledge network. Basically this layer forms the bridge between individual services and applications that may utilise the knowledge provided by the network. The goal is to construct ad-hoc, request based sub-networks of knowledge that serve application-specific requests without altering the structure of the original network. This is necessary to provide virtually unlimited request-based and ad-hoc created knowledge structures to knowledge “requesters” which are at the top of the knowledge provisioning pyramid. For that the above container component may by utilised and necessary features added.

The fourth (top) layer is the knowledge usage layer, where individual knowledge requesters (i.e., service ACE’s) are located in order to utilise the knowledge provided by knowledge networks or individual parts thereof.

Clearly, making the above operational architecture an implemented concepts must account for the need of any component and method to effectively operate in a distributed environment and for the fact that the results will eventually be a global knowledge network that should have no theoretical boundaries with respect to the amount of knowledge they may embrace or the type of knowledge to be handled. Also, it must consider that individual components should have the capacity to retain and maintain a memory that comprises the data and knowledge sources they embrace as well as relevant information of neighbouring components in order to maintain the distributed structures of the network. This ‘memory’ needs to be a machine-understandable syntax, comprising different standards in order to maintain semantic integrity and coherence of the knowledge embraced.

Overall, this perspective implies threefold high-level requirements for the implementation of knowledge networks. Firstly, structural requirements that provide necessary components capable of holding knowledge at different levels of granularity including the implementation of a highly flexible framework capable of linking individual knowledge components or any group thereof into distinct purpose-build sub-networks. Secondly, behavioural requirements which deal with more dynamic aspects of knowledge networks such as self-organization, self-optimisation, self-adaptation and self-configuration activities. Thirdly, predictive requirements enabling detailed analytics of individual knowledge components in order to derive new, useful and understandable knowledge.

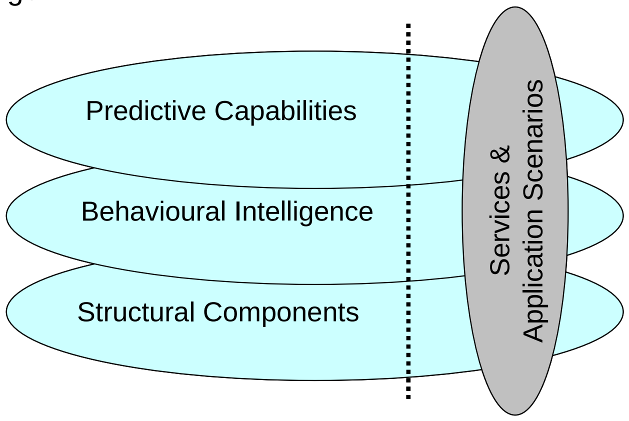


Figure 4: Conceptual Components of Knowledge Networks.

Based on the above a three stage design and implementation process is envisioned that will allow knowledge networks to evolve from the provision of basic factual information into a highly dynamic and intelligent vehicle exhibiting not only a high degree of autonomic behaviour but also predictive capabilities. This process is depicted in Figure 4, where the conceptual components identified above are organised into individual layers representing distinct aspects of knowledge networks, while the fourth layer, services & application scenarios, represents a validation and demonstration facility that will be used throughout the project in order to evaluate the correctness and the performance of the methods and structures proposed.

The specifications detailed in the following of this section mostly concern the structural aspect of knowledge networks. Some mechanisms to deal with behavioural intelligence are detailed in Section 5, while predictive capabilities are left for studying in later stages of the project.

4.3 Knowledge Networks Specifications

Let us now go in details about the specific components that we have identified can be used as building blocks for making the above operational perspective pratical.

4.3.1 Knowledge Component

Two main components make up the Knowledge Network. These are Knowledge Atoms (KA's) and Knowledge Containers (KC's). These components are sub-classes of Knowledge Component. The representation of both components will be facilitated entirely through the use of XML. This not only allows for the dynamic extensions of individual contextual aspects of both KA's and KC's, it also provides a standardised machine readable format which is widely used in current applications and standards. Furthermore it allows for different out of memory storage such as flat files and databases. Figure 5 shows the relationship between these elements.

The contextual information also follows this structure. Both atoms and containers will store the component's metadata as exemplified in Figure 6.

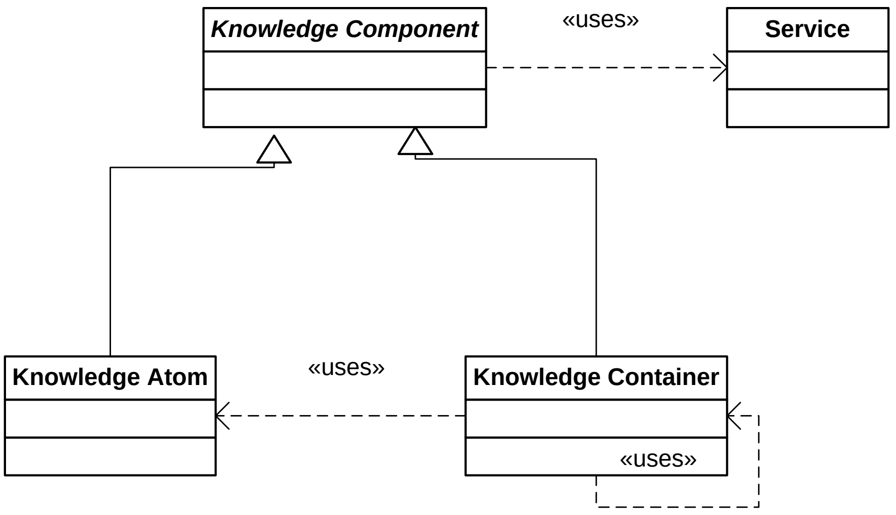
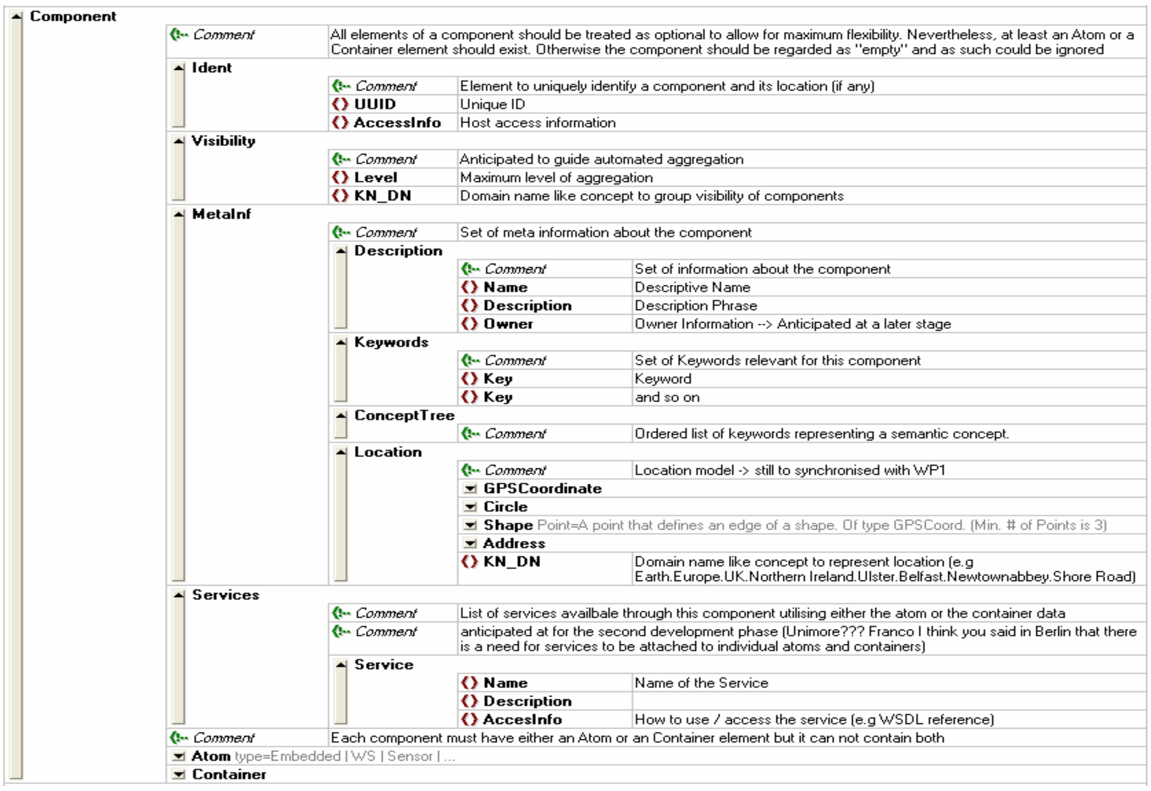


Figure 5: Knowledge Component Relationships.



- **Component**
  - *Comment*: All elements of a component should be treated as optional to allow for maximum flexibility. Nevertheless, at least an Atom or a Container element should exist. Otherwise the component should be regarded as "empty" and as such could be ignored
  - **Ident**
    - *Comment*: Element to uniquely identify a component and its location (if any)
    - **UUID**: Unique ID
    - **AccessInfo**: Host access information
  - **Visibility**
    - *Comment*: Anticipated to guide automated aggregation
    - **Level**: Maximum level of aggregation
    - **KN_DN**: Domain name like concept to group visibility of components
  - **MetaInf**
    - *Comment*: Set of meta information about the component
    - **Description**
      - *Comment*: Set of information about the component
      - **Name**: Descriptive Name
      - **Description**: Description Phrase
      - **Owner**: Owner Information --> Anticipated at a later stage
    - **Keywords**
      - *Comment*: Set of Keywords relevant for this component
      - **Key**: Keyword
      - **Key**: and so on
    - **ConceptTree**
      - *Comment*: Ordered list of keywords representing a semantic concept.
    - **Location**
      - *Comment*: Location model -> still to synchronised with WP1
      - **GPSCoordinate**
      - **Circle**
      - **Shape** Point=A point that defines an edge of a shape. Of type GPSCoord. (Min. # of Points is 3)
      - **Address**
      - **KN_DN**: Domain name like concept to represent location (e.g Earth.Europe.UK.Northern Ireland.Ulster.Belfast.Newtownabbey.Shore Road)
  - **Services**
    - *Comment*: List of services availbale through this component utilising either the atom or the container data
    - *Comment*: anticipated at for the second development phase (Unimore??? Franco I think you said in Berlin that there is a need for services to be attached to individual atoms and containers)
    - **Service**
      - **Name**: Name of the Service
      - **Description**
      - **AccesInfo**: How to use / access the service (e.g WSDL reference)
  - *Comment*: Each component must have either an Atom or an Container element but it can not contain both
  - **Atom** type=Embedded | WS | Sensor | ...
  - **Container**

Figure 6: Knowledge Component Contextual Information.

Simplified, each knowledge component has to provide a range of information that: uniquely identify it within the knowledge space; reveal its current location; semantics describing the knowledge referenced; services provided and of course information that describes the knowledge source itself. It has to be stressed that the set of elements shown are by no means complete. For specific implementations, other more specific elements may be added as desired. However, the elements shown are compulsory to enable access to the underlying data source and to facilitate knowledge aggregation within the organisational layer of the knowledge network. The atoms and containers will then also store their own component specific information as defined in the following sections.

The knowledge component also provides a set of abstract methods that are required to be implemented by all of its sub-classes. These methods allow the user to store and retrieve contextual or service information and are depicted below.

«interface» **Component**
+setModel(in model : XML Element) : XML Element *+getModel(in name : string(idl)) : XML Element* *+removeModel(in name : string(idl))* *+addService(in model : XML Element) : XML Element* *+removeService(in name : string(idl))* *+getServiceList() : XML Element*

Figure 7: Component Interface.

An abstract model has to be implemented that allows for specific extensions in order to accommodate for individual data sources as e.g. listed in Table 2:. The rational of this model is to enable individual knowledge users to access data, stemming from various sources, through a generic interface. The knowledge component provides an abstract interface that must be implemented by its sub-components – namely the knowledge atom or container. This interface provides methods to add or retrieve contextual information or service information. The user can retrieve the whole context or context relating to just a particular element through the getModel methods. A service can be added, which will also add metadata to the component. This metadata will be added in a service section but some may also be added to the context information. Context or a service can also be removed. The service will then provide its own interface to allow another component to use it. This interface is illustrated in Figure 7.

4.3.2 Add-Ons

One of the objectives is to keep the components as lightweight as possible. Because of this, the main components will only have a limited functionality which is only relevant to the construction of networks itself. To make the knowledge network completely generic however, there will be times when extra functionality will need to be added to satisfy the user requirements. One such instance would be when an atom is required to keep a history of its values. In this case, it may be required to periodically write its values to a database.

Table 1: List of Elements and possible Add-ons to Knowledge Network Components.

Name	Status	Description
Tree	Proposed	Semantic tree representation of keywords.
Map	Proposed	Semantic map representation of keywords
Location	Proposed	A concept representing the location of a knowledge source (e.g. physical location of a sensor). This concept should be synchronised with WP1.
Owner	Proposed	Owner information
Access	Proposed	Access information
Lifetime	Proposed	Knowledge lifetime information
QoX	Proposed	Individual Quality Measures
Trust	proposed	Quality of Trust, Trust Management
Security	Proposed	Security and Encryption Mechanism (to be specified)
Log	Proposed	Element to provide relevant usage, access and error events
Statistics	proposed	Element to provide relevant statistical information about the use, access of components, hosts, etc.

This sort of functionality is not compulsory for the general operation of the knowledge network and so add-on components will provide the extra functionality. This type of functionality is of particular importance to other work packages as it provides a mechanism to dynamically load dedicated services. Thus each WP may develop specific extensions to KN components that deal exclusively with the aspects addressed by a particular WP.

This Section provides details about the structure, the purpose and the scope of XML elements that have been identified to be useful for the representation, handling and supervision of knowledge as well as the context the knowledge occurs in. Unless stated otherwise, none of the elements are exclusive to a single component such that they may be added, removed or modified to the contextual part of a KA or KC as desired. Furthermore, not all of the elements listed are exclusively relevant to WP5, the rationale of some elements is to provide an XML "interface" to implement specifics of other WP's. For instance, the elements "Trust" and "QoC" represent elements that are reserved spaces to be used (and specified) by other WP's. In this case WP4 and WP6, respectively. Table 1 lists these elements and may serve as a basis for future discussions with other WP's.

Add-on components will include services to be executed by the component. These components will be passed as serialised Java classes to a service handler, which will re-construct and invoke them. Each service added will run on a separate thread to allow for independent operation from other services. Metadata will be required to describe the add-on component to allow the system to operate it. The metadata stored in any component is dynamic and can be altered depending on circumstances. The user of the component can also retrieve parts of the metadata depending on what service he requires. The interface used to add a service component is also defined by XML publishing relevant information to load, invoke and access a service. The specification thereof is shown in Figure 8.

```
<Services>
    <!--Extra services the component can provided as added components-->
    <Service>
        <Name>The name of a service this component provides as an add-on</Name>
        <Description>Sementic description of the service</Description>
        <URI>The address of the service, can be null for a local service</URI>
        <ClassName>The Java class name of the service object</ClassName>
        <Login>
            <User>The username to access if required</User>
            <Password>The password to access if required</Password>
        </Login>
        <Parameters>
            <!--Intitialisation parameters of the service-->
            <Parameter>A single parameter for the method<Name>The parameter name</Name>
                <Type>The parameter type</Type>
                <Value>The parameter value</Value>
            </Parameter>
        </Parameters>
        <Methods>
            <!--A list of methods for the service-->
            <Method>
                <!--A single method specification-->
                <Name>The methods name</Name>
                <Description>Sementic description of the method</Description>
                <Return>The return type of the method </Return>
                <Parameters>
                    <!--A list of parameters for the method-->
                    <Parameter>A single parameter for the method<Name>The parameter name</Name>
                        <Type>The parameter type</Type>
                        <Value>The parameter value</Value>
                    </Parameter>
                </Parameters>
            </Method>
        </Methods>
    </Service>
</Services>
```

Figure 8: Service Method Description.

When an add-on registers itself at an atom, it also registers its metadata. This metadata may be stored completely in the service section, or it may add to the component description in general, adding to its knowledge. The user then has the option of retrieving knowledge (semantics), service descriptions, or parts of the semantics based on the keywords they enter. To provide this functionality, the component will realise a 'ComponentHandler' interface that will have a set of methods to allow the user to retrieve context. The appropriate component (a Knowledge Atom) will also implement other methods to retrieve the service information. The methods are shown in the Component section 4.6.1 of the Interface section.

4.3.3 History Component

Part of the philosophy of the knowledge network is to provide a kind of who, what, when, where functionality. A history component is useful for writing values of a source at periodic intervals to a database to be later retrieved and analysed. As we are allowing add-on components to be completely generic, we can implement a standard history component with a known functionality. This is the default component that users can add-on to write a history of a source. This does not prevent a user from writing their own history component if they want to provide different functionality.

```
<Service>
        <Name>History Component</Name>
        <Description>Generic history component</Description>
        <URI>The address of the service</URI>
        <ClassName>The Java class name of the component object</ClassName>
        <Parameters>
                <!--Intitialisation parameters of the service-->
                <Parameter>
                        <!--milliseconds-->
                        <Name>fetchValueInterval</Name>
                        <Type>java.lang.Integer</Type>
                        <Name>host</Name>
                        <Type>java.lang.String</Type>
                        <Name>port</Name>
                        <Type>java.lang.String</Type>
                        <Name>username</Name>
                        <Type>java.lang.String</Type>ing
                        <Name>password</Name>
                        <Type>java.lang.Str</Type>
                </Parameter>
        </Parameters>
        <Methods>
                <!--A list of methods for the service-->
                <Method>
                        <!--A single method specification-->
                        <Description>Retrieve a list of elements</Description>
                        <Name>getValue</Name>
                        <Return>Element</Return>
                        <Parameters>
                                <!--A list of parameters for the method-->
                                <Parameter>
                                        <Name>startTime</Name>
                                        <Type>java.util.Date</Type>
                                </Parameter>
                                <Parameter>
                                        <Name>stopTime</Name>
                                        <Type>java.util.Date</Type>
                                </Parameter>
                        </Parameters>
                </Method>
        </Methods>
</Service>
```

Figure 9: Service Interface.

The main goal of knowledge networks is to access and organize data coming from a very heterogeneous set of knowledge sources. However, one of the most valuable applications of knowledge networks should be storing the history of values coming from sensors. By this way, a better and more accurate perception of the environment is achievable. Due to the availability of past data, it should for example be possible to infer something that is unknown about the present or even about the future. In our opinion knowledge networks should in fact deal with the problem of accessing data from a distributed and heterogeneous set of sensors but also with the problem of organizing and consolidate sensor data in a sort of knowledge.

Due to the fact that the storage is probably not the main goal of knowledge networks but has lots of implications we propose specifications of a standard history component. It has to be implemented not inside a knowledge atom but as a separate component. To realize its purposes it has to register itself to a KA (exposing values to be saved) using the *addService()* method (inherithed from KNComponent). Internally it simply fetches every *fetchValueInterval* milliseconds a value from the KA and stores it in a database. It also provides the method *getValue(startTime, stopTime)* to the KA needed to fetch data from the backend instead of the live sensor. By this way registering the history component to a KA simply produces a KA capable of storing and retrieving historical data.

The following XML piece is an example on how to register the default history component to a knowledge atom. Note that this not only provides the functionality to specify the specific component to be loaded but also publishes its configuration as well as its public interface.

4.3.4 Knowledge Atom, KA

Representing the most basic component of a knowledge network, the rough structure of a knowledge atom is depicted in Figure 10. It contains a knowledge source object and relevant descriptions that provide the context of the object contained. Within the scope of WP5, the sole purpose of a knowledge atom is to introduce a specific data source into the scope of a knowledge network and to provide generic access to the underlying data source; it is not concerned about any organisational aspects within or outside the knowledge network nor is it responsible for the configuration, maintenance or (de-) registration thereof.

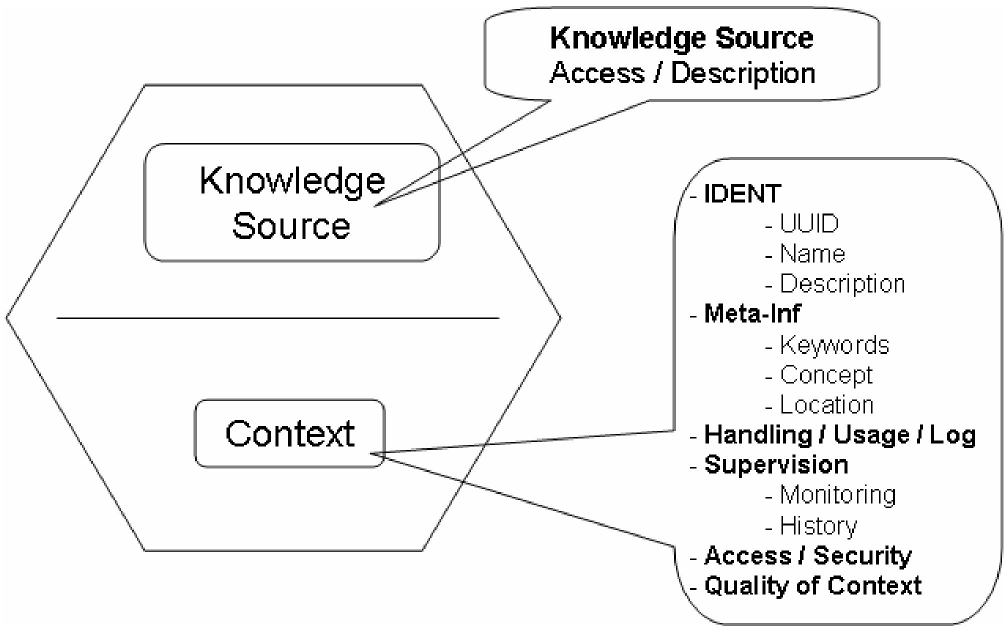


Figure 10: Knowledge Atom.

Definition 1: Knowledge Atom – is a data object containing a single piece of knowledge and relevant semantics.

Simplified, a knowledge atom encapsulates two objects: firstly it contains a knowledge object which reflects a single knowledge entity independent of its type, size or context; secondly it has relevant semantics of the knowledge object attached providing relevant descriptive context, system and usage based information that are relevant for the creation, maintenance and observation of the knowledge object. For instance, a knowledge atom could encapsulate the reading of a single sensor (e.g. temperature reading @ location GPS_COORDINATES[X]), where attached semantics could include the GPS location of the sensor, the purpose of the sensor, the required update frequency, etc. On the other hand the knowledge object could reflect a more complex structure such as e.g. the human DNA code.

