1

Distributed Context Space (DCS) - foundation of semantic P2P systems

Thomas Schwotzer

HTW Berlin
Wilhelminenhofstrasse 75A, 12459 Berlin, Germany
Thomas.Schwotzer@HTW-Berlin.de

Summary. Nearly any social network application is based on the client-server paradigm. This has several serious drawback (data security, costs). This paper introduces the concept of Distributed Context Space which is a concept for loosely coupled (mobile) P2P semantic systems. Shark is a reference implementation of DCS and iSphere is a social network application based on Shark. This paper shows how Semantic Web approaches combined with P2P can substitute the C/S paradigm in Web 2.0.

Key words: P2P, Distributed Semantic Web, Ad-hoc Networks, Social Networks

1.1 Introduction

The terms *Internet* and *World Wide Web* are often used as synonyms but there are fundamental differences: Internet is a worldwide connection of computers by means of the IP protocol. IP routes data through a dynamic network of connections. There is *no server* neither in the IP protocol nor in the Internet. TCP introduces flow control and reliable data transfer. From developers point of view, there are TCP server and clients. But it is just a technical issue. Each computer in the *Net* can be TCP client and server.

Client and server are intrinsic concepts of the World Wide Web, though. *Web server* store data. *Web clients* can get access to them. HTTP is the well-known communication protocol in the *Web*. The Web has its roots in FTP and Gopher server. Such data server had been created in a time in which computing power and hard drives were very expensive. It was not feasible to have all data in a desktop PC but it was possible to have a slim and cheap computer on the desk[1]. This time is over. Ordinary PCs are delivered with at

[1] There were often called *terminal* because there were the endpoint of a connection from an expensive and huge server computer somewhere in the cellar in the IT department.

least some hundred megabytes on the hard drive. Internet bandwidth isn't a technical challenge any more.

Web-Programming became *the* dominant way of building distributed systems in the Net. In its beginning, it was seen as a more convenient interface to FTP server and actually it was not more. For a growing number of developers, client-server (C/S) programming with Webframeworks became a kind of natural and sometimes only thinkable way of building Internet applications. This is a *Lock In* phenomenon, some call it a dangerous effect [Lan10].

Web applications are C/S applications. Web 2.0 doesn't differ from the WWW from technical point of view. It is still HTTP and HTML usually enriched with code running in the WWW browser which makes it faster and colorful. Web 2.0 applications allow users to changed the content on the server. Wikipedia, Wikis in general, blogs and social networks applications are well-known.

There are major differences from the application perspective, though. Web 2.0 users create a network to exchange data, e.g. rumors, pictures, experiences. Users are looking for others which similar interests. Especially social networks are the Internet pendant of the often cited ancient forum. People can simply go there, meet and have a talk.

But there is a fundamental and crucial problem with Web 2.0 based social networks: There is a server all the time. This has several remarkable drawbacks:

- Personal profiles are stored on a server. There is no principal technical barrier that prevents misuse of personal data[2]. Moreover, the core of the business concept of social network are dealing with user profiles: They run the WWW infrastructure and sell personal and private data of there users e.g. to marketing companies. Thus, *joining a social networks means giving away personal and sometimes sensitive data to a crowd of unknown information dealer.*
- Server based Web applications are potentially visible to the whole Web. From a technical point of view, each *private* discussion in a forum could be visible in the overall net. Blogs and Wikipedia *are created* as worldwide information exchange platform. Social networks give the *impression of privacy*, though. It is just an impression. An incredible number of such *private* discussions can be found with Web search engines. Especially young users with less IT experiences are tend to became victims of such not expected data leaks. Web based social networks are not a save and private place. Web based social networks are all the time, 24 hours 7 days a week under wordwide observation. It shouldn't be compared with an ancient forum - it is a kind of *big brother show*[3].

[2] Encryption isn't an option. Platform functionality (running on a server) needs processable data.