Within the context of a global knowledge network, knowledge atoms may be seen as "protected" objects, that is that they are not devisable, which is based on the simple fact that, independent of its complexity, they only embrace a single "piece" of knowledge. Nonetheless, the semantics of a knowledge atom may be extended if necessary. It is also envisioned that knowledge atoms and the knowledge they embrace exist locally rather than in a distributed environment. However, as a whole knowledge atoms may be shared, cloned, referenced or transported globally throughout the network. In the sequel knowledge atoms are denoted by $k = (o, S)$, where o represents the knowledge object and S its semantics, both are subject to further specification.

Figure 11 shows the full xml description of the knowledge atom concept as available to date.

```
<KNComponent>
    <Ident>
        <UUID>Unique ID for the component</UUID>
        <Type>Atom</Type>
        <AccessInfo>
            <URI>Uniform Resource Identifier to locate the component</URI>
            <Login>
                <User>The username to access if required</User>
                <Password>The password to access if required</Password>
            </Login>
        </AccessInfo>
    </Ident>
    <MetaInf>
        <Description>
            <Name>Description of the component</Name>
        </Description>
        <Keywords>
            <Key>Arbitrary key word or structure</Key>
        </Keywords>
    </MetaInf>
    <Atom>
        <Service>
            <Name>The name of the service this atom provides by default</Name>
            <Description>Sementic description of the service</Description>
            <URI>The address of the service, can be null for a local service</URI>
            <ClassName>The Java class name of the service object</ClassName>
            <Login>
                <User>The username to access if required</User>
                <Password>The password to access if required</Password>
            </Login>
            <Parameters>
                <Parameter>A single parameter for the method<Name>The parameter name</Name>
                    <Type>The parameter type</Type>
                    <Value>The parameter value</Value>
                </Parameter>
            </Parameters>
            <Methods>
```

```
                <Method>
                    <Name>isAlive</Name>
                    <Return>Boolean -->True if atom is accessible</Return>
                </Method>
                <Method>
                    <Name>getValue</Name>
                    <Return>Element --> The atom value</Return>
                </Method>
                <Method>
                    <Name>getType</Name>
                    <Return>Element --> The atom type</Return>
                </Method>
                <Method>
                    <Name>getConfig</Name>
                    <Return>Element --> The atom access configuration</Return>
                </Method>
            </Methods>
        </Service>
        <Data>
            <Value>Atom value</Value>
            <Type>Data type of the atom value</Type>
            <Config>Configuration of the atom source</Config>
        </Data>
    </Atom>
    <Services>
        <Service>
            <Name>The name of a service this component provides as an add-on</Name>
            <Description>Sementic description of the service</Description>
            <URI>The address of the service, can be null for a local service</URI>
            <ClassName>The Java class name of the service object</ClassName>
            <Login>
                <User>The username to access if required</User>
                <Password>The password to access if required</Password>
            </Login>
            <Parameters>
                <Parameter>A single parameter for the method<Name>The parameter name</Name>
                    <Type>The parameter type</Type>
                    <Value>The parameter value</Value>
                </Parameter>
            </Parameters>
            <Methods>
                <Method>
                    <Name>The methods name</Name>
                    <Description>Sementic description of the method</Description>
                    <Return>The return type of the method </Return>
                    <Parameters>
                        <Parameter>A single parameter for the method<Name>The parameter name</Name>
                            <Type>The parameter type</Type>
                            <Value>The parameter value</Value>
                        </Parameter>
                    </Parameters>
                </Method>
            </Methods>
        </Service>
    </Services>
</KNComponent>
```

Figure 11: Current XML Specification of Knowledge Atoms.

As described previously, the rationale of knowledge atoms is to introduce new data stemming from various data sources into the scope of the knowledge network. To be properly registered, organized and used two types of generic interfaces have to be established. The component interface needs to be extended to allow for source values and services to be accessed. Figure 12 describes the Knowledge Atom interface, whereas a preliminary specification of the knowledge atom component is shown in Figure 11, where the data access interface is realized by the getValue, getType and getConfig methods. The second aspect, the exchange of contextual information, is realized through the Component interface, which is also implemented by the knowledge container component. The third aspect is the service access. This is done through the addService and removeService methods. AddService will add an add-on component to the atom that will run a service. This can be of any type and is described by the serviceDescription.

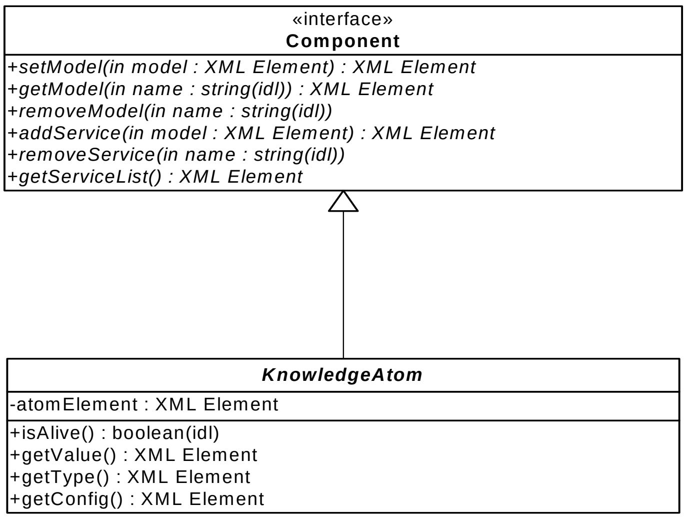

«interface»
Component
+setModel(in model : XML Element) : XML Element
+getModel(in name : string(idl)) : XML Element
+removeModel(in name : string(idl))
+addService(in model : XML Element) : XML Element
+removeService(in name : string(idl))
+getServiceList() : XML Element
KnowledgeAtom
-atomElement : XML Element
+isAlive() : boolean(idl)
+getValue() : XML Element
+getType() : XML Element
+getConfig() : XML Element

Figure 12: Knowledge Atom Interface.

Let us now briefly outline some possible data sources that are of particular interest to the concept of knowledge networks. The type and number of knowledge sources presented in the table below is by no means complete with respect to the type of sources available in the "real" world. Nevertheless it is envisioned that a multitude of different knowledge sources will be incorporated through the concepts of knowledge atoms through the course of the project and beyond.

Table 2: Example Data Sources for Knowledge Atoms.

Type	Description
Embedded	Small pieces of data to be embedded into the atom structure itself. Particular relevant for non-volatile data.
Web Service	Data made available through a web service.
Sensor	Data available through the concept of sensors. Possible sub-types may include hardware based sensors, software based sensors and sensor arrays of either of the above.
Knowledge Broker	Data made available through a brokerage system. Specific implementation will depend on the type of system to be used
ACE	Data made available though the generic or specific interface of an ACE (this type will depend on the realisation of ACE's)
Knowledge Container	Atoms that require data from within the KN require access to KC's

4.3.5 Knowledge Container, KC

The Knowledge Container is the aggregating component of the knowledge network. It will group elements to define the structure of the network, while the values are retrieved from atoms. The containers can hold other containers or other atoms or both. A container may also act as an aggregated source and store aggregated information in an atom, while also storing other containers to navigate to lower levels in the network. Figure 13 is a diagram describing the basic concepts of a Knowledge Container.

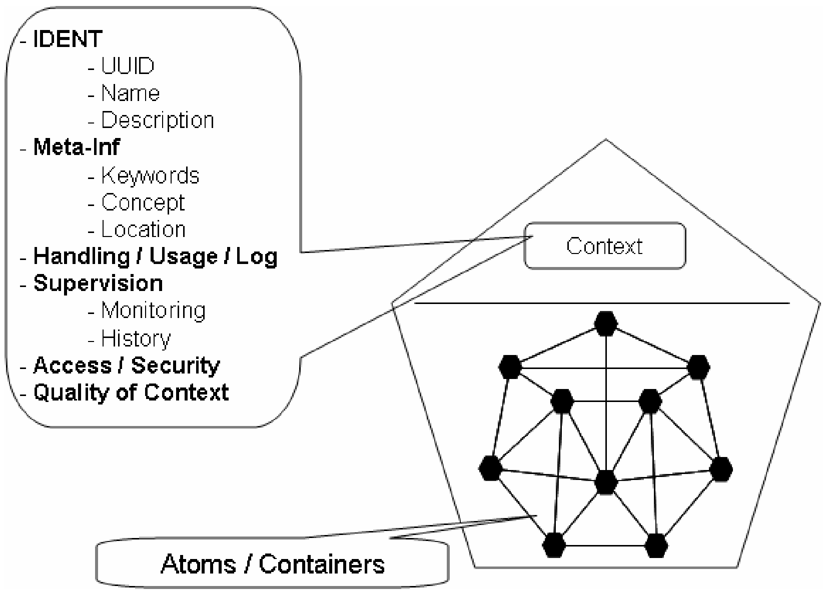


Figure 13: Knowledge Container.

Definition 2: Autonomous Knowledge Component– is a purpose built, loosely coupled collection of knowledge atoms / knowledge container.

The algorithms used for aggregation can be of any type and are yet to be decided upon. The hierarchical structure we provide is slightly different to an ad-hoc peer-to-peer system. It does not allow any element to link to any other, as some restrictions are provided by the hierarchical structure. To this extent, we do not want an atom to be allowed to also be a container that may aggregate other elements in a random manner. This is more chaotic than the structure we envisage. We have decided to keep the knowledge container and the knowledge atom concepts separate. The atoms will be aggregated by algorithms producing a largely static structure that will change only gradually through time. We then propose to generate overlay networks defined by temporary links between atoms to produce a more dynamic structure. This structure however is secondary compared to the static hierarchy and will not actually change the hierarchical structure. Similar to knowledge atoms, additional semantic information may be attached to a KC, thus providing relevant information about the type, scope, purpose, usage, etc. of the knowledge they embrace, the purpose they were created for and the way they are used. Unlike knowledge atoms, KC's are freely extendable, dividable and modifiable, that is that new knowledge atoms can be added, older ones may be removed or hierarchical links between them may be modified, created or removed at any time. Figure 14 describes the additional metadata that may be used to describe a knowledge container. This stores the location of components aggregated by the container.

```xml
<KNComponent>
    <Ident>
        <UUID>Unique ID for the component</UUID>
        <Type>Container</Type>
        <AccessInfo>
            <URI>Uniform Resource Identifier to locate the component</URI>
            <Login>
                <User>The username to access if required</User>
                <Password>The password to access if required</Password>
            </Login>
        </AccessInfo>
    </Ident>
    <MetaInf>
        <Description>
            <Name>Description of the component</Name>
        </Description>
        <Keywords>
            <Key>Arbitrary key word or structure</Key>
        </Keywords>
    </MetaInf>
    <Container>
        <Ident>
            <UUID>Unique ID for the component</UUID>
            <Type>Atom | Container</Type>
            <AccessInfo>
                <URI>Uniform Resource Identifier to locate the component</URI>
                <Login>
                    <User>The username to access if required</User>
                    <Password>The password to access if required</Password>
                </Login>
            </AccessInfo>
        </Ident>
        <Ident>
            <UUID>Unique ID for the component</UUID>
            <Type>Atom | Container</Type>
            <AccessInfo>
                <URI>Uniform Resource Identifier to locate the component</URI>
                <Login>
                    <User>The username to access if required</User>
                    <Password>The password to access if required</Password>
                </Login>
            </AccessInfo>
        </Ident>
    </Container>
    <Services>
        <Service>
            <Name>The name of a service this component provides as an add-on</Name>
            <Description>Sementic description of the service</Description>
            <URI>The address of the service, can be null for a local service</URI>
            <ClassName>The Java class name of the service object</ClassName>
            <Login>
                <User>The username to access if required</User>
                <Password>The password to access if required</Password>
            </Login>
            <Parameters>
                <Parameter>A single parameter for the method<Name>The parameter name</Name>
                    <Type>The parameter type</Type>
                    <Value>The parameter value</Value>
                </Parameter>
            </Parameters>
            <Methods>
                <Method>
                    <Name>The methods name</Name>
                    <Description>Sementic description of the method</Description>
                    <Return>The return type of the method </Return>
                    <Parameters>
                        <!--A list of parameters for the method-->
                        <Parameter>A single parameter for the method<Name>The parameter name</Name>
                            <Type>The parameter type</Type>
                            <Value>The parameter value</Value>
                        </Parameter>
                    </Parameters>
                </Method>
            </Methods>
        </Service>
    </Services>
  </KNComponent>
```

Figure 14: Current XML Specifications of Knowledge Containers.

As has been depicted in Figure 13 the underlying concept of a knowledge container is similar to the concept of a knowledge atom. That is, it encapsulates knowledge. However, unlike knowledge atoms the purpose of a knowledge container is to organize knowledge in a semantic or spatial fashion rather than providing access to the underlying information. Basically a knowledge container or KC may embrace (or reference) any number of knowledge atoms or knowledge containers independent of their location (locally or remotely). In order to enable the construction of larger highly distributed knowledge network structures and to support the concept of self-similarity, individual knowledge containers may contain other containers or any number of knowledge atoms. This allows the construction of knowledge networks utilising the concept of KC's only, whereas the knowledge sources only exist as leaves of a constructed network. In other words, knowledge atoms store the knowledge (or provide access to it) whereas KC's are used to organise it. This concept also enables the construction of networks of networks where each node (KC) of a network-like structure may contain a network itself which in turn could contain networks and so on. Within a P2P environment this concept is known as the construction of super networks which is depicted in Figure 16.

Similar to the knowledge atom, a preliminary specification of the knowledge container component is shown in Figure 14. As shown the KC concept also implements the Component interface which realizes generic access to context related information stemming from KA's as well as KC's. Utilizing this interface abstracts the access of contextual information from both KA's and KC's into a single concept. Other methods specified deal with the registration, access and removal of elements as required in order to construct a network like structure. Note that unlike the knowledge atom implementation, the knowledge container component is not abstract. On the contrary it is intended to be final and as such not to be sub-classed. The reason for this is based on the fact that no specific implementations are envisioned for this component. Instead, required functionality for the organization of knowledge has to be available in all containers.

Organizing containers based on various semantic, hierarchical, geographical or logical concepts the rationale of a KC is to provide purpose-built, structured knowledge on higher levels of granularity. Similar to knowledge atoms, additional descriptive information are attached to a KC via a context object, thus providing relevant information about the type, scope, purpose, usage, etc. of the knowledge they embrace, the purpose they were created for or the way they are used. Unlike knowledge atoms, KC's are freely extendable, divisible and modifiable, that is, that new KA's or KC's may be added whereas others may be removed. Furthermore, any conceptual links implemented to organize individual components may be modified, created or removed at any time. Please note that although some concepts represented through the context part of a KC and a KA respectively may be shared or inherited, they are not the same and therefore not necessarily equivalent.

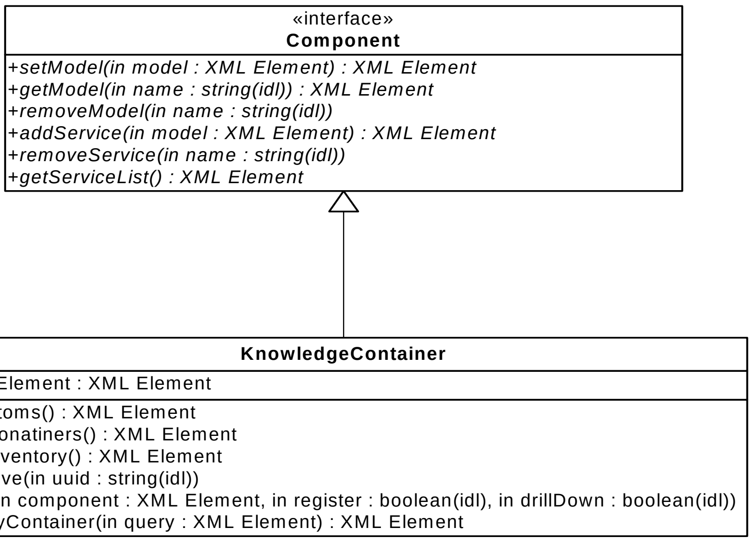

«interface»
Component
+setModel(in model : XML Element) : XML Element
+getModel(in name : string(idl)) : XML Element
+removeModel(in name : string(idl))
+addService(in model : XML Element) : XML Element
+removeService(in name : string(idl))
+getServiceList() : XML Element
KnowledgeContainer
-atomElement : XML Element
+getAtoms() : XML Element
+getConatiners() : XML Element
+getInventory() : XML Element
+remove(in uuid : string(idl))
+add(in component : XML Element, in register : boolean(idl), in drillDown : boolean(idl))
+queryContainer(in query : XML Element) : XML Element

Figure 15: Knowledge Container Interface.

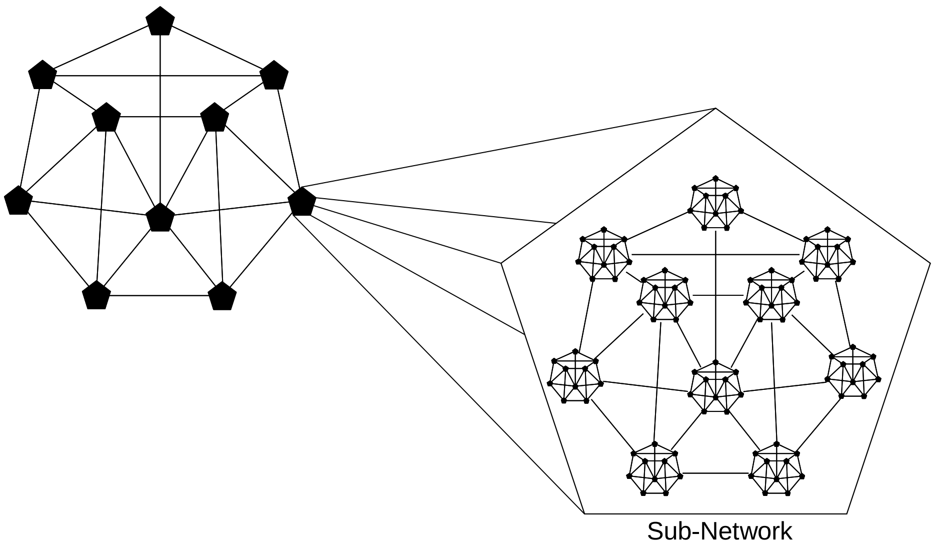


Figure 16: Super Networks

A KC could embrace a number of sensor readings (knowledge atoms) that together reflect a higher concept. For instance, in a weather sensor example, a number of temperature readings can be aggregated into a singe KC to define average or sampled field temperatures in a zone. Alternative, a mix of sensor readings (e.g., humidity, pressure, wind, other than temperature) in a region, could be used to provide weather information at a specific location. Using a weather example, the purpose of a KC could be the provision of weather information of all major cities in e.g. United Kingdom. Assuming that there exist relevant sensors in all cities concerned and that their also exists a dedicated KC for each city (preferably individual KC's reside on a computational resource that is somehow connected to the city they belong too) then another KC (e.g. WEATHER(United Kingdom)[KC[0], KC[1], KC[2], …]) could be created that embrace all other KC's concerned thus providing a central dedicated knowledge resource where individual KC's can be added or removed automatically and which can be used by other services.

Alternatively, the KC's embraced by the WEATHER(United Kingdom) KC may be sub-grouped even further taking additional information into account such as discrete geographical regions. The notion of the original KC could then be extended as follows:

WEATHER(United Kingdom)[
WEATHER(England) [KC[0], KC[1], KC[2], …],
WEATHER(Scotland) [KC[0], KC[1], KC[2], …],
WEATHER(Wales) [KC[0], KC[1], KC[2], …],
WEATHER(Northern Ireland) [KC[0], KC[1], KC[2], …]
]

Although simplistic, the above weather example validates the suitability of the KC approach and shows its flexibility and the extendability towards more complex scenarios. It also shows that KC's can be expressed formally through a dedicated mark-up language, which fosters interoperability among different physical and virtual resources.

4.4 Interfaces, Summary

This section provides a summary of all available interfaces for each component of the knowledge network as depicted below.

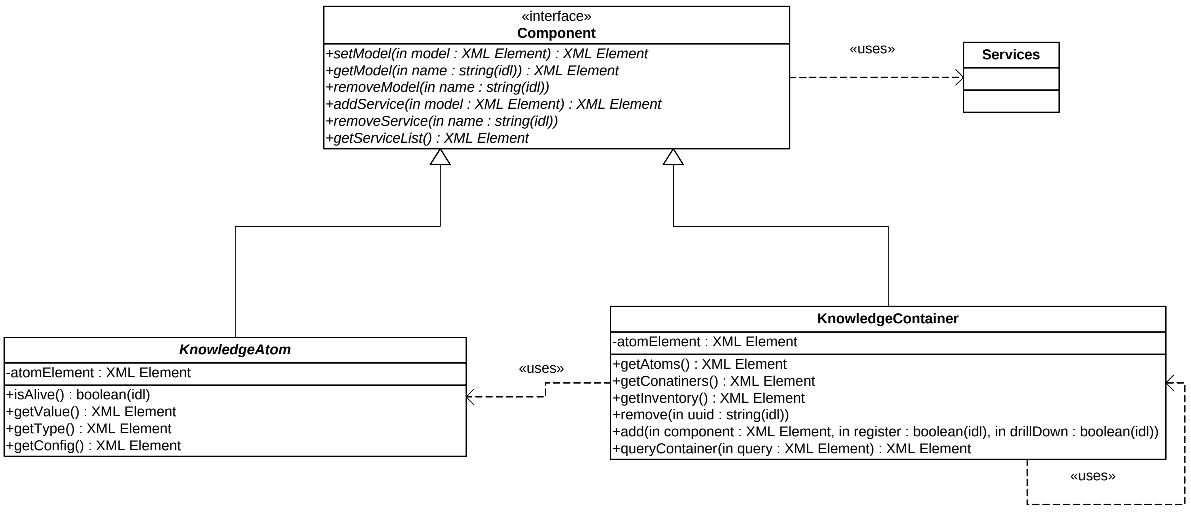

«interface»
Component
+setModel(in model : XML Element) : XML Element
+getModel(in name : string(idl)) : XML Element
+removeModel(in name : string(idl))
+addService(in model : XML Element) : XML Element
+removeService(in name : string(idl))
+getServiceList() : XML Element
«uses»
Services
KnowledgeAtom
-atomElement : XML Element
+isAlive() : boolean(idl)
+getValue() : XML Element
+getType() : XML Element
+getConfig() : XML Element
«uses»
KnowledgeContainer
-atomElement : XML Element
+getAtoms() : XML Element
+getConatiners() : XML Element
+getInventory() : XML Element
+remove(in uuid : string(idl))
+add(in component : XML Element, in register : boolean(idl), in drillDown : boolean(idl))
+queryContainer(in query : XML Element) : XML Element
«uses»

Figure 17: Summary, Interfaces.

4.5 Knowledge Execution

Within the knowledge network component layer as outlined above distributed knowledge resources may be linked together through a reference mechanism indicating the location of individual KC's that are part of another KC. However, for knowledge execution purposes, knowledge components have to be accessed directly, that is a communication link has to be established that allows efficient bi-directional data transfer between relevant components of one or more KC's, thus creating a virtual view of a distinct set of knowledge. Such functionality may be implemented through P2P overlay networks that can be constructed in an ad-hoc manner to be used by a service or application requesting a discrete set of knowledge.

To enable the efficient construction of such virtual views without altering the knowledge network itself the concept of a virtual KC is introduced next. Basically, a virtual KC is, in structure and functionality, equivalent to a real KC. The difference is that it only links to other KC's that already exist on the network, it never links to knowledge resources directly, which allows for complete separation of the knowledge provisioning layer and the knowledge view layer as illustrated through the knowledge provisioning pyramid depicted earlier on. Furthermore, virtual KC's may also be used by services to construct knowledge requests and if populated will provide necessary information to construct the overlay network itself.

One of the advantages of virtual views is that highly specific collections of knowledge can be created in an ad-hoc manner without changing the underlying data structures. Thus, if a service expires its corresponding KC may be destroyed without changing the underlying knowledge network. Utilising KC's to specify and create such views has another advantage, that is that services can specify the type of knowledge they desire directly on the object which will deliver the knowledge once it has been located and necessary overlays have been built. This again fosters self-similarity of knowledge components and interoperability of computational resources.

To retrieve knowledge we will provide a query system that is XML based. This system will navigate the knowledge network to identify the appropriate sources to query. When these have been identified they can be queried and the relevant information retrieved. There will be a query mediator that interfaces with the different XML-based query engines that are to be used. RDF would be a suitable format for navigating the network or querying simple sources such as sensors. However, with complex XML data at the sources we may require something like the declarative XML query language XQuery or the deductive XML query language Xcerpt to query it. The sources will store a query engine suitable to their data and the mediator will then interface with them and control the query process. We also recognise that it may be possible to use the results of the query execution to update the knowledge in the network by strengthening links between sources that are consistently used together to answer a query. These links will produce overlay networks that can be used to help to optimise query executions.

4.6 Knowledge Organization

4.6.1 Batch vs. On-line Organisation

We recognise two different situations when knowledge will be organised. When the network is initially being constructed, it will need to retrieve information from its sensors and organise this in an on-line manner. Once this knowledge is generated, it may be stored at one location (or replicated across multiple one) and then, if the network is re-started, this knowledge can be used to re-configure the network in a batch manner based on the stored knowledge. The organisation will primarily be done using the on-line information as this will dynamically change through time and only be available in an on-line format. Batch organisation can then be used for re-construction. However, with something like a history component, the batch organisation could be run periodically to retrieve historical data to also update the network. In the scope of knowledge networks, both batch and on-line organizations have advantages and can be used separately or in a combined way.

Batch organisation enables a fast access to information because related data is stored at a given location and is already available. There is no need to access different KCs in different locations by following external links. This fast access comes at the cost of possible inconsistencies which may exist between a data source (i.e. a sensor that produces data) and the KC where this data is copied to. Therefore, batch organisation of knowledge is usually used when fast access is necessary and the desired information is valid for a long time. Also, batch organization facilitates storing of heavy-weight information which could have problems in being highly distributed and build on-the-fly.

Online organisation can be used in environments with limited storage space and when it is not possible to organize a priori great amounts of information. Because of the fact that in the online case, data is not replicated to store it in various KCs, only a small amount of additional memory is necessary to save links to the original data. A second advantage is the consistency and freshness of online organised knowledge, in that organized data is built on-the-fly, on the basis of the fresher available information. One drawback of online organisation of knowledge is the fact that requesting data may take some time because desired information can only be obtained indirectly by following at least one link. This, in addition, usually leads to consumption of network resources because the links which have to be followed may point to physically distant locations. Hence, online organisation of knowledge is used in situations where storage capacity is limited or consistency of data plays an important role, whereas the time necessary to obtain information is of minor priority.

The following short examples illustrate the described organisation methods. An array of motion detectors, monitoring a building and supporting an alarm system, would be aggregated in a KC in an online way. The network in this building is most probably capable to handle a number of frequent requests to each sensor relatively fast. Batch organisation would not be suitable because state changes of the sensors have to be recognised as fast as possible. Copying data to a KC and updating it rarely would delay the reaction time of the alarm system. In contrast, a KC that aggregates stated facts which will most probably not change often, would be organised in a batch way. If those facts are for example mathematical theorems like trigonometric functions and these theorems are accessed very frequently for different calculations, an online organisation would result in a lot of unnecessary network traffic. Replicating the concerned theorems once and copying them to an appropriate KC would minimise network resource consumption. In this example, inconsistency is no problem because mathematical theorems are unlikely to change.

4.6.2 Vertical vs. Horizontal Organization

Naturally, the structure and relations of a knowledge network is highly dynamic and mainly depends on the scope the network is used for. Being capable of self-adapting itself to better serve future requests as well as reacting on the dynamics of volatile knowledge values as well as sources a knowledge network will evolve over time. However, internally two different concepts of organisation are used to shape different levels of knowledge granularity, namely vertical and horizontal structures.

At the bottom of both organisational concepts are always knowledge atoms, which is due to the fact that KA's are seen as active knowledge sources that feed information into the network rather than organising them. Considering a tree like organisational structure for either concept, that is when avoiding cyclic references of components within the same branch, atoms are always found as a leaf node. Containers on the other hand are solely designed to organise knowledge, such that they are never leaves but build up the remainder of the tree or to be more precise knowledge network. Note the distinction between a tree and a network is important for the organisation but not for the representation. That is that when organising or querying a knowledge network cyclic references are detected and serve as endpoints in order to avoid processing the same branch of knowledge over and over again. Nevertheless, within the global structure of a knowledge network cyclic references can not be avoided and are in fact desired to allow for multiple entry points into a network for dedicated maintenance, organisation and querying tasks.

Simplified, vertical structures are those relationships that involve multiple knowledge containers at different levels, where the level of knowledge granularity is equal to the position of the container the knowledge is referenced from. The rationale of a vertical organisation is that high level concepts of sources that exist at a lower level can be grouped together thus providing a conceptual view at different levels of granularity. This concept is visualised in Fig. 17, where several KC's are organised at different levels to provide different views of the atoms that are located at the bottom of the tree. The main advantage of such an organisation is that individual knowledge can be accessed directly by querying for the concept used to organise the knowledge rather than iterating over all available atoms independently to determine if they are relevant to a current query or not.

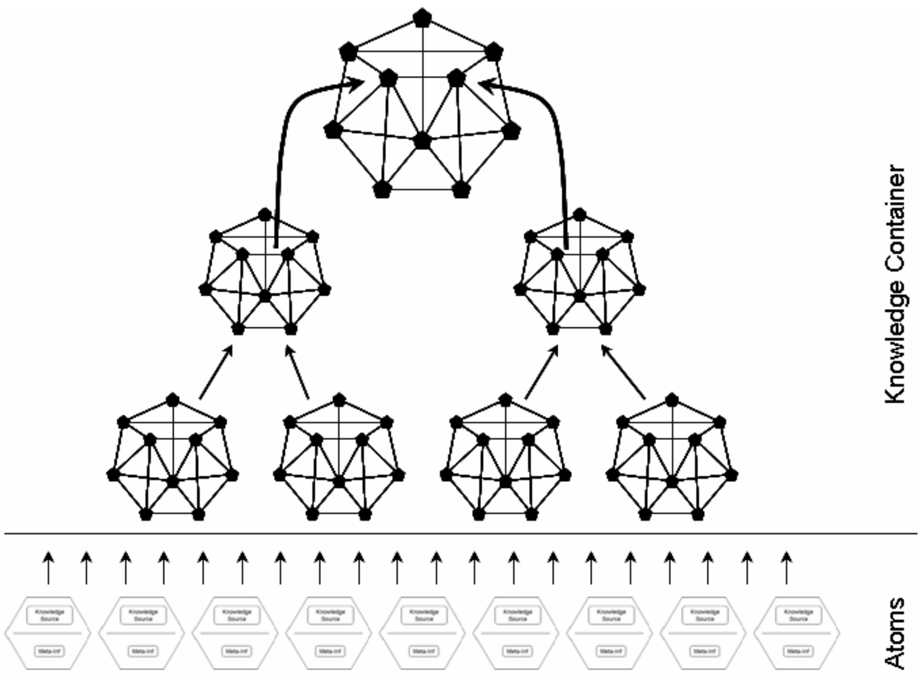


Figure 18: Vertical Organisation.

For example, in a sensor network scenario, a number of different readings, such as temperature, wind force, wind direction etc. may be available for a number of different locations, e.g. each city in Europe (Figure 18). Assuming that each sensor exposes the Atom concept and registers itself into the scope of a knowledge network, then these atoms can be organised automatically based on the concepts they reflect.

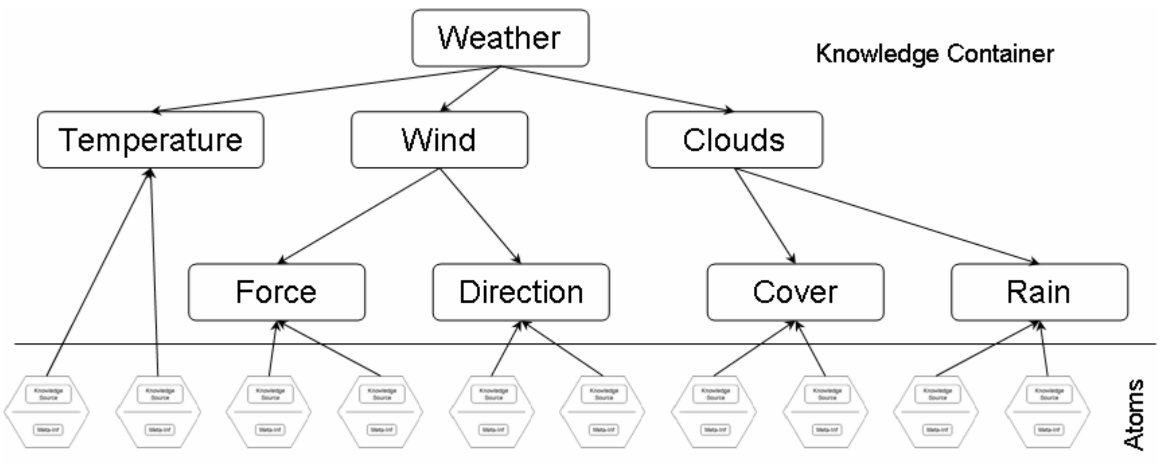


Figure 19: Example – Vertical Knowledge Organisation.

As depicted in Figure 19, all sensors are grouped based on their type, e.g. temperature. Furthermore some sensors are then grouped to higher concepts such as wind, clouds or weather as the top level container in this example. Horizontal structures, as depicted in Figure 19, are those relationships that enable to organise knowledge that is conceptually located at the same level of abstraction.

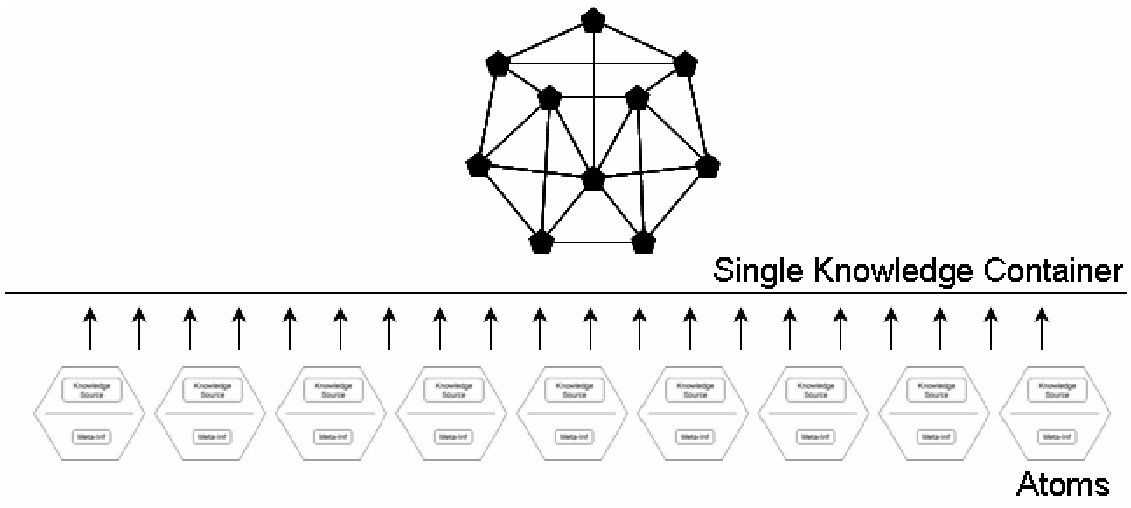


Figure 20: Horizontal Organisation.

Such structures can be used to choose and select an area in the knowledge space on the basis of given attributes. For instance, each sensor in the sensor network scenario above, may exhibit the concept of its location, e.g the sensor is located in London. Then all atoms that are also located in London may be organised into a single container independent of their type, reading etc. Horizontal structures are likely to be (conceptually) distributed on a wider area than vertical ones and as such are not so fine-grained. Therefore such structures may be faster to query that vertical ones.

4.7 Knowledge Network Components and ACE's

In this section we are going to sketch some ideas about how the relationship between knowledge networks and ACE's. These kind of relations are fundamental since:

1. The components of the knowledge network will be implemented by means of ACE's
2. Application-level ACE's will access the knowledge network to acquire context information.

Before describing the above two relations let us briefly recap the ACE architecture as currently proposed in WP1 (see Figure 21).

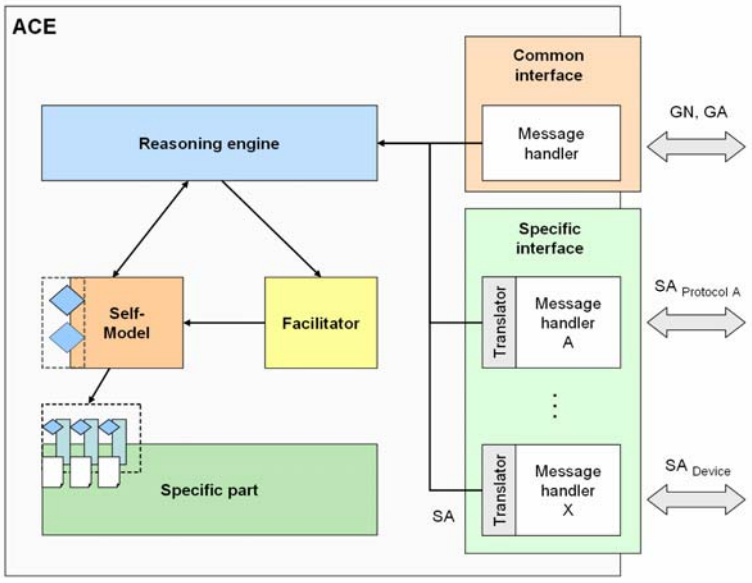


Figure 21: Currently proposed ACE architecture (the specific interface has made more explicit than in the ACE architecture discussed in D1.1).

The ACE architecture is composed of six main parts:

1. The **specific part** is where the specific services of an ACE are located. For example, an ACE offering some kind of message delivering service will have in the specific part all the code dealing with the actual delivering of messages.
2. The **self-model** is a Finite State Automaton (FSA) describing the behaviour of an ACE. Specific transition in the FSA triggers the methods in the specific part. For example, depending on its self-model, the ACE decides whether to fulfil a request or not, and which specific service to employ.
3. The **reasoning engine** runs the self-model triggering state transitions on the basis of received events
4. The communication between the ACE and the external world takes place either with a discovery protocol called GA-GN (this discovery is carried on by the **common interface**), or by specific messages to access the ACE specific functionalities (these messages are handled in the **specific Interface**[1]).

[1] The ACE architecture we present here is the same that has been defined in WP1, and that is actually described in D1.1. However, in order to better clarify how KN components can be implemented via ACEs, we have made more explicit than in D1.1 the presence of a specific interface.

5. The **facilitator** is the component in charge of enforcing autonomic behaviours by changing dynamically the self-model's FSA. For example, the facilitator can add new states to the FSA or rewire its links to change the ACE behaviour.

This ACE architecture provides a good separation of concerns between the various parts of an ACE, and enforces flexibility and autonomic behaviour.

4.7.1 Implementing Knowledge Network Components with ACEs

Now, by recalling that Knowledge Atoms (KA) and Knowledge Containers (KC) are the two main components of the knowledge network, Let us know show how noth these two components can be readily mapped in the ACE architecture.

For a Knowledge Atom (see Figure 22):

- The **specific part** comprises the methods to access underlying information sources as well as individual reasoning and knowledge processing capabilities. Which capabilities are to be implemented differs from KA to KA.
- The **self-model** maintains the state of the aggregation process. It may be a finite state automaton indicating what the KA is doing. For example, it can simply be an automaton with 2 states: "collect data" and "sleep". The KA cycles between these two states.
- The **facilitator** changes the self-model to enact autonomic algorithms, possibly changing the way in which aggregated values are computed.
- With the **GN-GA protocol** the KA can describe to enquiring ACE's which knowledge it is able to produce. For example, it can express semantically in its GA message that it "provides the average temperature over a specific area".
- Finally, a **message handler in the specific interface** could be in charge of delivering the proper knowledge, provided by the KA, to a requester (e.g., actually get the proper average value).

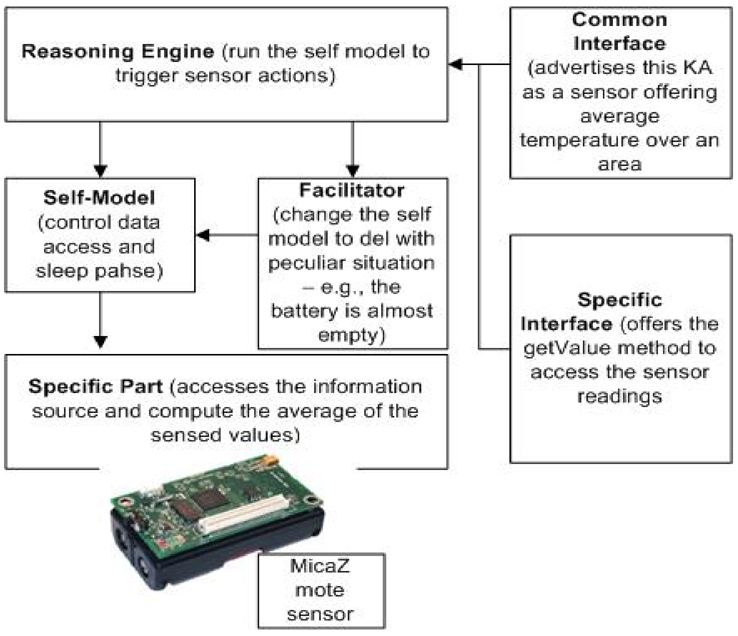

Reasoning Engine (run the self model to trigger sensor actions)
Common Interface (advertises this KA as a sensor offering average temperature over an area)
Self-Model (control data access and sleep pahse)
Facilitator (change the self model to del with peculiar situation – e.g., the battery is almost empty)
Specific Interface (offers the getValue method to access the sensor readings)
Specific Part (accesses the information source and compute the average of the sensed values)
MicaZ mote sensor

Figure 22: Knowledge atom implemented in the ACE architecture

For a Knowledge Container (see Figure 23):

- The **specific part** comprises the methods to link (i.e., hold a reference) to the various KA
- The self-model maintains the state of which KA are contained (i.e. linked) by the KC and decides whether to link to new ACE or to remove previously existing links.
- The facilitator can change the self model to deal with unexpected situations. For example, if a network link to a KA breaks down, the facilitator can change the self model to reflect the fact that KA is no longer available.
- With the **GN-GA protocol** the KC can describe both that it can offer a reference to a collection of KA (this is the GA message), and that it is looking for KA to aggregate (this is the GN message).
- Finally, a **message handler in the specific interface** could be in charge of providing an access to specific KA in the KC collection.

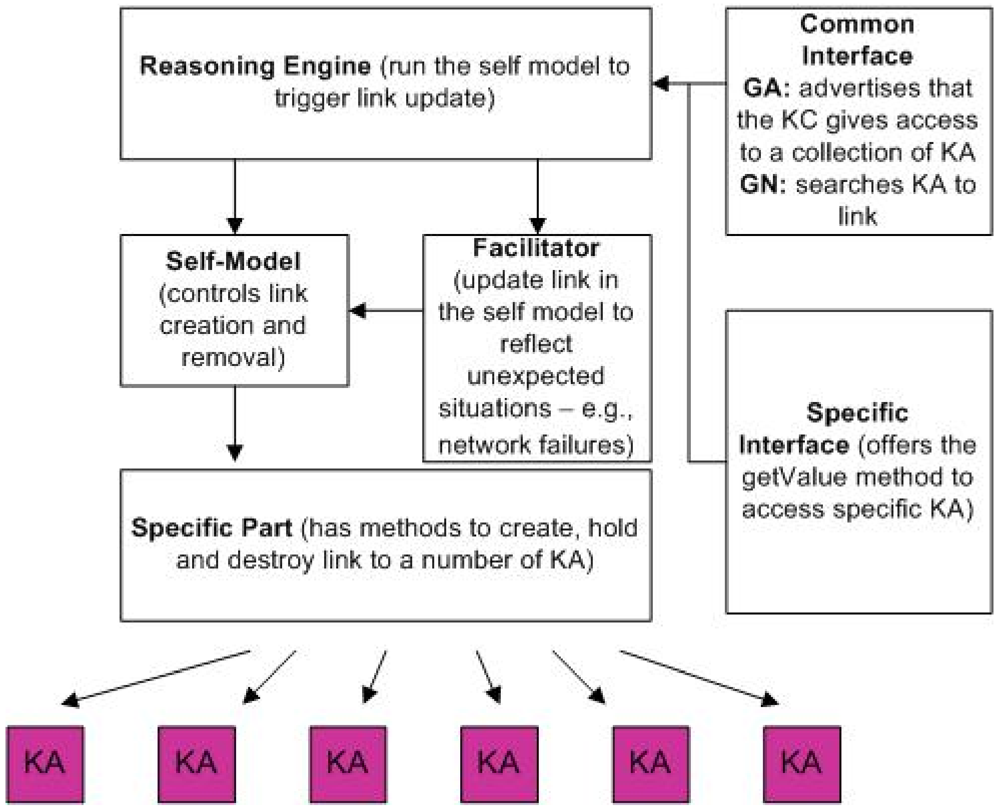

Reasoning Engine (run the self model to trigger link update)
Common Interface
GA: advertises that the KC gives access to a collection of KA
GN: searches KA to link
Self-Model (controls link creation and removal)
Facilitator (update link in the self model to reflect unexpected situations – e.g., network failures)
Specific Interface (offers the getValue method to access specific KA)
Specific Part (has methods to create, hold and destroy link to a number of KA)
KA
KA
KA
KA
KA
KA

Figure 23: knowledge container implemented in the ACE architecture.