[3] http://en.wikipedia.org/wiki/Big_Brother_(TV_series)

- Applications become mobile due to the fast evolution of mobile devices from heavy telephones to fully fledged mobile computers. Modern SmartPhones support a variety of communication protocols in different layers. They support GSM, GPRS and other 3G protocols, e.g. UMTS especially in Europe. Bluetooth and Wireless LAN are supported on a growing number of SmartPhones.
  Usually, there are two general ways of building mobile applications: A native application can be build which runs with iOS, Android, Symbian, Windows Mobile, a mobile Linux etc. Native applications usually communicate with a server via TCP or HTTP.
  Nearly any SmartPhone comes with a Web browser. Mobile Web applications can be build just by writing code in a script language which is supported by a mobile browser. There are frameworks like PhoneGap[4], which supports cross platform mobile Web development.
  Mobile Web 2.0 are C/S applications as well. Each data is sent from the mobile phone to a server via public land mobile networks (PLMN). Each data transmission consumes resources (money, bandwith, time). Imagine a scenario in which two mobile users are standing nearby exchanging a picture. In worst case, it runs through two different PLMNs and produces costs for both partners - just to bridge a gap of several meters. See [Sch09a] for a picture and the full worst case scenario: *Mobile access to social networks wastes resources.*

Building a WWW application has two aspects: Building a server and client. Server technologies were developed quite fast in the beginning of the Web. There are a number of well-known Web server supporting different programming languages. The Web 2.0 was a booster for tools and frameworks for rich client development. Web GUIs can be build even without detailed knowledge about computer science. Building a Web application with developer frameworks and tools in general doesn't require computer scientists any longer. Very good.

Nevertheless, a C/S architecture isn't an appropriate way of building social networks due to at least the reasons above. This paper introduces the concept of Distributed Context Space (DCS) and a reference implementation called (Shark - Sharked Knowledge). DCS is based on the P2P paradigm. It uses semantic technologies to transmit data and interests between peers. DCS fits to spontaneous networks and the reference implementation proofed that DCS allows to build a social network application without a server. This system is called *iSphere*. It runs on less mobile PCs which communicate via fixed Internet protocols and on mobile phones communicating via PLMNs but also via Bluetooth. DCS doesn't make any assumption about the underlying protocol. It even runs on top of the WWW protocols - with a major difference: Each DCS node has its own WWW server. Thus, the client can be coded with all sophisticated GUI building tools. The communication between the

[4] http://www.phonegap.com/

DCS nodes is done with the DCS protocol, though. This approach is similar to the concept of Diaspora, see section 1.3. The difference are in the concept of finding other users. DCS uses a concept called interest which is a semantic data structure. Details and examples will be presented and discussed.

1.2 Distributed Context Space (DCS)

1.2.1 Communication Concept - KEP

Each communication takes place in a context. People use different words, offer different ideas, personal opinions etc. depending of the context. A talk with friends in a bar differs fundamentally from a presentation at a trade show. *Context* is a core concept of DCS.

DCS only describes the communication between two peers. In the first step, a peer shall be seen as an individual[5].

Peers can communicate. The Knowledge Exchange Protocol (KEP) is used to formalize this process. KEP is an asynchronous, stateless protocol. There are two commands.

Expose: A **peer** can **expose** an **interest** to another peer in an asynchronous fashion[6]. The issuer cannot be sure that its interest has reached the recipient (the **remote peer**). An interests states about what, when, where and with whom a peer is willing to communicate.

Insert: A peer can send **knowledge** to a remote peer in the same asynchronous way. Knowledge is a number of information in arbitrary formats. Information is set into a context when send. Details will be explained in this paper.

KEP doesn't define neither an order of KEP messages nor the reaction on a KEP message. DCS peers are autonomous peers. The can send message, receive messages whenever they want to. Let's have an example. A peer (lets call it Anton) can e.g. walk around and *expose* any other peer e.g. Bianca its interest in e.g. soccer. Bianca retrieves such an interest can do whatever she wants. She can answer and declare that she is also interested in soccer. She can also send an *insert* including *knowledge* which actually is a message, picture or movie about a soccer game. She can also store that interest and expose any other peer Antons' interest. She can also ignore Anton and his interests.

A peer can also send knowledge to any passing peers. This is the digital pendant for delivering leaflets in e.g. a shopping mall.

[5] It can be shown that DCS peers can represent *groups* of persons. It can also be shown that *federations of peers* can be build which model information and knowledge flow in hierarchical organization. This is out of scope of this paper.

[6] In the reference implementation, an *relay peer* is introduced which allows a asynchronous exchange even over synchronous protocols like TCP.