4.7.2 How application ACE accesses the Knowledge Network?

Given that the knowledge atoms are implements as ACE, the access to the knowledge network is similar to invoking services to an ACE:

- An application ACE will express its need for knowledge by means of a GN message.
- Suitable knowledge atoms will answer with their GA message.
- The knowledge atom will also provide a reference in its specific interface to a specific message handler (e.g., get average)
- The application ACE sends a “getValue” message to the handler

- The knowledge atom responds with the value.

Naturally, ACE discovery services will be applied also to knowledge atoms to let "application" ACE retrieve suitable knowledge information. Similar consideration applies for an ACE trying to access a knowledge container.

In addition, in each ACE, its self-model can be considered as a knowledge atom, which represents its internal self-model. Such self-model, can become part of a knowledge network simply by having it registered into a KC (typically, the ACE will also act as a KC for itself, to enable its internal model to become part of a larger knowledge network). We emphasize that in our view, knowledge atoms are not static, but can contain dynamically varying information, which is in line with the ACE idea of having an internal facilitator in ACE's that continuously updates the self-model.

4.8 Checking Knowledge Network Specifications Against WP6 Requirements

During the specifications identification work, we have always kept into account the requirements identified in WP6. The result is that the above described specifications either already meet such requirement or have been conceived so as to make it possible to meet such requirements in next developments. Below, we copy the WP5 requirement table that can be found in D6.1, with an additional column shortly discussing how each of the requirements have been (or is to be) met.

WP	*Functional / Non Funct.*	*Scenario*	*Requirement*	*Answer*
R_5_1	F	*	Knowledge networks must support for a virtual view of environment to facilitate the concept of interest to adapt to changing conditions	Knowledge Atoms are sorts of virtual sensors that can provide a virtual view of the environment
R_5_2	F	*	There must be support for distribution of knowledge across a dynamic network	Knowledge networks can be distributed, and mechanisms for distributed knowledge management are being studied
R_5_3	F	*	There must be support for self-similar knowledge aggregation and for access to knowledge at different granularity levels	Knowledge containers can promote self-similar aggregation of knowledge and perception at different granularity levels
R_5_4	F	SH/UM	There must be support to represent and manage knowledge related to the user and the social level (user and social context profiling).	Knowledge atoms, as virtual sensor, can embody any kind of knowledge, there included user and social context.
R_5_5	F	*	There must be support to represent and manage knowledge related to the ACE level (profiling of ACEs and of their dynamic and aggregated status)	ACEs can expose their self-model as a knowledge atoms, to that knowledge network can include knowledge about ACEs.
R_5_6	F	*	There must be support to manage in an integrated (cross-layer) way user-level, ACE-level, and network-level, knowledge.	With the concept of knowledge atom as virtual sensors, various knowledge atoms managing different type of knowledge can be integrated with each other.
R_5_7	F	*	There must be support for construction and management of aggregated distributed knowledge.	Knowledge atoms and knowledge container can be use to build, via proper algorithms, aggregation of distributed knowledge.
R_5_8	F	*	To reach a high level of integration, knowledge networks should provide standard semantic mechanisms to organize and compose heterogeneous information and heterogeneous services.	Although ontology does not enter the specifications defined so far, knowledge atoms and knowledge containers can be represented according to specific ontologies to enable semantic knowledge management.

WP	*Functional / Non Funct.*	*Scenario*	*Requirement*	*Answer*
R_5_9	NF	*	To achieve a proper reconfiguration meaningful context information are to collected with high time granularity.	Knowledge atoms are connected with data sources and can always provide up-to-date information.
R_5_10	NF	*	Knowledge networks should provide also for producing and organizing new knowledge, inferred from existing one (e.g., for the sake of prediction).	Proper algorithms for knowledge management can be used to have new knowledge containers that represents somewhat new knowledge inferred from the information of contained knowledge atoms.
R_5_11	F	*	It may be necessary to protect selected sensible parts of knowledge networks from attacks (or, which is the same, knowledge networks must be able to exploit the security services of WP4).	Being possible to implement knowledge network components in terms of ACEs, whatever security solutions will be envisioned for ACEs this will be also immediately applicable to knowledge network components.
R_5_12	F	SH/UM	There must be support for spatial knowledge and spatial representation of situations.	The concept of location is explicitly part of knowledge network components.
R_5_13	F	WA	There must be support for semantic knowledge and shared ontologies to facilitate interoperability	See answer to R_5_8.

5 Mechanisms and Applications of Knowledge Networks

In this section, we report on experiments being performed to test two specific mechanisms that promise to be of general use for the building of self-organized knowledge networks (namely, field-based overlays and self-organized region aggregation), and then sketches several use cases in which knowledge networks can be fruitfully applied both as a support to situation-awareness and autonomicity.

5.1 Self-Maintaining Overlay Data Structures as Knowledge Networks

A general mechanism we studied and experimented with to support the activities of knowledge networks is based on overlay field-based data structures. These overlays are distributed data structures encoding specific aspects of the ACEs' operational environment. They can be propagated across a network as a sort of virtual force field, in order to represent and "communicate" context information.

From a modeling perspective an overlay data structure can be modeled as a knowledge container (KC) collecting a number of knowledge atoms (KA) that constitute the distributed elements of the overly.

The strength of these overlay data structures is that they can be accessed piecewise as the ACEs visit different places of the distributed environment. This lets the ACEs access the right information at the right location. ACEs interacting and perceiving their operational environment by means of these knowledge networks can disregard the underlying physical network and its dynamics (see Figure 24).

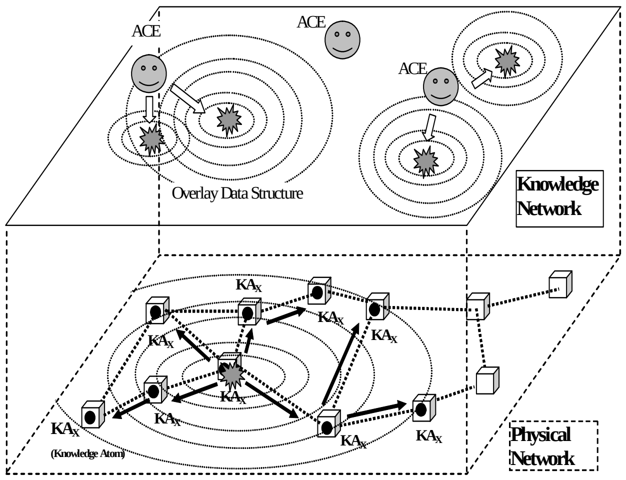


Figure 24: ACEs perceive a number of overlay data structures describing their environment. Each overlay is composed by a number of knowledge atoms.

In addition, overlay data structures enable sorts of "stigmergic" interactions [BonDT99] in that ACEs' interactions can be mediated by these kinds of overlay "markers" distributed across the environment. From another perspective, overlay data structures generalize the idea of overlay networks. Overlay networks are basically routing distributed data structures providing ACEs with a suitable application-specific view of the network (i.e. they allow ACEs to perceive a specific overlay topology of the network) [Rat01]. Overlay data structures do not focus on network topology only. They are general-purpose and can encode any kind of context information, thus they are a perfect match for knowledge networks.

To clarify this idea let us focus on the problem of coordinating the movements of some application ACEs in a distributed environment [MamZ06]. In particular, we focus on the simple application of having two persons, provided with a PDA, moving across an environment instrumented with an ad-hoc network infrastructure. The goal of the application is to allow one person to be guided by the PDA, to follow the other person. A simple solution based on overlay data structures is to let the person to-be-followed to spread in the environment (i.e., ad-hoc network) a data structure that increases an integer value by one at every hop as it gets farther from the source. This creates a sort of gradient that can be followed downhill by the other person to complete the application (see Figure 25(a)). If the person to-be-followed moves, it is important that the overlay data structure adjust its shape accordingly, so that the gradient leads to that person anyway (see Figure 25(b)). The power of this approach is that the knowledge network provides expressive contextual information tailored for that specific task. The ACE running on the PDA does not need to have any map of the environment, nor does it have to execute complex algorithms to decide where to go. It just blindly follows the overlay data structure.

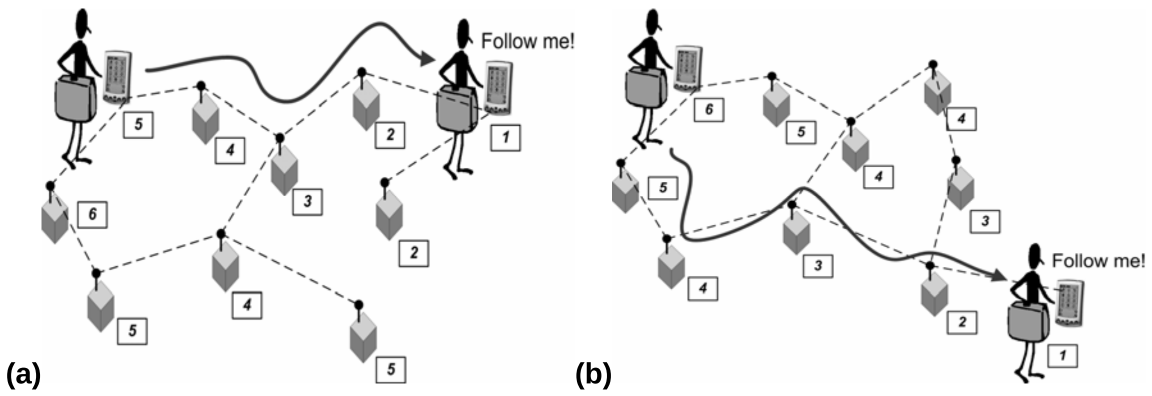


Figure 25: (a) gradient overlay data structure enables an ACE to follow another one. (b) The data structure is updated to reflect the new ACE position.

Beside this exemplary application, overlay data structures can be applied in a wide range of application scenarios, ranging from robotics to network routing:

Motion Coordination. As already stated in the previous example, overlay data structures, spread across a properly networked environment, have been used in [MamZ06] for the sake of enabling ACEs (e.g. users carrying a PDA, robots, cars) to coordinate their respective movements. The goals of ACEs' coordinated movements can be various: letting them to meet somewhere, distribute themselves accordingly to specific spatial patterns, or simply move in the environment without interfering with each other and avoiding the emergence of traffic jams. As previously stated, overlay data structures provide suitable tools for this task, in fact, they can be accessed piecewise to guide ACEs motion step-by-step.

Routing in a Mobile Ad-Hoc Network. Routing can be easily modeled as a coordination problem: ACEs (i.e. network nodes) need to cooperate forwarding each other messages to enable long-range, multi-hop communication. The main principle underlying many routing algorithms is to build several overlay data structures (implemented by means of a set of distributed routing tables) suitable to provide route information. Specifically, these data structures create paths in the network enabling ACEs to forward messages in the right direction. These paths (i.e. data structures) are maintained to take into account changes in the network topology [Poo01]. The idea at the basis of distributed routing data structure is the same as motion coordination: provide ACEs with a ready-to-use representation of the context (i.e. where the message should go next).

Swarm Intelligence. From a general perspective, overlay data structures are at the core of a number of swarm-intelligent (e.g. ant-inspired) systems [BonDT99]. These approaches mimic the way in which social insects, like ants, coordinate their activities to achieve complex tasks (e.g. the mechanism used by ants to find food can be used in the context of computer networks to route packets or find relevant resources). The key to these approaches is in emulating the way in which ants interact with one another. They do so by means of pheromone signals they spread in the environment that will be perceived by other ants later on. These pheromone signals can be used to find food sources, or to coordinate efforts in moving some heavy objects, etc. Pheromone signals can be easily modeled by means of overlay data structures. Overlay data structures implementing the concept of pheromone could be distributed by the ACEs themselves as they move across the network. These data structures can then be used as trails driving ACEs' activities. For example, the research projects Anthill [BabM02] and SwarmLinda [MenT03] share the idea of applying ant-inspired algorithms to Internet-scale Peer-to-Peer systems. Here, overlay data structures - modeling ants' pheromones - create paths connecting peers that share similar files, thus enabling, for example, an effective content-based navigation in the network of peers.

Amorphous computer. Overlay data structures are at the core of ~~the~~ amorphous computer [Nag02] research. An amorphous computer consists of massive numbers of identically-programmed and locally-interacting computing nodes, embedded in space. Overlay data structures can be spread and deployed in the amorphous computer to let various patterns and shapes emerge among the computational particles. Just to mention few trivial examples, if a leader particle spreads a hop-increasing overlay data structure (as defined above), it is possible to create approximately circular regions of controlled size: particles sensing the overlay are able to determine if they are in or out a specific circular region of radius R (i.e. they are in if they sense the data structure with a value lower than R). Similarly, if a line of particles propagate the above data structure, stripes instead of circles can be identified in the amorphous computer.

Modular Robotics. A modular or self-reconfigurable robot is a collection of simple autonomous actuators with few degrees of freedom connected with each other. A distributed control algorithm is executed by all the actuators to let the robot assume a global coherent shape or a global coherent motion pattern (i.e. gait). Some proposed approaches adopt overlay data structures to control such a robot [SheS02]. A distributed shape or motion gait is encoded by means of overlay data structures spread across the robot specifying how the robot's actuators should bend. Robots are programmed to bend their actuators depending on the sensed data, thus realizing the prescribed motion gait.

The fact that overly data structures have been successfully employed in all these scenarios motivate their adoption in knowledge networks to serve as a general mechanisms to encode context information and make it available to ACEs.

5.1.1 Modeling Overlay Knowledge Networks and Their Self-Maintenance Algorithm

From a modeling perspective an overlay data structure can be modeled as a knowledge container (KC) collecting a number of knowledge atoms (KA) that constitute the distributed elements of the overlay. In particular, the KC expressing overlay data structures can be defined by means of a couple *(C,P)*. The content *C* can be an arbitrary data structure representing the information carried on by the knowledge atom. The propagation rule *P* determines how the overlay data structure should be distributed and propagated across the network. This includes determining the ``scope'' of the overlay (i.e. the distance at which it should be propagated and possibly the spatial direction of propagation) and how such propagation can be affected by the presence or the absence of other data structures in the system.

The conceptual links between the knowledge atoms (expressed by the propagation rule) represent an example of horizontal knowledge organization.

In addition, the propagation rules can determine how the content *C* should change while it is propagated. Overlay data structure are not necessarily distributed replicas: by assuming different values in different nodes, they can be effectively used to shape a structure expressing some kind of contextual and spatial information. In addition, a maintenance mechanism should be enforced to let the overlay data structure preserve its intended distribution *(C,P)* despite network contingencies.

The idea of overlay data structures can potentially be implemented on any distributed infrastructure providing basic support for data storing (to store data values), communication mechanisms (to propagate overlay data structures) and event-notification mechanisms (to update overlay data structures and notify ACEs about changes in overlay data structures' values).

In most of the application scenarios described in the previous section, the main context information to be stored by an overlay is the hop-distance form the source. *Hop-based overlay data structures* are those having a content (C) and a propagation-rule (P) that depend only on the hop-distance from the source. Hop-based overlay data structures enable to express and diffuse across the network (possibly within a bounded scope) contextual information related to the network distance from the source. These kind of data structures have been widely used in motion coordination, routing and location-based-information-access applications. We designed the *Hop* data structure as a basic template to build this kind of overlays. In particular, the *Hop* data structure has a integer hop-counter (hop) as a content. Once one of these data structures is injected in the network, it propagates breadth-first maintaining the hop-distance from the source. These overlay data structures have to be maintained despite network topology changes either due nodes mobility or failures. The self-maintaining algorithm that will be described next performs exactly this task. The strength of these data structures, from a software engineering point of view, is that ACEs have simply to inject these data structures without further taking care of their update. All the burden in maintaining data structures is moved away form the ACEs.

Other than the modeling phase, we designed an algorithm to enable hop-based data structures to self-maintain their distribution despite network dynamism. To describe this algorithm, we will use the term *knowledge container KC* to refer to the whole distributed overlay. We will use the term *knowledge atom KA* to indicate a single piece of the overlay stored in a single node. For example a *Hop KC* is made of a number of *KA* each stored in a node of the network. Recall that a *Hop* KCs propagates increasing its integer content by one at every hop. Given a local KA of such a KC called 'X', we will call another KA 'Y' a *supporting KA* of 'X' if: 'Y' belongs to the same KC as 'X', 'Y' is one-hop distant from 'X', the value of 'Y' is equal to the value of 'X' minus one. With such a definition, a supporting KA of 'X' is a KA that could have created 'X' during its propagation. Moreover, we will say that 'X' is in a *safe-state* if it has at least a supporting KA, or if it is in the node that first injected the KC (i.e. hop value = 0). We will say that a KA is not in a safe-state if the above condition does not apply (i.e. it has not any supporting KA and it has a hop value greater that 0).

The basic idea is that a KA that is not in a safe-state should not be there, since no neighbor data could have created it.

Each local KA can subscribe to the arrival or the removal of other KA of its type (i.e., belonging to the same KC) in its one-hop neighborhood. Upon a removal, each KA reacts by checking if it is still in a safe-state. In the case a KA is not in a safe state, it erases itself from the local node. This eventually causes a cascading deletion of KA until a safe-state KA can be found, or the source is eventually reached, or all the KA in that connected sub-network are deleted.

In the case a KA is in a safe-state, the removal of neighbor KA triggers a reaction in which the KA propagates to that node. It is worth noticing that this mechanism is the same as when a new node is connected to the network. Similar considerations apply with regard to KA arrival: when a KA senses the arrival of a KA having a value higher than its own plus one, it means that, because of topology changes, a short-cut leading to the source has been created. In such a situation the KA can propagate to the new node to overwrite the previous KA, fixing the KC shape. This set of mechanisms is enough to make *Hop* KC self-maintain.

5.1.2 Experiments

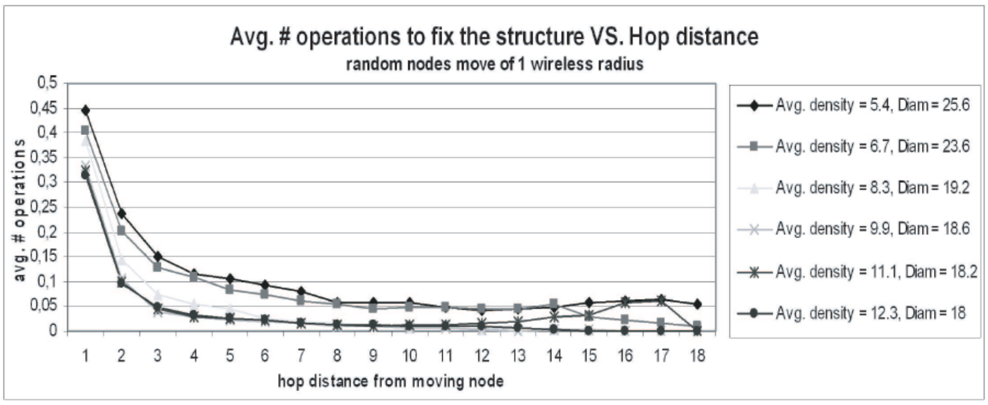
The effectiveness of our approach is of course related to costs and performance in managing overlay distributed data structures.

The cost of propagating a data structure, relying on a multi-hop mechanism, is something inherently scalable. Each node will have to propagate the data structure only to its immediate neighbors. The size of the network does not matter since the global effort to spread the data structure is fairly partitioned between the constituting nodes.

The scalability of data structures maintenance is less clear. The main requirement for our algorithms is to be independent of the network size. This implies maintenance operations must be confined within a locality from where events that altered the data structure (e.g., a network topology change) happened. If it is so, concurrent events at distant points of the network do not accumulate locally. If, on the contrary, maintenance operations always spread across the whole network, distant concurrent events do accumulate and the system does not scale.

With regard to *Hop* data structures, establishing if maintenance operations are confined to an area neighboring the place in which the network topology had actually changed is rather complicated. The size of this neighborhood is not fixed and cannot be predicted a-priori, since it depends on the network topology.

(a)



(b)

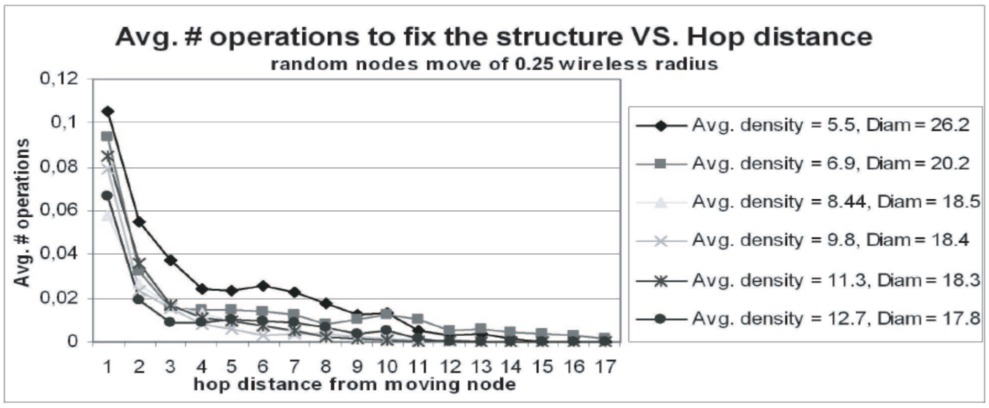


Figure 26: The number of maintenance operation decreases sharply with the hop distance from topology reconfigurations caused by: (a) random node movements for 1 wireless radius. (b) random node movements for 1/4 wireless radius.

Thus, trying to answer, we exploited a knowledge network simulator developed within our group over Repast (see also Section 6), and performed a large number of experiments to measure the scope of maintenance operations. To perform the experiments, we run several simulations varying the node density and their initial position. In particular, we run six sets of experiments where we randomly deployed 200, 250, 300, 350, 400, 450 nodes in the same area; thus obtaining an increasing node density and a shrinking network diameter. All the experiments were repeated a large number (over 100) of times with different initial network topologies and the results were averaged together. The experiment consisted in a randomly chosen node injecting a *Hop* data structure in the network. After that, randomly chosen nodes start moving independently (following a random waypoint motion pattern) perturbing the network. In particular, a randomly picked node moves randomly for a distance equals to 1 wireless radius. This movement changes the network topology by creating and disrupting links. The number of messages sent between nodes to adjust the data structure, according to the new topology, is recorded. Specifically, we evaluate the average number of messages exchanged by nodes located at x-hop away from the moving node. Then, we average these numbers over a large set of topology changes. The results of this experiment are in Figure 26(a).

The experiments reported in Figure 26(b) have been conducted in the same manner. This time, however, nodes move for a distance of 1/4 wireless radius. This second set of experiments is intended to show what happens for very little topology reconfigurations (wider reconfigurations can be depicted as a chain of these smaller ones).

The most important consideration we can make looking at the figure is that, when a node moves and the network topology changes consequently, a lot of update operations will be required near the area where the topology changes, while only few operations will be required far away from it. This implies that, even if the network and the data structures being propagated have no artificial boundaries, the operations to keep their shape consistent are strictly confined within a locality scope. This result is even more significant if compared to the average network diameter (averaged over the various experiments). It is easy, in fact, to see that the number of operations required to maintain a data structure falls close to zero well before the average diameter of the network, thus confirming the quality of our results. This fact supports the idea that the operations to fix distant concurrent topology changes do not add up, making the system scalable.

In conclusion of this section, we foster the idea that overlay data structures are a powerful mechanism to support knowledge networks. In fact, they both enable the expression and retrieval of context information in a flexible and distributed way, and they can scale to large scenarios with a lot of ACEs being supported by knowledge networks.

5.2 Self-organized Region-based Knowledge Aggregation

An additional mechanism for knowledge networks concern distributed knowledge aggregation in large scale ad-hoc networks, i.e., sensor networks.

In the next few years, we will assist to an increasing presence of sensors in our environments. There will be sensors deployed in our cities, in the countryside and possibly even in open waters for monitoring marine life. Probably, such mass deployment of sensor network systems will induce a radical change in their usage. Rather than being closed special-purpose systems devote to specific phenomena, as they are today, they will form the basis of truly pervasive shared sensing infrastructure, publicly available for general-purpose sensing activities by a variety of users [MulA06, Cur05]. In general, such pervasive sensing infrastructure can be of help to human, mechanical, or digital "users" to achieve higher degrees of perception and context-awareness. The need to effectively gather environmental information in a compact and energy efficient way, also by mobile users other than by fixed sinks, calls for algorithms enabling in-network aggregation of data and identification of relevant patterns. Knowledge networks aim to fill the gap between the physical world and user level services providing an efficient way to collect and provide contextual knowledge. To reach the goal they need mechanism and algorithms to collect, organize and infer existing and new knowledge.

One of the most basic primitives needed to organize data is aggregation. Using it we can select an item or a group of items from a set belonging to a common property. Aggregation is a very general primitive and can be used to aggregate sensor with similar readings but also persons, services, devices showing a common pattern. We can select for example all the persons dressing a red hat in the university campus or all the routers deployed in the second floor which are not busy since two hours ago. In general we can consider everything in the physical world as a sort of sensor exposing a particular set of values. Starting from these considerations, we have developed and simulated an aggregation algorithm suitable for building and maintaining knowledge networks. In particular it can successfully realize a form of either horizontal and vertical aggregation. Over an existing environment fulfilled of sensing ACE's exposing the KA interface we can inject this algorithm to build links between logically correlated KAs. Moreover we can exploit the formation of such logically correlated regions to compute, at no additional costs and on a per-region basis, other aggregated data.

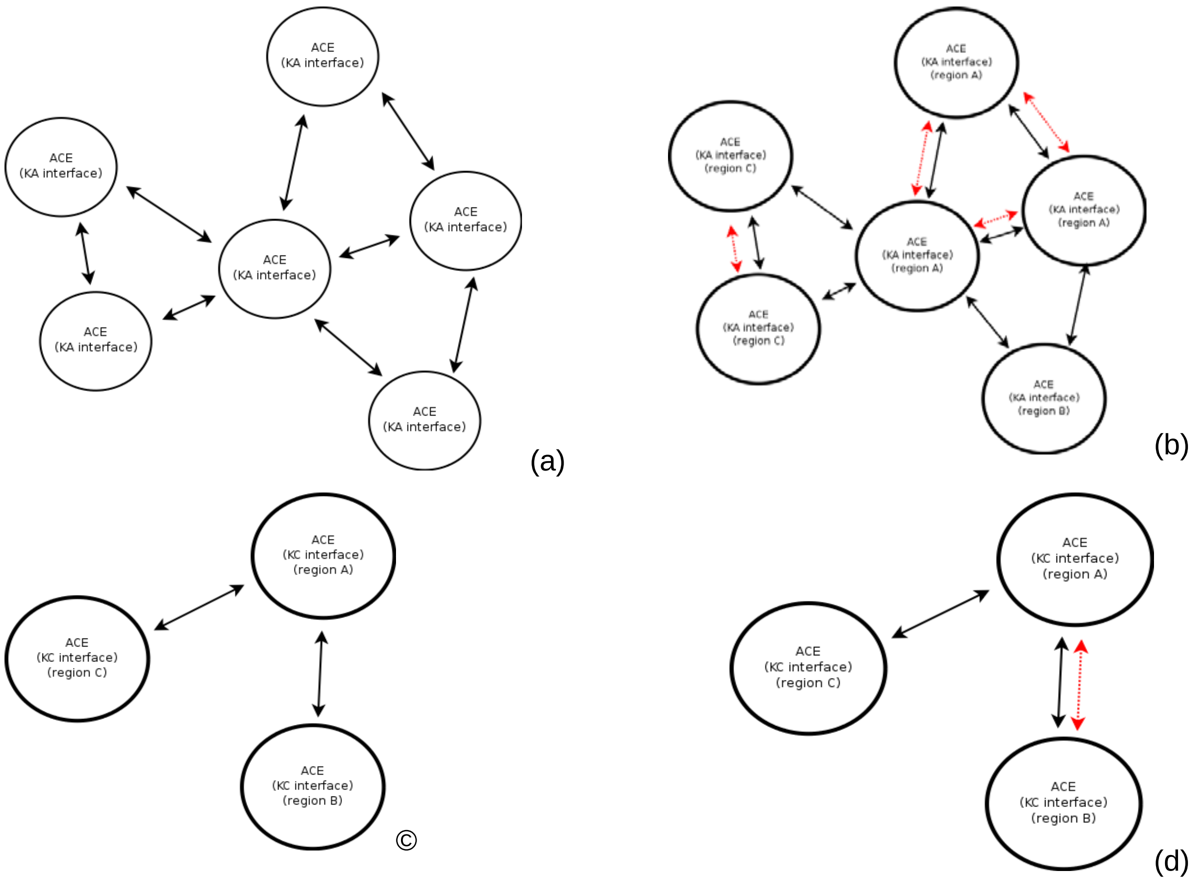

ACE (KA interface)
(a)
ACE (KA interface) (region A)
ACE (KA interface) (region C)
ACE (KA interface) (region B)
(b)
ACE (KC interface) (region A)
ACE (KC interface) (region C)
ACE (KC interface) (region B)
©
(d)

Figure 27: Individual Stages of Region Self-Organisation.

It is interesting to note the subtle difference between the different ways by which horizontal and vertical aggregation store knowledge. The first one just establishes relationships between different KAs, building a sort of knowledge overlay in which related concepts or values are linked. The second one uses data coming from different KCs to compute and obtain a more structured vision of the environment. Figure 27 clearly highlights the differences between the two approaches. In Figure 27(a-b) different ACE's exposing KA interface are represented with their physical channels used to communicate each other. Horizontal aggregation build a virtual overlay of links between related concepts, every ACE continues to expose only the KA interface but can make available to the outside data related not strictly to its values but also to the whole region in which it is included. Now, we can imagine ACE's belonging to the same region to be linked to a common KC. In Figure 27(c-d), the algorithm is used at the KC level. After the vertical aggregation process has been executed, different KC are organized in a more structured way. For example regions A and B are recognized to be related. In this example we show a very distributed application of knowledge networks concepts, but it is important to have in mind that the described mechanisms should be suitable also for a simple and centralized implementation based, for example, on relational databases.

5.2.1 The Region Aggregation Noise Algorithm

As described before, our algorithm is divided into two different blocks, one performing horizontal aggregation and the other the vertical one:

- A distributed algorithm is continuously running in the knowledge network as a sort of "background noise" with the goal of partitioning the network into regions characterized by similar patterns for sensed data;
- The formation of such regions is then used to compute on a per-region basis, aggregations of sensed data, so that users and services accessing the network through the appropriate interface can be provided with such pre-computed aggregated data.

Basically, our algorithm work as follows. Consider a network composed by different devices that are able to communicate each other. Each device is executing an ACE and expose values through the KA interface. Let s_i and s_j be two ACE's exposing a KA. They can be considered neighbors if they able to communicate. Define the values into by s_i and s_j as $v(s_i)$ and $v(s_j)$, and let us assume that a generic distance function D can be defined for couples of *v* values (thus defining v as a metric space), i.e., $D(v(s_i), v(s_j))$. Region formation is then based on interatively computing the value of the logical link $l(s_i,s_j)$ for each and every ACE of the system, as in the following "*Update_link*" procedure:

```
Update_link:
if $D(v(s_i), v(s_j)) < T$ {
        $l(s_i,s_j) = min(l(s_i,s_j) + delta, 1)$
} else {
        $l(s_i,s_j) = max(l(s_i,s_j) - delta, 0)$
}
```

Where: *T* is a threshold that determines whether the measured values are close enough for $l(s_i,s_j)$ to be re-enforced or, otherwise, weakened, and *delta* is a value affecting the reactiveness of the algorithm in updating link. What is already clear, though, is that after some iterations, if the $D(v(s_i), v(s_j))$ is lower than threshold *T*, $l(s_i,s_j)$ will converge to *1* otherwise to *0*. In the simplest case, one could consider two nodes s_i and s_j to be in the same region when $l(s_i,s_j)$ is over a threshold T_h, However, to improve stability, we introduced a hysteretic cycle with two threshold T_l and T_h.

Concerning *T*, a challenging issue in our approach consists in tackling the difference between the strictly local nature of “*Update_link*” interactions and the inherently global meaning of the threshold *T*. How can two nodes evaluate which is the right threshold to establish if they are similar enough to be in same region or not if they don't know anything about the rest of the network ? For instance, a difference of 10°C in a wood can be considered relevant during normal days but irrelevant for the sake of fire detection. To deal with this problem avoiding the need for a priori information, we opted to define T by exploiting dynamically collected global values of the property *v*. In particular we define *T* as a portion of the whole range of values seen over the network. Using scalar values, we defined *T* as:

$$T = (globalMax - globalMin) * p$$

where *p* is a real number between *0* an *1*. By this way, one can parameterize the sensibility of the algorithm by using a relative value *p* rather than some absolute value requiring a priori knowledge on the whole range of *v* values. If one wants to obtain very large regions to organize the network based on macroscopic difference one can select *p* close to *1*. If one is interested in more fine-grained region organizations one can select *p* close to *0*. It is worth emphasizing that globalMax and globalMin are just two possible global values to be used to partition the environment into regions.

The distributed execution of the algorithm is based on a sort of gossip scheme [Bab05]: each ACE periodically wakes up, randomly selects a specific number (or a specific percentage) of its neighbors, exchange with them the needed data (i.e., the *v* values, plus other data that will be detailed in the following), and then execute the “*Update_link*” procedure for each of the selected neighbors:

```
Do_forerever:
    Wait(t);
    neigh[] = Select_neighbor(num_neigh);
    Foreach(neigh[])
        Data = Exchange_data();
        Update_link(data);
Done
```

By considering the situation in which regions are already formed, computing aggregation function in a region reduces to executing a gossip-based aggregation algorithm only between those couples of neighbor ACE’s that are in the same region (i.e., for which the *l* is over the T_h threshold). Again, computing per-region aggregation function does not introduce significant additional burden to the network. The exchange of data between nodes can occur by piggybacking over the existing messages, and the computation of local aggregation algorithms reduces to adding a simple “*Local_aggregation*” function in the main body of our basic scheme, as follows:

```
Do_forerever:
    Wait(t);
    neigh[] = Select_neighbor(num_neigh);
    Foreach(neigh[])
        Data = Exchange_data();
        Update_link(data);
        Global_aggregation();
        If(connected) Local_aggregation();
Done
```

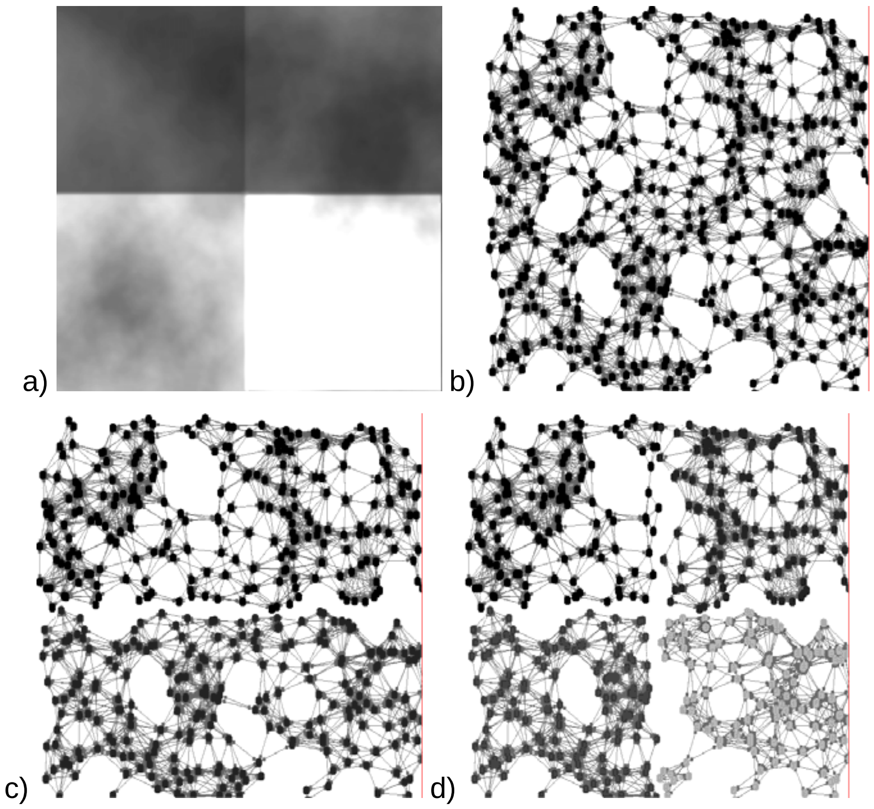


Figure 28: Examples of region self-partitioning. a) a sample scalar field with 4 regions with different values of a property v; b) a 500-nodes sensor network immersed in the above scalar field, with links representing the actual physical layer or, which is the same, the self-organization into a single global region (as it happens when p=1); c) example of the overlay region organization with p=0,4 leading to a partitioning into 2 “large” regions (we show only the logical links between the nodes that are logically connected); d) example of the overlay region organization with p=0,05 leading to a partitioning into 4 smaller regions,

5.2.2 Experiments

We have performed numerous experiments based on simulations using the adapted Repast framework (see Section 6). Our goal was twofold. First, we wanted to evaluate the effectiveness of the region detection algorithm. Second, we wanted to evaluate the convergence and accuracy level of the aggregation algorithms, and the trade-off between accuracy and energy consumption.

The results of the simulations were obtained by simulating scalar fields in which the sensor network is immersed similar to that of Figure 28-a. Though we have conducted several experiments on fields with different shapes and values, we have always obtained comparable results from both qualitative and quantitative viewpoint. Therefore, we report here on an environment filled with 500 wireless sensors disposed over a random graph such that the mean number of neighbors for each node is 15 (i.e., qualitatively assimilable to the sensor network of Figure 28-b). The simulated scalar field exhibits values *v* such that four different quadrants are recognizable (as in Figure 1-a). Each quadrant has a fixed mean *m* and variance *s.* Starting from the top left quadrant and proceeding clockwise, they could be identified as q_1, q_2, q_3, q_4. Mean values $m_1..m_4$ of *f* in $q_1..q_4$ are respectively 120, 80, 20, -20. Variances $s_1..s_4$ are arranged such that in each quadrant are allowed values *v* in range [*m* – 2, *m* + 2].

Network behaviour can be described from both a static and dynamic point of view. From the former we can analyze, independently from the speed of convergence, which are stable states reached by the network and evaluate the effects of related parameter *p*. From the latter we show the dynamic behavior of the network, the speed of convergence and the accuracy level depending on *num_neigh* and *t*.

Let us first focus on region detection.

From a static viewpoint, as described in Subsection 2.1 and as shown in Figure 1, variations on the parameter *p* induce the network in self-partition into regions of different sizes. The same behavior has been verified to apply for networks immersed in fields with different shapes and with different sizes, as in Figure 28(b-d)).

Let us now switch to the dynamic viewpoint and show how variations of the gossip percent *num_neigh* and the sleep cycle *t* affect the speed of convergence and the accuracy of the region detection algorithm. Let's consider a simulated a 500-nodes sensor network and a scalar field similar to that of Figure 28. Initially all nodes are not connected with any neighbor. We collect data over the first *255* cycles. Within cycles from *0* to *128 p* is set to *0.4*. During this interval the network converge to a status similar to that of Figure 1-c, i.e., splitting the network into regions. At cycle *129*, we changed *p* from *0.4* to *1.0* , making the network re-compact into a single region (as in Figure 1-b).

In Figure 29-a we show the evolution in the average number of nodes per region as time passes, by varying the gossip percentage. Figure 3-b shows the same kind of evolution but by varying the sleep period *t* of sensor nodes. Values are collected at the completion of each simulation cycle. Both the graph show that the number of nodes of the region start from *0*, grow to *250* during the first phase *[0 – 128 cycles]* and than reaches 500 during the second phase *[129 – 255 cycles]*. Clearly, reducing the gossip percentage or increasing the sleep period *t* make the network slower in the region detection process.

From Figure 29, it also emerges that the speed of the network is less influenced by variations of *num_neigh* than by variations of *t*.

The strange "stairs-like" trend of data lines obtained by setting *t=4* and *t=8* (Figure 29-b) clearly show the non-linear nature of the algorithm. These are mostly due to the fact that, when a region is forming, lots of sub regions are growing within it connecting the most similar neighbors. Only when the new actual minimum ID of the new region reaches a node, such node recognize it is becoming part of a new region.

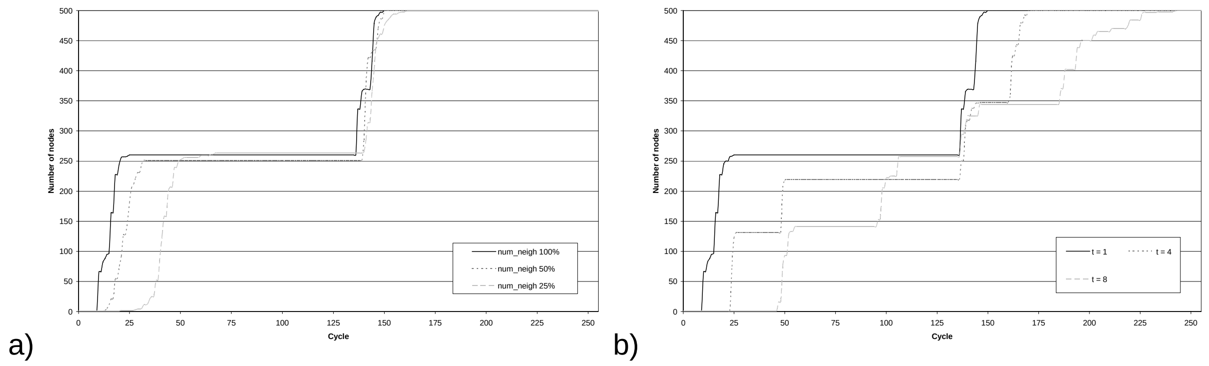


Figure 29: Evolution of region detection. a) *t = 1. num_neigh = 1*, *num_neigh = 0.5*, *num_neigh = 0.25*; b) *num_neigh = 1. t = 1*, *t = 4*, *t = 8*.

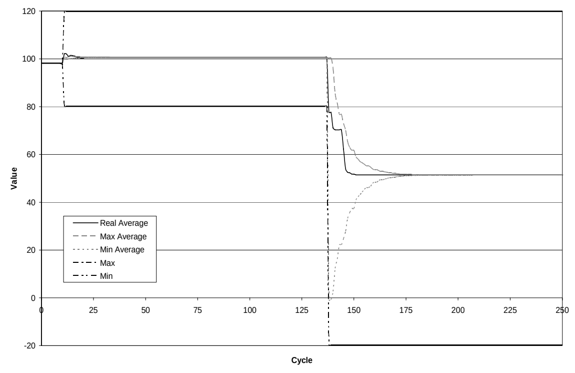
Let us know focus on the behavior of the region aggregation noise approach in evaluating aggregated values.

From the static viewpoint, all local aggregation algorithms experiences correctly reach convergence towards the corrent (real) value.

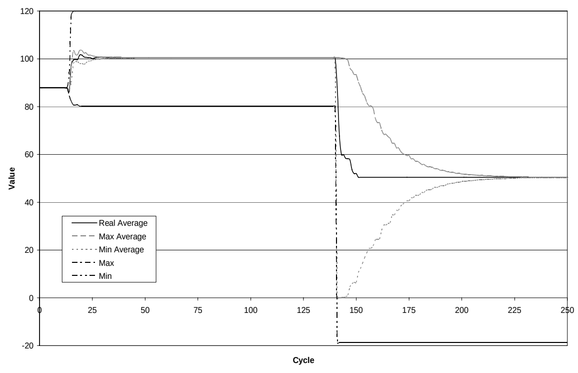
From the dynamic viewpoint, Figure 30 shows the trend of several values aggregated on a per region basis. Curves in each graph represent the minimum (worst case) estimate of the region maximum, the maximum (worst case) estimate of the region minimum, the minimum and the maximum (the two worst cases) estimates of the average, and the real actual value of the average computed over all nodes within the growing region. Figure 30-a show results obtained with *num_neigh=1.0* and *t* = 1. Figure 30-b and 30-c show results obtained reducing *num_neigh* to *0.5* and increasing *t* to *4*, respectively, Clearly, reducing the gossip percentage or increasing the sleep period *t* make the network slower not only in region detection but also in correctly evaluating locally aggregated values.

All the graphs in Figure 30 show the same trend. During the first cycles while links are being reinforced, all the aggregated values don't change. At the beginning (cycle *0*), when the region starts forming is clearly visible a fast convergence of the local maximum and minimum to their new values respectively of *120* and *80*. Average related values have a relatively small transitory and eventually reach the value of *100* as expected. At cycle 128, *p* is changed to *p* = 1.0 and the region starts growing another time. The local maximum does not have to change its value. The local minimum reaches quickly its new value (*-20*) in a few iterations. Average values instead have a longer transitory but eventually slowly converge to the expected value of *50*. Observing Figure 30 it is clear that different aggregate values behave differently varying *num_neigh* and *t*. In particular accuracy of average related values are really more sensible to variations of *num_neigh* and *t* than the local minimum and maximum have.

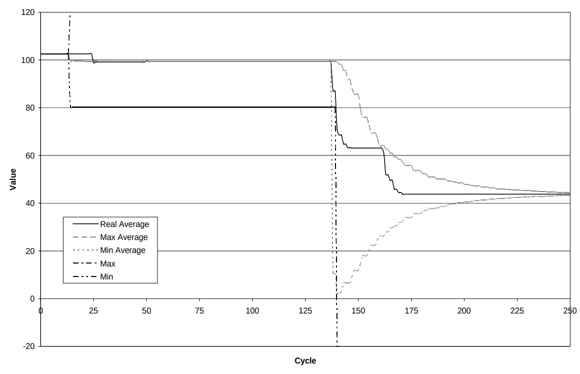
To summarize this, there is a clear trade-off between energy consumption and accuracy: higher *num_neigh* and the lower *t* clearly provides for more accuracy over time, but overall increase the energy consumed. Due to the high convergence speed of *Max* e *Min* showed under all conditions tested and to the fact that regions are expected to have relatively limited size, scalability of the region aggregation noise approach should not be a major issue. We tested it with sensor networks up to 10000 nodes obtaining similar results.



a)



b)



c)

Figure 30: Per region aggregated values. Minimum estimate of the maximum, maximum estimate of the minimum, minimum and maximum estimates of the average and real value of the average. a) *num_neigh = 1.0*, *t* = 1; b) *num_neigh* = 0.5, *t* = 1; c) *num_neigh = 1.0*, *t* = 4.

5.3 Application Use Cases

Other than having performed the above described studies, we have carefully evaluated via pencil and paper design exercise various possible usages of knowledge networks. We shortly overview here some of this performed studied.

5.3.1 Living Diaries and Social Serendipity

The living diary is a knowledge-network centric application, which aims at exploiting the pervasive devices embedded in an environment to produce a sort of digital self-composing diary in the form of a knowledge network. When a user moves in an environment (as in an exhibition) with a PDA and enriched with peripherals to access embedded devices (e.g., sensors and RFID tags) and to produce situational information (e.g., a GPS), this can generate a lot of elementary information about the context (i.e., knowledge atoms). On this base, a specific service (i.e., an ACE or a knowledge container devoted to manage such atoms), can be in charge of collecting all contextual information that is gathered from the environment, and organize such information into a knowledge network that reports in an organized way all the facts, and events (e.g., people met and objects encountered) having occurred (see Figure 31). The collection of knowledge atoms describing the situations a user is in and has been – stored in PDAs or portable devices and possibly downloaded to some knowledge network repository on need – acts as an historical memory for the user. A user, on need, can then exploit a specific ACE to browse his living diary. As another example, in an exhibition center, it is possible to envisage some exhibition-specific ACE that, by browsing in the past the living diary of a user to detect detailed preferences and habit, can act as a personalized guide to the exhibition.

As simple as it can be, the living diary shows very clearly how it may be useful to link pieces of knowledge together so that they can eventually represent a complex situation (in this case a complex history) and so as to make it possible to “navigate” the resulting knowledge network to make services become aware of situations.

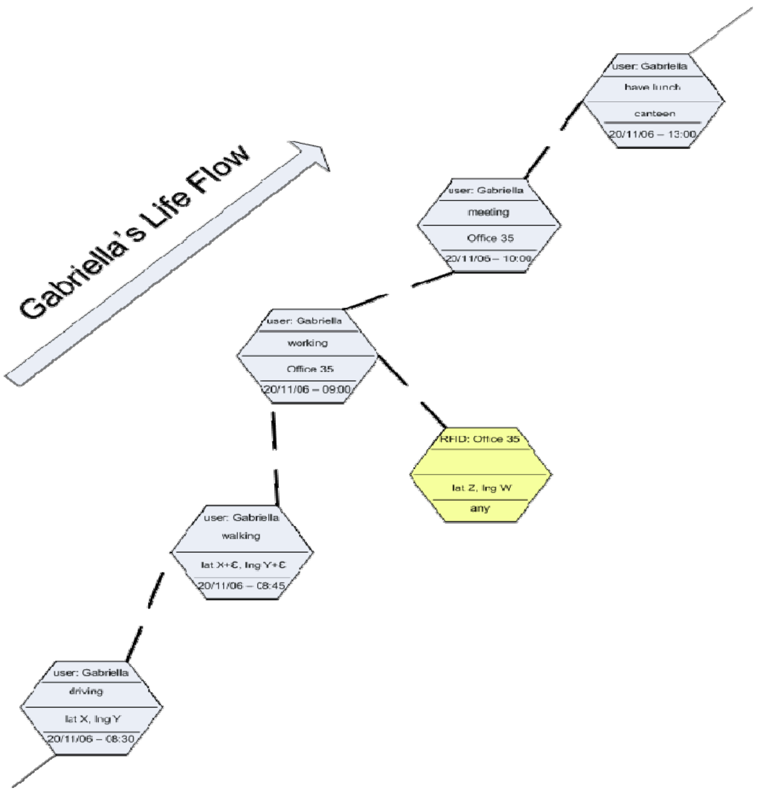


Figure 31: Example of Gabriella’s living diary in the form of a knowledge network.

To put the living diary example forward, imagine an exhibition scenario where there is a sort of open place for people to meet or simply to rest (a cafeteria for example). We assume that such cafeteria is enriched with pervasive and wireless devices as the rest of the exhibition. When a user enters the cafeteria, some devices (e.g, an RFID reader) can recognize that such user has entered, so that the cafeteria is always aware of who’s in it. At this point, a user can decide to share with other users (either directly in an ad-hoc way or by mediation of some server of the cafeteria) his own personal profile, or even his living diary.

By sharing (portions of) the knowledge networks representing the living diaries of the persons in the cafeteria, and by allowing group of ACEs to navigate such diaries, one can think of discovering relations between persons, common interests, or common past events (see Figure 32). As a trivial example, ACEs can identify that two persons have just attended the same show in the exhibition, and can decide to signal this fact to them so as to promote socialization and exchange of experiences. More generally, ACEs can cooperatively explore the past life of users to discover facts and social affinities that users’ could have never discovered otherwise. In the example of Figure 32, ACEs could discover that Gabriella and Franco already met in the past during a meeting. This can be classified an a service supporting people-to-people coordination.

Eventually, ACEs devoted to analyse, merge, and synthezise multiple living diaries, can extract useful synthetic information about the classes of persons that are currently in the cafeteria and, say, exploit such information for commercial (i.e., advertisement) reasons.

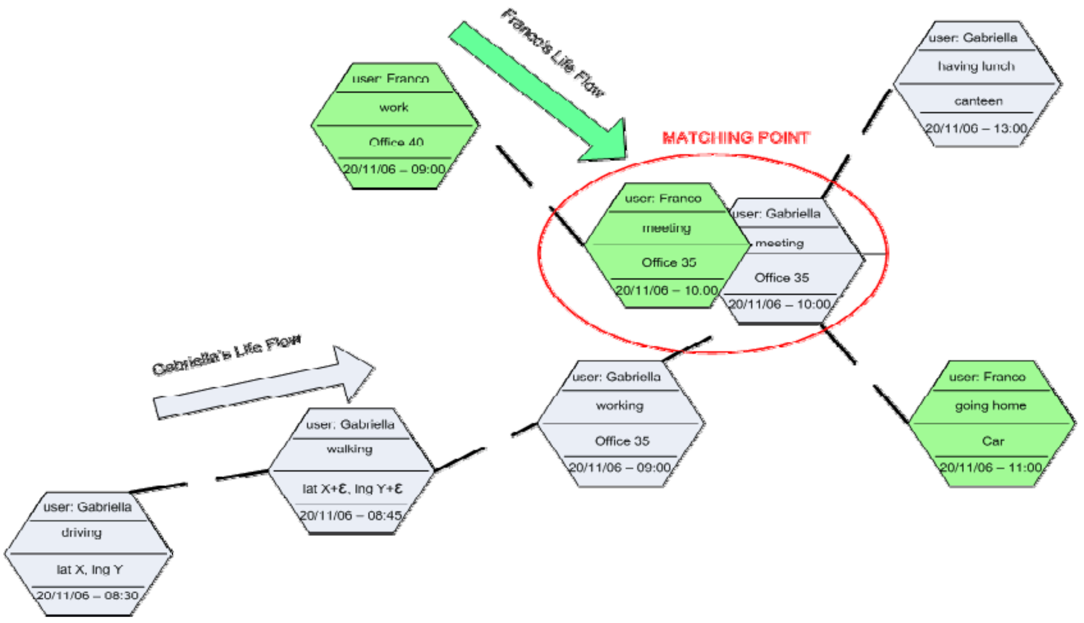


Figure 32: Social serendipity at the Blogcafé by navigating and merging the knowledge networks (i.e., the living Diary) of different people.

5.3.2 Overlay Field-based Knowledge Networks for Supporting Independent Living

Consider a person, suffering from mild dementia, who has a problem in orienting himself even at home. Let us suppose such person lives in a smart house, with an embedded pervasive computing infrastructure made up of several sensing nodes and computational capabilities. On this base, such person can be supported via a simple mobile device (e.g., a PDA) to improve his spatial sense of the environment, i.e., by having the PDA giving him directions or suggesting him where objects and or persons can be found (see Figure 33). In other word, the idea is to exploit a pervasive computing infrastructure to improve the spatial-awareness of this person, which necessarily implies the associated software services to be spatial-aware.

With this regard, knowledge networks have to provide a virtual view of the environment they are operating in to allow the concept of interest to be properly represented from a spatial perspective. The concept of overlay field-based knowledge networks discussed in Section 5.1 perfectly fits this situation. In fact, we can assume that, in the distributed environment, an overlay knowledge network can be built such that:

- the presence of an object or of a person of interest in an environment is translated into an overlay field-based knowledge network that propagates in the environment (whether a house or a larger environment such as an exhibition center);
- self-maintenance algorithms provides for automatically updating the knowledge network as the person of interest moves or some object is moved;

The result is a sort of live knowledge network that, when queried by a user (better: when queried by some service component on the users’ PDA), can give directions on where the object or the person is (i.e., how far it is and in which direction).

Again, this example shows that situational (spatial in this case) information cannot be simply considered as a set of independent knowledge pieces available somewhere. Rather, for a satisfactory adaptive orchestration of distributed activities (whether this is intended to be the orchestrated configuration of individual components or the coordination of distributed service components), the exploitation of local knowledge only may not be enough. Nor can one think of concentrating in a single site or of replicating anywhere all available knowledge, especially when this knowledge represents dynamically evolving situations, i.e., it is subject to obsolescence. The compromise is to enable components which need more than simply local knowledge to organize and correlate distributed knowledge into sorts of networks that enable distributed components to “navigate” through the available knowledge to attain, on demand, the required degree of contextual awareness.

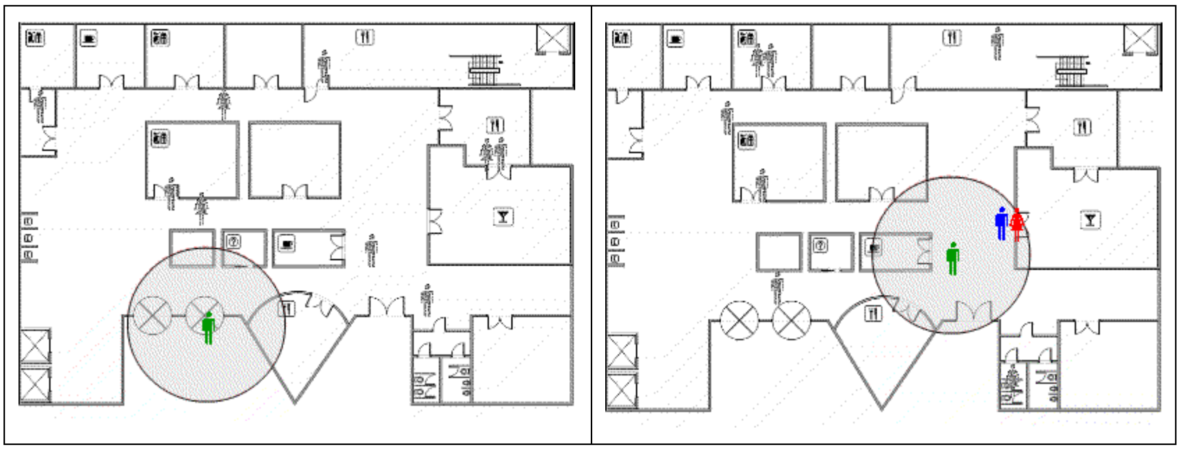


Figure 33: Supporting independent living in a smart home.

5.3.3 Examples of Batch vs. On-line organization of Knowledge Networks

The KN architecture developed so far provides two possibilities how to organise the location of knowledge within the knowledge network: batch and online (see also Section 4.6.1). Batch organisation means that data that is aggregated in a KC, is directly copied to this KC and it is typically stored at one place. Online organisation, in contrast, means that data in a KC is not necessarily stored in one and the same location. What is stored in an online organised KC are links to data sources, not the data itself.

Let us now sketch some simple examples we have developed to clarify the usages of batch vs. on-line organization.

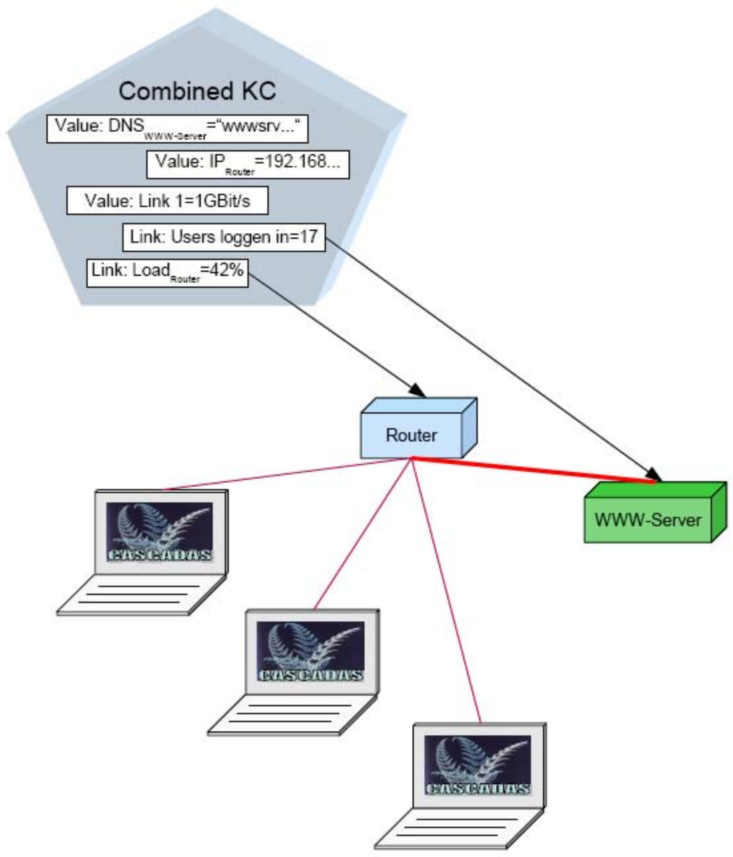

Combined KC
Value: DNS$_{WWW\text{-}Server}$="wwwsrv..."
Value: IP$_{Router}$=192.168...
Value: Link 1=1GBit/s
Link: Users loggen in=17
Link: Load$_{Router}$=42%
Router
WWW-Server

Figure 34: Combined KC, aggregating knowledge about a network.

A KC describing a local area network and its parameters is an example where a combined knowledge organisation structure is most suitable (Figure 34). Information concerning design parameters of the described network may be copied to the KC, i.e. batch organised, because they will rarely change. This information can be e.g. the addresses of routers and servers, the link capacity between network elements, DNS entries, and so on. Parameters whose values rely on sensors are preferably stored online, i.e. the network describing KC just contains links to the original data sources instead of the values themselves. So, current bandwidth consumption, usage rate of routers or the current usage of services, available in the monitored network, can be obtained in real-time.

Let us now consider an example in the area of traffic monitoring and regulation. In the scenario depicted below, the traffic situation in the city is monitored by road intersection counters installed at traffic lights. The traffic management system includes dynamic message signs, which informs car drivers about the current road situation and alternative routes. It can dynamically control the traffic lights as well as change the maximum allowed speed. As depicted above, intersection counters (car counters) are installed at the traffic lights which are controlling the traffic at the major road intersection points in the city. From the ACE model perspective, each counter can be seen as a knowledge atom (KA) which provides information (knowledge) about the amount of cars located at a particular road intersection point at a given time. In order to calculate the amount of cars located in a certain area at a particular time, the intersection counters need to exchange their knowledge about the number of cars at all involved traffic lights. The intersection counters 1, 2 and 3 exchange their traffic information among each other and build up horizontal knowledge organisation structures (Figure 35). In order to generate higher level knowledge and recognise or predict certain traffic situations in our scenario (i.e. traffic jam recognition or prediction), it is required to combine the knowledge about the number of cars from the intersection counters with other types of knowledge like for example the weather forecast, day of the week and the current time. In this scenario the system combines different types of low level knowledge in order to generate high level knowledge and builds up therefore a vertical knowledge organisation structure. By combining the high level knowledge with other types of knowledge (both low and high level) knowledge containers (KCs) containing more complex knowledge structures could be built. In the example depicted in Figure 35, the knowledge about traffic jams is combined with the city map in order to generate the recommendations for alternative routes.

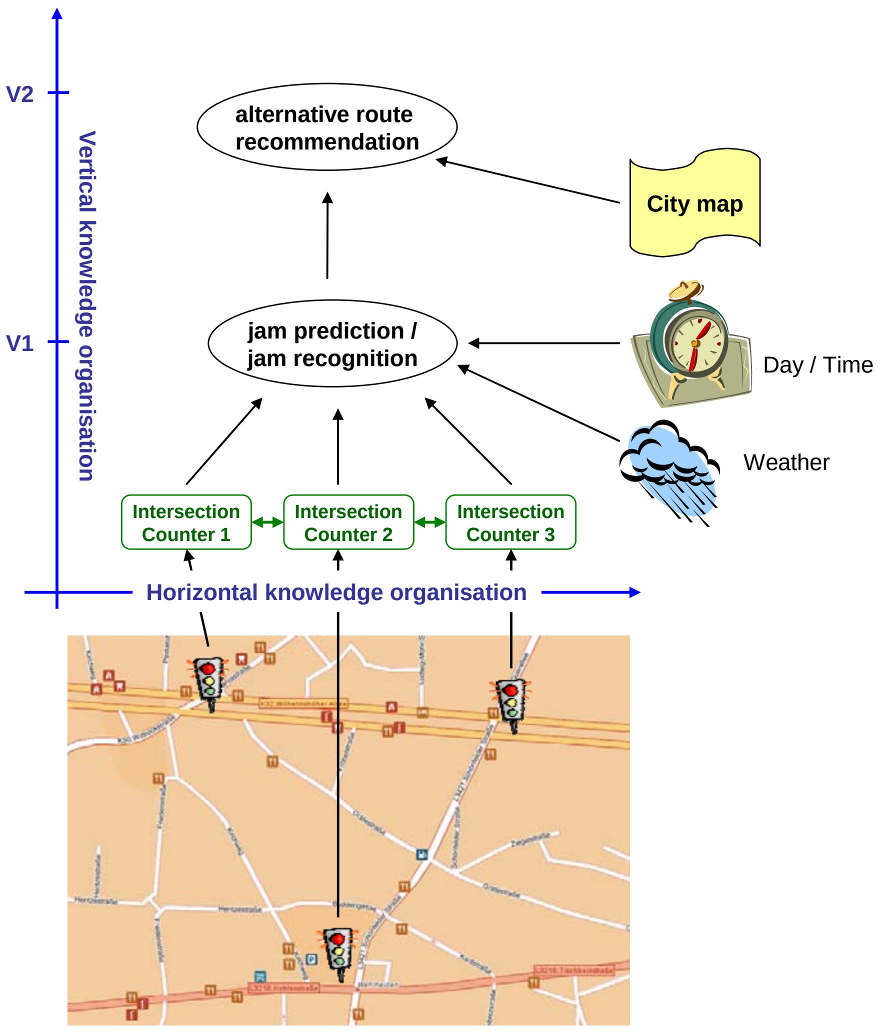

V2
V1
Vertical knowledge organisation
alternative route recommendation
City map
jam prediction / jam recognition
Day / Time
Weather
Intersection Counter 1
Intersection Counter 2
Intersection Counter 3
Horizontal knowledge organisation

Figure 35: Traffic situation monitoring scenario.

5.3.4 Knowledge Networks for ACEs Discovery and Interaction

The idea of Knowledge Networks can be successfully exploited to drive ACEs’ aggregation, i.e. by supporting ACEs in discovering other ACEs and interacting with them.

Clearly, it is possible to conceive that an ACE requesting the services of other ACEs manages on its own all the details about discovering other ACEs by comparing its GNs ("goal needed", see also D1.1) with the published GA ("goal achievable") and handling the interaction with all of them by invoking the right operations and passing well-formatted parameters to such operations

However, it is also possible to consider that some intelligence substrate in the system (realized by a knowledge network) can support ACEs in this activity. In this case, one can think of ACEs depositing their GNs in the knowledge network, and have this action induce "aggregation reactions" to occur in the knowledge network. In particular, some kind of "enzyme ACEs" embedded in the knowledge network and having the form of a knowledge container, can be made capable of reasoning about (maybe) little pieces of the knowledge network, and can in charge of manipulating the deposited GN in trying to fulfill the service request. Analyzing the GN (expressed semantically with knowledge networks atoms), they find and put together other ACEs, that can participate in achieving that GN. If the mentioned concepts are unknown to the enzyme ACEs, no reaction will occur and the request will decay.

TO exemplify this idea, let us briefly sketch a simple communication service, called Urgent Message Delivery (UMD). Let's imagine a user application that needs to deliver text messages (e.g. urgent communications) to persons within an organization. The self-aggregation reaction process starts with the user ACE injecting into the environment a service request, expressed as a GN message. An "enzyme ACE" catches the request and exploits its known concepts to find, for instance, a device where the person may be logged-in (PDA, GSM phone, laptop). If this is the case, it then attempts to deliver (possibly interacting with other sibling enzyme ACEs into the system) the message to that device. Therefore, the service aggregation will comprise, for instance, the "log-in manager service" and the "SMS phone service" (if a GSM phone is used).

If no "loggable device" can be exploited, it then seeks other ways to reach the person: for instance, using a location service (with RFID tags and wireless sensors), it detects that the person is currently in a certain room and that in such room there is a fax machine installed. The next possible aggregation strategy will likely put together the "localization service", the "fax sender" service and others.

The key principle of this knowledge-network-based service aggregation is that:

- services are dynamically found and linked together to fulfill a GN;
- such aggregation is performed by dedicated enzyme ACEs belonging to the knowledge network;
- self-adaptation and autonomic behaviours can be enforced into enzyme ACEs and not necessarily into every single ACE;
- services are aggregated in a situation-aware manner: if the situation changes (e.g. the GSM is no longer available), this is handled by reconfiguring the aggregation structure, replacing useless ACEs and finding new ones.

For better developing the above idea, which is a sort of "embryonic" example of cognitive stigmergy, we expect strict interactions with WP1.

6 Alpha Software for Testing Knowledge Networks Concepts

We shortly describe here the software modules we have developed so far to put knowledge networks to work and to test and evaluate knowledge network mechanisms. In-progress software development work and future development work are also discussed.

6.1 Knowledge Network Components and KN Repository

We have a number of components implemented to construct and test the knowledge network with. In short, these components are the knowledge atoms, the knowledge containers, with the respective interfaces for accessing to knowledge. The overall software architecture which has been developed so far enables building simple repositories of distributed knowledge (currently generated by simulated sources) and accessing them via the Web.

In the absence of an ACE toolkit available (and of the corresponding communication protocols) at this stage of the project, a simplified XML-RPC mechanism has been chosen as the protocol to provide a distributed Web-based test environment. An existing library, helma.xmlrpc, which is part of the Apache XML-RPC project (http://ws.apache.org/xmlrpc/), has been used to implement this functionality which advantages can be summarised as follows:

- XML-RPC is far more lightweight than Java's built in RMI support due to the fact that it only passes parameters rather than objects.
- Java programs can use XML-RPC to connect directly to any other system supporting XML-RPC, rather than limiting connections to fellow RMI systems or having to use complex (and expensive) CORBA object request brokers. This also allows for direct P2P connections.
- The use of HTTP as a transport substrate makes it relatively easy to integrate XML-RPC with web-enabled applications that spread across a heterogeneous computing landscape.
- XML-RPC uses only a tiny subset of HTTP so that Java applications can easily avoid the overhead of full-scale HTTP processing. The core processing of XML-RPC takes advantage of Java's built-in understanding of TCP/IP.
- XML-RPC allows a client to specify which method it wants to use and then looks for a handler. Because the reference is done by name, there aren't any stubs to manage or include, and changes can be made much more easily at runtime.

In a nutshell, the XML-RPC approach provides a flexible mechanism to access individual components of the knowledge network and their methods directly via the RPC interface. Thus, each knowledge network component can be seen as a distinct resource that is accessible via a unique URI. This promotes one of the key objectives of the Cascadas project. That is to realise knowledge networks with independent and light weight components that can be linked together in a distributed environment. Furthermore, it allows for the dynamic extension of individual components in a way that specific services are added / removed at runtime.

Individual client applications may use Java Remote Procedure Calls (RMI) to communicate with individual network components. The network itself may be composed of a number of distributed networks that also communicate with each other using RPC. On each computational resource, the network is made up of a number of containers and atoms with references to each other. For the simulation environment individual knowledge sources are also modelled by dedicated RPC handlers which can again be remote.

The basic architecture for the developed components hosted in this environment is shown in Figure 36.

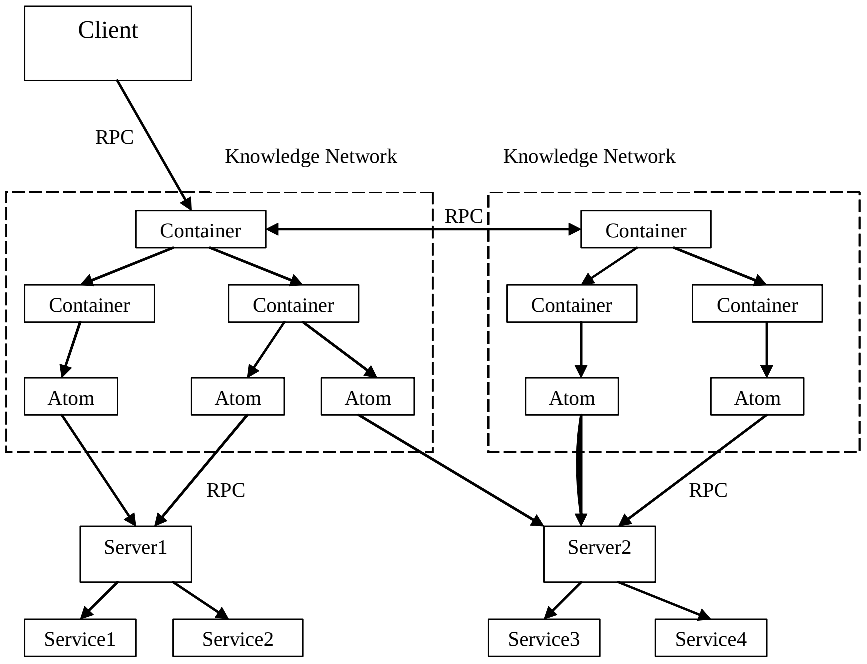


Figure 36: Architecture of the Test Environment.

Currently, the core of both types of components, atom and container, and specific handlers for the unique identification thereof, the provision of context related information and the dynamic registration of services were implemented. This allows the registration of atoms into the virtual space of a knowledge network which is identified by a web server that exposes the XML-RPC service. Furthermore client access to those components and their underlying (simulated) knowledge sources has been realised. As visualised in Figure 37, it is possible to register simulated knowledge sources to virtual knowledge network spaces as simulated by individual RPC services and to generically access those sources via the atom interface. Also, it is possible for clients (e.g., services and ACE – when the latter will be available) to access to knowledge via RPC.

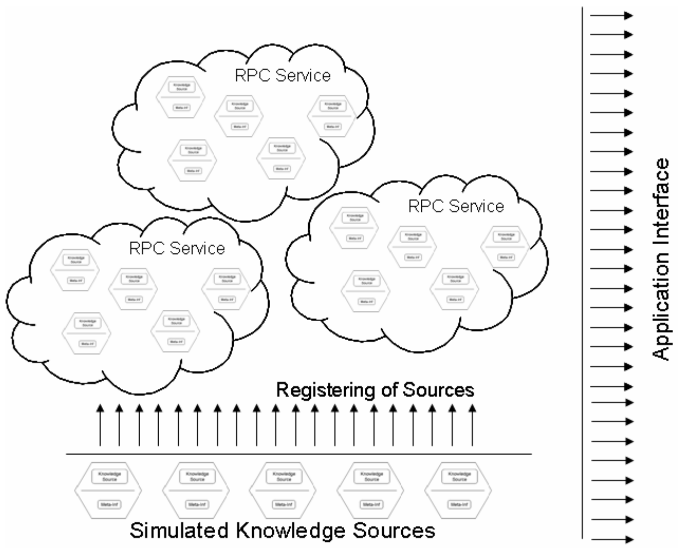


Figure 37: Simulating Knowledge Networks

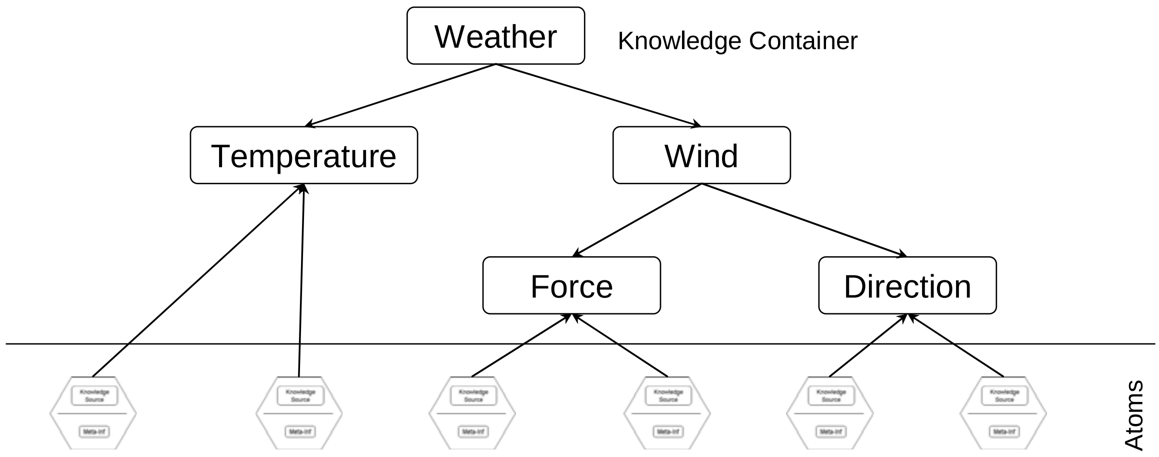


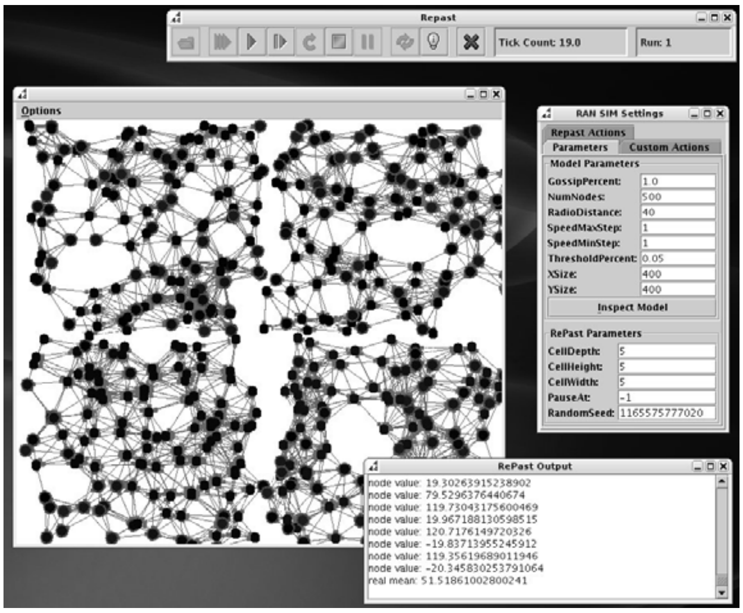
Figure 38: Weather example network.

To evaluate the developed components and concepts we exploited the above software to create a dedicated demonstrator that simulates the hierarchical relations of a simplified yet distributed network for the provision of weather related data. The hierarchical relations of this data are depicted in Figure 38.

Within this example three types of knowledge sources are available, though that number may be extended as desired. These measure temperature, wind force and wind direction, which are conceptually organised as shown. We have generated individual services for each concept in order to serve (simulated) sensed values for each type of sensor. These are made available through an RPC based service also registered to the web server that publishes the components of the knowledge network. Knowledge atoms representing each individual sensor are also generated and added to the web server, where we have around 1500 atoms representing the three different types of sensor for host of selected European cities. Each atom provides a standard interface allowing it to be accessed by any other component in the knowledge space, thus it is uniquely identifiable. Each atom is also self-descriptive in a way that it is equipped with a full set of metadata describing individual concepts of interests, including e.g. its type, its location, its geographical region etc. each of these meta-data may be used inside the network for individual organisational purposes.

This aggregation should produce the network hierarchy shown above, which is not known beforehand but may be derived based on the descriptive data of the atoms registered within the scope of the network.

Other than the above components and mechanisms for hierarchical organization of knowledge components, additional software components to allow for individual knowledge organisation, there included a container component for managing the references inside the network and some distributed algorithms suitable to (self-)organize knowledge (as discussed in the following Subsection) also been implemented and tested. However, these components/algorithms have been not yet fully integrated with the main software. This implies that, so far, no automated organisation can be facilitated ‘inside’ the network, although this forms the cornerstone for the next development stage.



Tick Count: 19.0

Run: 1

Model Parameters

GossipPercent:	1.0
NumNodes:	500
RadioDistance:	40
SpeedMaxStep:	1
SpeedMinStep:	1
ThresholdPercent:	0.05
XSize:	400
YSize:	400

RePast Parameters

CellDepth:	5
CellHeight:	5
CellWidth:	5
PauseAt:	-1
RandomSeed:	1165575777020

```
node value: 19.30263915238902
node value: 79.5296376440674
node value: 119.73043175600469
node value: 19.967188130598515
node value: 120.7176149720326
node value: -19.83713955245912
node value: 119.35619689011946
node value: -20.345830253791064
real mean: 51.51861002800241
```

Figure 39: The simulation environment. On the top there is a bar to control the simulation flow. The window on the right is to set the main parameters and in the centre is visible the main simulation window.

6.2 A Simulator for Knowledge Network Mechanisms

To demonstrate the organisational concepts related to knowledge networks in distributed settings, i.e., to test and evaluate the algorithms for overlay field-based data structures and the region aggregation noise algorithm, we have developed a simulation environment by extending upon the Repast simulator.

The Repast simulator (http://repast.sourceforge.net/) is fully based on Java, and provides a set of basic classes to build simulation of distributed computational scenarios, in which a set of “nodes” are immersed in a spatial environment and there can move and interact with each other. This makes Repast very useful to simulate spatially-situated distributed computational systems such as swarms, ad-hoc networks, sensor networks, by extending the basic classes as needed to simulate specific behaviours for simulated nodes.

We build upon Repast to simulate a number of computer-based nodes capable of interacting with each other in an ad-hoc way, capable of "sensing" properties in the environment, and thus acting as simulated KA. In particular, we used it to simulate and evaluate overlay-based approach and to test the region aggregation noise algorithms, via the development of appropriate KA classes to implement and run these algorithms in the simulation environment.

A snapshot of the simulation environment is in Figure 39.

6.3 Future Extensions

At the time of writing, we are in the (quite advanced) process of integrating the above two software tools (see Figure 40). This integration will make the Repast simulation environment generate data, possibly pre-aggregated and pre-organized by proper distributed mechanisms such as the region aggregation noise simulated over a sensor network. Then, it will interface with the knowledge network repository to dynamically feed with data the knowledge atoms in there. The resulting software package, for which we do not exclude the possibility of integrating other sources of knowledge, will be of great use to demonstrate the possibility of exploiting both distributed and centralized mechanisms for building knowledge networks.

Following this integration and the verifications of its functionalities, we plan add functionalities for coordination of distributed knowledge repositories and to integrate the knowledge network repository with some real data source (i.e., a real sensor network gathering real data from the environment and some PDAs gathering user data such as profiles and GPS position) and of developing some distributed knowledge network mechanisms over a real ad-hoc network of sensors and/or PDAs. The result will be a framework for both centralized and distributed knowledge network management, for which we will make available a proper interface for user to access and manage knowledge. Overall, this will form the so called "beta release" of knowledge network software (Figure 41).

Later on, and in coordination with the other WPs, the beta release will be gradually transformed into a sort of knowledge service layer for the use of ACEs and possibly implemented by ACEs. In other words, we will transform the knowledge access interface into an ACE interface, and will gradually re-engineer the software so as to make it implemented in terms of ACE ensembles.

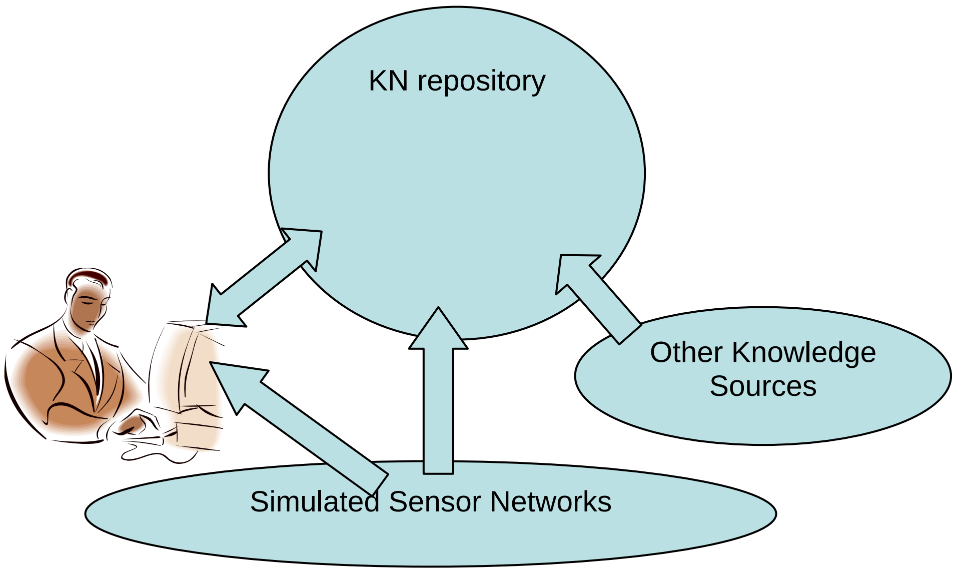


Figure 40: In-progress integration of the two alpha software modules.

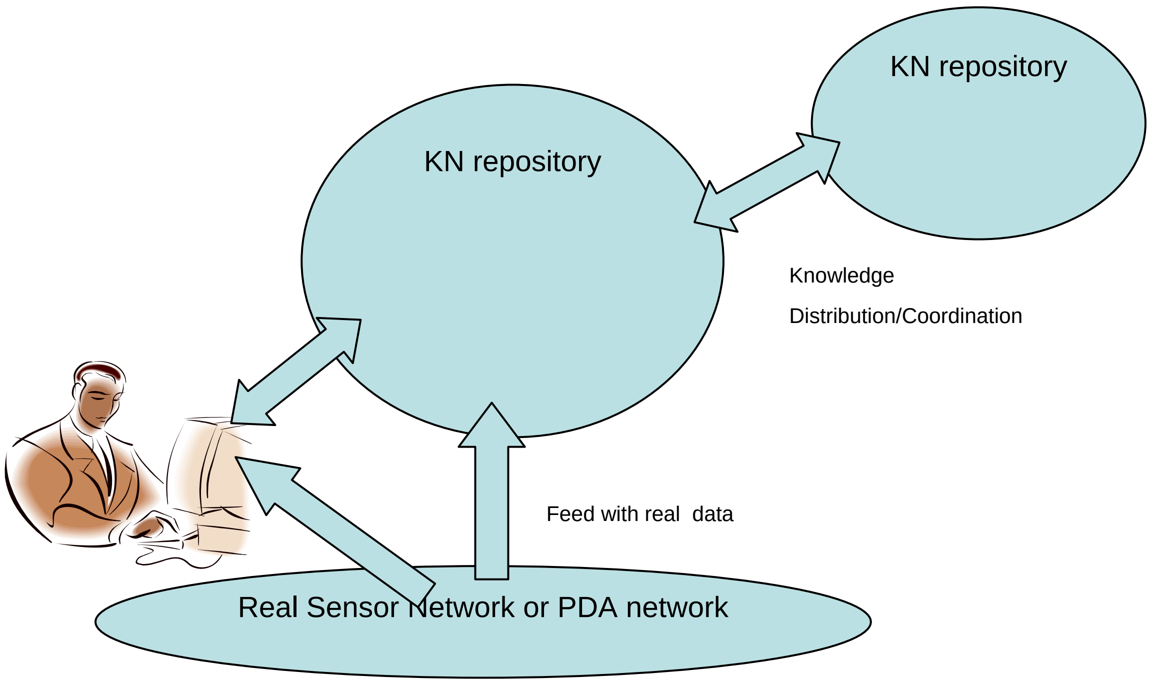

KN repository
KN repository
Knowledge Distribution/Coordination
Feed with real data
Real Sensor Network or PDA network

Figure 41: Possible scheme for beta release of knowledge network software.

7 Conclusions and Roadmap

The reported activities on knowledge network testify the achievement of a series of notable research results. In summary:

- We identified and agreed on a general reference architecture for knowledge networks;

- We managed to work out a sound set of structural specifications for knowledge network components and their organization:
- We deeply studied and successfully experienced several distributed mechanisms and algorithms for managing knowledge;
- We made a thorough analysis of the possible applications of knowledge networks for autonomic communication
- We developed a set of alpha modules for testing with knowledge network concepts.

In addition, within the workpackage, we have also well-defined ideas on how to proceed with our activities, to make the knowledge network idea a practical and usable tool for the support of autonomic communication services.

Next steps will focused on two main threads of inter-related activities: *(i)* studying more advanced mechanisms for knowledge networks and *(ii)* advancing the study the issue of managing knowledge network ensembles and the development of the associated software infrastructure and tools.

7.1 Knowledge Network Mechanisms

The study of knowledge network mechanisms will concern algorithms by which it is possible to have a set of distributed "sensors" (i.e., knowledge atoms) self-organize with each other so as to autonomically and adaptively produce newly aggregated knowledge about a situation, with a particular attention to aggregation along the spatial dimension (which is particularly useful in pervasive computing scenarios). Self-organization and knowledge management along the temporal dimension will be explored too, as a possible mechanism to integrate advanced inference mechanisms in knowledge networks, and eventually being able to produce "prediction" about likely-to-occur situations.

Also, we will study mechanisms which enable knowledge components (i.e., knowledge atoms) to self-aggregate along a general semantic dimension, other than along the spatial dimension. This implies having knowledge atoms (i.e., the knowledge within them) represented according to a specific ontology, and studying algorithms that makes it possible to self-identify semantic relationships between knowledge atoms, build in an adaptive way the corresponding semantic networks, and exploit these semantic networks to infer in expressive and powerful ways information about situations occurring in a context.

At a later stage, we will eventually study how the overall framework identified for knowledge network can be fruitfully exploited as an infrastructure to enforce forms of knowledge-mediated interactions between ACEs. These would somewhat resemble forms of pheromone-mediated interactions in ant colonies, and would capture similar properties of self-organization and self-adaptation, with the add-on of the possibility of being based on meaningful knowledge and of exploiting higher cognitive capabilities than that available by simple reactive ants.

7.2 Knowledge Network Ensembles and Knowledge Network Software

As a second thread of activities, we will explore the research issues involved in the management of situated knowledge network ensembles and in the exploitation of such knowledge by ACEs. Following the achieved identification of the structural requirements and of some basic mechanisms for knowledge management (which already reflects in the alpha software release), we will continue researching how situated knowledge networks can be put to use, and how knowledge may be combined/split in differing scales of use by ACEs and ACE aggregates.

A specific attention will be put on studying and experimenting how ensembles of knowledge components can be made available to components in a network (which implies identifying strategies for knowledge propagation) and how such knowledge can be made available at different levels of granularity, so as to enforce a self-similar perspective on knowledge management and access.

Another related aspect that will be deeply studied concerns the requirements for knowledge networks to be scalable and light weight. While such properties can be achieved via various means, the nature of knowledge networks – which is overall a form of distributed data structure – suggests exploring methods and strategies for properly partitioning the ACEs forming the knowledge network across the network, as well as methods and strategies for self-adaptation of such distribution. The study of these strategies will be investigated also from a semantic and knowledge driven approach, and will possibly involve formulation of regression based models and self-growing cell structures (as found in the domain of unsupervised learning).

Strictly related to the above, the task will focus on developing proper tools, metrics, and algorithms, for the evaluation and monitoring of knowledge networks. A particular attention will be posed on algorithms for increasing the reliability and the degree of trust in the knowledge provided by knowledge networks, that is, algorithms to enable a knowledge network to self-verify whether the interpreted/aggregated context is correct and consistent and, in the case of inconsistencies, making being able to determine the source of the failure try to resolve the inconsistencies.

All these activities will also directly result in the continuous advance of the knowledge network software (along the lines identified in Section 6 and with the continuous integration in it of new mechanisms and tools), to become eventually part of the CASCADAS Open Source Toolkit.

8 References

[ABA+03] Ittai Abraham, Baruch Awerbuch, Yossi Azar, Yair Bartal, Dahlia Malkhi, Elan Pavlov, (2003). "A Generic Scheme for Building Overlay Networks in Adversarial Scenarios", Proceedings of the 17th International Symposium on Parallel and Distributed Processing, 22-26 April 2003, Nice, France.

[AgSi94] R. Agrawal, S. Srikant: Fast Algorithms for Mining Association Rules, Proc. Of the 20th VLDB Conference, Santiago, Chile, 1994

[AgSi95] R. Agrawal, R. Srikant; Mining Sequential Patterns; Proc. Of the Int'l Conference on Data Engineering (ICDE); Taipei, Taiwan, March 1995. Expanded version available as IBM Research Report RJ9910, October 1994

[AIS93] R. Agrawal, T. Imielinski, A. Swami; Mining Associations between Sets of Items in Massive Databases; Int'l Conference on Management of Data; Proc. Of the ACM-SIGMOD; Washington D.C., May 1993, 207-216.

[AlbB02] R. Albert, A. Barabasi, "Statistical Mechanics of Complex Networks", Reviews of. Modern. Physics, 74(47), 2002.

[AndS04] S. Androutsellis-Theotokis, D. Spinellis, "A Survey of P2P Content Distribution Techniques", ACM Computing Surveys, 36(4):335-371, Dec. 2004.

[Bab05] O. Babaoglu, G. Canright, A. Deutsch, G. Di Caro, F. Ducatelle, L. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, "Design Patterns from Biology for Distributed

Computing", in Proceedings of the European Conference on Complex Systems, Paris (F), November 2005.

[BabMM02] O. Babaoglu, H. Meling, A. Montresor, "Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems", IEEE International Conference on Distributed Computing Systems, Vienna, Austria, 2002.

[BaDR04] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg: A Survey on context-Aware systems. 2004

[BBM+02] Alessandro Bassi, Micah Beck, Terry Moore, James S. Plank, (2002). "The Logistical Backbone: Scalable Infrastructure for Global Data Grids", Proceedings of Asian Computing Science Conference, Hanoi, Vietnam, Lecture Notes in Computer Science, Vol. 2550/2002, pp. 1-12, Springer, December 4-6, 2002.

[BCA01] Blair, G.S., Coulson, G., Andersen, A. (2001). The design and implementation of OpenORB version 2, IEEE Distributed Systems Online Journal 2 (6), pp: 45-52

[BeRH94] Frazer Bennett, Tristan Richardson, and Andy Harter: Teleporting - Making Applications Mobile. In Proceedings of 1994 Workshop on Mobile Computing Systems and Applications, Santa Cruz, December 1994.

[BH02] Erwin Bonsma, Cefn Hoile (2002) "A distributed implementation of the SWAN peer-to-peer look-up system using mobile agents", 1st International Workshop on Agents and Peer-to-Peer Computing (AP2PC 2002), AAMAS2002, Bologna, Italy, July 2002.

[BKR+02] Rebecca Braynard, Dejan Kostic, Adolfo Rodriguez, Jeffrey Chase, and Amin Vahdat, (2002) "Opus: an Overlay Peer Utility Service", Proceedings of the 5th International Conference on Open Architectures and Network Programming (OPENARCH), June 2002.

[BMU+97] S. Brin, R. Motwani, J.D. Ullman, S. Trur; Dynamic Itemset Counting and Implication Rules for Market Basket Data. Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 255-264, Tuscon, Arizona, May 13-15 1997

[BonDT99] E. Bonabeau, M. Dorigo, G. Theraulaz, "Swarm Intelligence. From Natural to Artificial Systems". Oxford University Press, 1999.

[BoTh00] Bonabeau, E. and Theraulaz, G. (2000). Swarm smarts. Scientific American, pp. 72-79, March

[BreO04] C. Brewster, K. O'Hara, "Knowledge Representation with Ontologies: Present and Future", IEEE Intelligent Systems, Vol. 20, No.1, pp. 71-81, Jan. 2004.

[BrLe04] Ronald J. Brachman, and Hector J. Levesque: Knowledge Representation and Reasoning. Morgan Kaufmann Publishers, San Francisco, California, USA, 2004

[Cab03] G. Cabri, L. Leonardi, M. Mamei, F. Zambonelli, "Location-dependent Services for Mobile Users", IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems And Humans, Vol. 33, No. 6, pp. 667-681, November 2003

[Cast02] Castro, M.;, "One Ring to Rule Them All: Service Discovery and Binding in Structured Peer-to-Peer Overlay Networks," SIGOPS, France, September 2002.

[Chen04] Harry Lik Chen: An Intelligent Broker Architecture for Pervasive Context-Aware Systems. PhD thesis, University of Maryland, Baltimore Country, USA, 2004

[CKG04] Condie, Tyson; Kamvar, Sepandar; Garcia-Molina, Hector; Adaptive P2P Topologies; Technical Report, 2004

[Cla03] Clark, D., Partridge, C., Ramming, C., Wroclawski, J. (2003). A Knowledge Plane for the Internet, Proceedings of the 2003 ACM SIGCOMM Conference, Karlsruhe (D), ACM Press, pp: 3-10

[CMK+99] Campbell, A. T., De Meer H. G., Kounavis M. E., Miki K., Vicente J. and Villela, D. A., (1999 "The Genesis Kernel: A Virtual Network Operating System for Spawning Network Architectures", Proc. IEEE OPENARCH'99, New York, March

[CMN+06] K. Curran, M. Mulvenna, C. Nugent, A. Galis; "Challenges and Research Directions in Autonomic Communications"; International Journal of Internet Protocol Technology , Vol. 2, No. 1, 2006, ISSN: 1743-8209, InderScience Publishers

[CSW+00] I. Clarke, O. Sandberg, B.Wiley, and T.W. Hong, (2000) "Freenet: A Distributed Anonymous Information Storage and Retrieval System", Workshop on Design Issues in Anonymity and Unobservability, pages 311–320, July 2000.

[Cur05] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A.L. Murphy, and G.P. Picco, TinyLime: Bridging Mobile and Sensor Networks through Middleware, In Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom'05), IEEE CC Press, pp. 61-72, March 2005.

[DeAb01] Anind K. Dey, and Gregory D. Abowd: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Human-Computer Interaction (HCI) Journal, Volume 16, No. 2, 3 and 4, 2001

[Dey00] Anind K. Dey: Providing Architectural Support for Building Context-Aware Applications. PhD thesis, Georgia Institute of Technology, 2000

[Doy05] J. Doyle, et al., "The Robust Yet Fragile Nature of the Internet", Proceedings of the National Academy of Science, Vol.102, No. 41, Nov. 2005.

[EO98] Association and Sequencing, Exclusive Ore Inc., 1998 – 2000

[Est02] D. Estrin, D. Culler, K. Pister, G. Sukjatme, "Connecting the Physical World with Pervasive Networks", IEEE Pervasive Computing, 1(1):59-69, Jan. 2002.

[GaCr03] Garcia-Molina, Hector; Crespo, Arturo; Semantic Overlay Networks for P2P Systems, Technical Report Stanford University, 2003

[GaSt04] Jun Gao, P. Steenkiste (2004). "Design and evaluation of a distributed scalable content discovery system", IEEE Journal on Selected Areas in Communications, Vol. 22, No. 1, pp. 54-56, Jan. 2004.

[GBR+99] Grossman, R., Bailey, S., Ramu, A., Malhi, B., Cornelison, M., Hallstrom, P., and Qin, X.. The Management and Mining of Multiple Predictive Models Using the Predictive Modeling Markup Language (PMML). AFCEA 1999 Conference.

[GeNi87] Genesereth, M. R., Nilsson, N. J.; "Logical Foundations of Artificial Intelligence", Morgan-Kaufman; ISBN 0934613311, 1987.

[GFC00] Ghosh, A., Fry, M., and Crowcroft, J., "An Architecture for Application Layer Routing," Active Networks, May 2000.

[Gint00] Ginits, H.; "Game Theory Evolving"; Princeton: Princeton University Press; ISBN: 0691009430, 2000

[Gnut01] Gnutella: http://www.gnutelliums.com/ (2001)

[Gruber] http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

[HalA06] D. Hales, S. Arteconi, "SLACER: A Self-Organizing Protocol for Coordination in P2P Networks", IEEE Intelligent Systems, Vol. 22, No. 2, April 2006.

[HaZa03] M. El-Hajj, O. R. Zaïane, COFI-tree Mining: A New Approach to Pattern Growth with Reduced Candidacy Generation, in Workshop on Frequent Itemset Mining Implementations (FIMI'03) in conjunction with IEEE-ICDM 2003, Melbourne, Florida, USA, 19 November, 2003

[HeIR03] K. Hendricksen, J. Indulska, and A. Rakotonirainy: Generating context management infrastructure from high-level context models. In the Proc. of the 4th Internation conference on Mobile Data Management, 2003

[HoLa01] Jason I. Hong, and James A. Landay: An infrastructure approach to context-aware computing. Human-Computer Interaction, Vol. 16, 2001

[Holl96] Holland O.E. (1996), Multi-agent systems: lessons from social insects and collective robotics, AAAI Spring

[HPM+00] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M. Hsu; FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining. Int. Conf. Knowledge Discovery and Data Mining (KDD2000), Boston, 2000, pp 355 - 259

[HPSH00] Ken Hinckley, Jeffrey S. Pierce, Mike Sinclair, and Eric Horvitz: Sensing techniques for mobile interaction. In UIST, pages 91–100, 2000

[HPY00] J. Han, J. Pei, Y. Yin. Mining Frequent Patterns without Candidate Generation, Proc. 2000 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'00), Dallas, TX, May 2000

[HSPL02] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, and J. Altmann: Context-awareness on mobile devices – the hydrogen approach. In the Proc. of the 36th Annual Hawaii International Conference on System Sciences, 2002

[HuhS05] M. N. Huhns, M. P. Singh, "Service-Oriented Computing: Key Concepts and Principles", IEEE Internet Computing 9(1):75-81, 2005.

[HWBM02] C. Hoile, F. Wang, E. Bonsma and P. Marrow, "Core Specification and Experiments in DIET: A Decentralised Ecosystem-inspired Mobile Agent System", Proc. 1st Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS2002), pp. 623-630, Bologna, Italy, July 2002,.

[Keah02] Keahey, K., (2002). "Computational Grids in Action: The National Fusion Collaboratory," Future Generation Computer Systems, Vol. 18, No. 8, October 2002, pp. 1005-1015.

[KlMa02] Klingberg, T., and Manfredi, R.; The Gnutella Protocol Version 0.6 Draft, Gnutella Developer Forum, 2002, http://groups.yahoo.com/group/the_gdf/files/Development/.

[Korp01] Korpela, E.; "SETI@home: Massively Distributed Computing for SETI," Computing in Science and Engineering, Vol. 3, No. 1, January 2001.

[LaFl94] Mik Lamming and Mike Flynn: Forget-me-not: intimate computing in support of human memory. In Proceedings FRIEND21 Symposium on Next Generation Human Interfaces, 1994

[LCH+05] B.T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, I. Stoica; Implementing Declarative Overlays;20th ACM Symposium on Operating Systems Principles (SOSP), Brighton, UK, 2005.

[LeGu90] Douglas B. Lenat, and Ramanathan V. Guha: Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project. Addison-Wesley, Reading, Maryland, USA, 1990

[LiuS06] H. Liu, P. Singh, "ConceptNet: a Practical Commonsense Reasoning Toolkit", BT Technology Journal, 2006, to appear.

[LRS02] Lv, Q., Ratnasamy, S., and Shenker, S., (2002) "Can Heterogeneity Make Gnutella Scalable?" Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), Cambridge, MA, March 2002

[LWS+03] Alexander Löser, Martin Wolpers, Wolf Siberski, Wolfgang Nejdl. Semantic Overlay Clusters within Super-Peer Networks .International Workshop on Databases, Information Systems, and P2P Computing, colocated with 29th International Conference on Very Large Databases (VLDB2003), Berlin, Germany, 2003.

[MamZ06] M. Mamei, F. Zambonelli, Field-based Coordination for Pervasive Multiagent Systems, Springer-Verlag (Berlin, D), 2006.

[MamZL04] M. Mamei, F. Zambonelli, L. Leonardi, "Co-Fields: a Physically Inspired Approach to Distributed Motion Coordination", IEEE Pervasive Computing, 3(2):52-60, April 2004.

[ManZ06] A. Manzalini, F. Zambonelli, "Towards Autonomic and Situation-Aware Communication Services", 1st IEEE Workshop on Distributed Intelligent Systems, Prague (CZ), June 2006.

[MCP98] F. Masseglia, F. Cathala, P.Poncelet; The PSP Approach for Mining Sequential Patterns. 2nd European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD98), France, 1998, pp 176 – 184

[MenT03] R. Menezes, R. Tolksdorf, "A New Approach to Scalable Linda-systems based on Swarms", ACM Symposium on Applied Computer, Orlando, FL, USA, 2003

[MTT03] De Meer, H., Tutschku, K., and Tran-Gia, P. (2003) "Dynamic Operation in Peer-to-Peer Overlay Networks," Praxis der Informationsverarbeitung und Kommunikation, (PIK Journal), Special Issue on Peer-to-Peer Systems, June 2003.

[MulA06] R. Müller, G. Alonso, "Shared Queries in Sensor Networks for Multi-User Support", Technical Report 508, ETH Zürich, Institute of Pervasive Computing, Feb. 2006. 23(3):219-252, Aug. 2005.

[MulZ06] M. Mulvenna, F. Zambonelli, K. Curran, C. Nugent, "Knowledge Networks", 2nd IFIP Workshop on Autonomic Communication, LNCS No. 3947, January 2006.

[Nag02] R. Nagpal, "Programmable self-assembly using biologically-inspired multiagent control", ACM Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy, 2002

[NSNT97] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker: Agile application-aware adaptation for mobility. In Sixteen ACM Symposium on Operating Systems Principles, pages 276–287, Saint Malo, France, 1997

[OWL] OWL Web Ontology Language Overview, http://www.w3.org/TR/owl-features/

[Par97] V. Parunak, "Go to the Ant: Engineering Principles from Natural Agent Systems", Annals of Operations Research, 75:69-101, 1997.

[PBS04] Parunak, V., Brueckner, S., Sauter, J.; Digital Pheromones for Coordination of Unmanned Vehicles. Workshop on Environments for Multi-agent Systems (E4MAS), LNAI 3374, Springer Verlag, 2004

[PCY97] J.-S. Park, M.-S. Chen, P. S. Yu, Using a Hash-Based Method with Transaction Trimming for Mining Association Rules, IEEE Trans. On Knowledge and Data Engineering, Vol. 9, No. 5, October 1997, pp. 813-825.

[Peri02] Filip Perich: A service for aggregating and interpreting contextual information. Technical report, Hewlett Packard Labs, 2002

[Phi04] M. Philipose, K. Fishkin, M. Perkowitz, D. Patterson, D. Fox, H. Kautz, D. Hahnel, "Inferring Activities from Interactions with Objects", IEEE Pervasive Computing, 3(4):50-57, 2004.

[PHM+01] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. In Proceedings of the 2001 International Conference on Data Engineering (ICDE'01), Heidelberg, Germany, April 2001.

[PlatoGT] http://plato.stanford.edu/entries/game-theory/

[Poo01] R. Poor, "Embedded Networks: Pervasive, Low-Power, Wireless Connectivity", PhD Thesis, MIT, 2001.

[PWC+05] Parunak, H. Van, Peter Weinstein, Paul Chiusano, Sven Brueckner. 2005. Sift and Sort: Climbing the Semantic Pyramid. Proceedings of ESOA'05, LNCS, 2005.

[Rat01] S. Ratsanamy, P. Francis, M. Handley, R. Karp, "A scalable content-addressable network". ACM SIGCOMM Conference, San Diego, CA, USA, 2001

[Ratn02] Ratnasamy, S., (2002) "A Scalable Content-Addressable Network," Ph.D. Thesis, U.C. Berkeley, October 2002.

[RoDr01] Rowstron, A., and P. Druschel, (2001) "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", International Conference on Distributed Systems Platforms (Middleware), pages 329–350, Heidelberg, Germany, Nov. 2001.

[SGMH04] Sterritt R, Gunning D, Meban A, Henning P, (May 2004) "Exploring Autonomic Options in an Unified Fault Management Architecture through Reflex Reactions via Pulse Monitoring", Proceedings of IEEE Workshop on the Engineering of Autonomic Systems (EASe 2004) at the 11th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2004), Brno, Czech Republic, 24-27 May, Pages 449-455

[ShBe05] Q. Z. Sheng, and B. Benatallah: ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services. Society, Proc. of the 4th International Conference on Mobile Business, 2005

[SheS02] W. Shen, B. Salemi, P. Will. Hormone-inspired adaptive communication and distributed control for conro self-reconfigurable robots. IEEE Transactions on Robotics and Automation, 18(5):1-12, 2002.

[Stoic01] I. Stoica, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," ACM SIGCOMM '01, San Diego, CA, September 2001.

[Stoke03] Stokes, M.;"Gnutella2 specification document – first draft," Gnutella2 Web site (http://www.gnutella2.com/gnutella2_draft.htm), 2003.

[StoN04] K. Stoy, R. Nagpal, "Self-Reconfiguration Using Directed Growth", 7th International Symposium on Distributed Autonomous Robotic Systems, Toulouse (F), 2004.

[StPo04] T. Strang, and C. Linnhoff-Popien: A context modelling survey. In the Proc. of the First Internation Workshop on Advanced Context Modelling, Reasoning And Management, 2004

[Str03a]. Strang, T. Service Interoperability in Ubiquitous Computing Environments, PhD thesis, Ludwig Maximilians University Munich, Oct. 2003.

[Str03b]. Strang, T., Linnhoff-Popien, C., Frank,K. CoOL: A Context Ontology Language to enable Contextual Interoperability. In LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003), J.-B. Stefani, I. Dameure, and D. Hagimont, Eds., vol. 2893 of Lecture Notes in Computer Science (LNCS), Springer Verlag, pp. 236–247, 2003.

[SUN05] SUN Microsystems, (2005) "JXTA Technology", http://www.sun.com/software/jxta/, March, 2005.

[Toi96] H. Toivonen; Sampling large databases for association rules; 22th International Conference on Very Large Databases (VLDB'96), 134 – 145, Mumbay, India, September 1996. Morgan Kaufmann.

[Tum05] L. Tummolini, C. Castelfranchi, A. Ricci, M. Viroli, A. Omicini, "Exhibitionists and Voyeurs do it better: A Shared Environment Approach for Flexible Coordination with Tacit Messages", Environments for MultiAgent Systems. LNAI 3374, Springer-Verlag, January 2005.

[Usc96] Uschold, M. and M. Grueninger, Ontologies: Principles, methods, and applications. Knowledge Engineering Review, 1996. 11(2): p. 93–155.

[Wan04] Wang, X. H., Zhang, D.Q., Gu, T., Pung, H.K., Ontology Based Context Modeling and Reasoning using OWL. In Workshop Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communications (PerCom2004), (Orlando, Fl.,USA, March 2004), pp. 18–22.

[Wat98] Watts, D., J., Strogatz, S.H., Collective Dynamics of 'Small-World' Networks. Nature, 1998. 393: p. 440-442.

[WHFG92] Roy Want, Andy Hopper, Veronica Falcao, and Jon Gibbons: The active badge location system. Technical Report 92.1, Olivetti Research Ltd., ORL, 24a Trumpington Street, Cambridge CB2 1QA, 1992

[Wino01] Terry Winograd: Architectures for Context. Human-Computer-Interaction, vol.16, no. 2, 3 and 4, 2001

[Zaki00] M. J. Zaki; Scalable Algorithms for Association Mining; IEEE Transactions on Knowledge and Data Engineering, Vol. 12, No. 3, May/June 2000, pp 372-390.

[Zaki01] M. J. Zaki, SPADE: An Efficient Algorithm for Mining Frequent Sequences; in Machine Learning Journal, special issue on Unsupervised Learning (Doug Fisher, ed.), Vol. 42 Nos. 1/2, Jan/Feb 2001, pp 31-60.

[Zam06] F. Zambonelli, "Self-Management and the Many Facets of Non-Self", IEEE Intelligent Systems, Vol. 22, No. 2, April 2006.