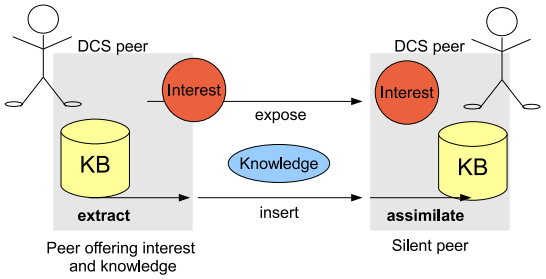

Fig. 1.1. Active and passive (silent) DCS peer

DCS assumes that each peer is an independent entity. It cannot be described what a peer has to do. It only describes a protocol between peers. Peers can even betray and lay. That's reality: Each information which is passed from a local data storage to another entity can potentially be used in each thinkable way. Encryption can be used to ensure, that information cannot be read by third parties. But no concept can really be ensure that the recipient uses information as the sender wants it to do.

DCS assumes that each peer has a conceptual **knowledge base** in which information can be stored. Each peer can get new knowledge from other peers by means of the KEP *insert*. Each peer can decide how to handle such knowledge. The decision will usually strongly depend on the context. Knowledge can be dropped, it can (in parts) be added to the knowledge base. The process of adding new knowledge to the knowledge base is called **assimilation**. Peers can send knowledge. Taking knowledge from the local knowledge base is called **extraction**.

1.2.2 Interests and Context Space

An interest describes the context in which a peer is willing to communicate. It has seven facets:

topic (cardinality: 0..n) The topic facet describes about *what* a peer is willing to communicate. Soccer was the topic in the example above. Each interest can have an arbitrary number of topics. An empty topic facets indicates that the peer is interested in everything.

peer (0..1) The peer facet describes the peer itself. Each peer *can* reveal its identity. The peer facet is not the same as a page in a social network. The peer facet doesn't contain any picture etc. It contains just a single name and an arbitrary number of addresses (TCP, Bluetooth, E-Mail etc.). A peer doesn't have to reveal its identity. It is an *anonymous peer* in this case. Obviously, authentication certification techniques should be used to ensure that a peer is who it claims to be.

remote peer(s) (0..n) This facet is used to describe with whom a peer wants to communicate. Such a concept is often called a *white list*. An empty set is interpreted that the user wants to communicate with anybody. Peers who get an interest can follow this list but are also free to ignore it. Peer *Dave* can explain that it only wants to communicate with peer *Ester* about a topic. It is possible that another peer*Fabius* gets the original interest. He can now contact *Dave*. *Dave* has no way to prevent *Fabius* from sending a message. But *Dave* is free to ignore any message from him.

originator peer (0..1) Peers can exchange knowledge. Knowledge is information in a context. The originator facet describes who actually put information in that context. Imagine a peer L that holds a little library. Another peer A may have access to L's library and gets a copy. A third peer B could be interested in articles about topic t. B might also know that L has a well organized library and other peers have access. B could declare its interest: B (*peer facet*) is interested in information about t (*topic*). It would communicate with arbitrary peers (*remote peer*) but it is only interested in knowledge which has been inserted by L (*originator*). An originator isn't necessarily the author. It is a peer that put information into a context.

direction (0..2) This facet describes the way in which knowledge is ought to flow. Two values are allowed in this facet: *in* and *out*. **In** states an interest to retrieve knowledge and *out* states that the peers is willing to send knowledge. Both directions are allowed if this facet isn't specified or both tags are set.

time (0..n) A peer can define when (periods of time) it is willing to communicate. As already mentioned: This facet can be ignored by other peers. Each peer can contact another peer whenever it wants. Nevertheless, the holder of the interest can ignore message which are retrieved outside such a *window of awareness*.

location (0..1) This facet is similar to the time facet. It can be used to describe constraints at which places a communication shall take place.

The seven facets are independent. A change e.g. in the peer facet has not necessarily an impact on other facets. From a mathematical point of view, the seven facets spawn a seven dimensional space which is called the **context space**. Defined facets are points on the axis of coordinates. Thus, each interest defines a subspace of the overall context space.

The term *context space* is derived from [Len98]. A context space has been used in the CYC project to split a knowledge base to make it processable. The CYC project tried to create knowledge base containing the overall knowledge of the world - often called world knowledge. DCS follows the concept of *microtheories* which is the diametric approach of building a worldwide knowledge base, see [Sow00].

1.2.3 Semantic Tags, Peer Semantic Tags

DCS uses **semantic tags** to define points on the seven axis of the context space. The concept will be motivated at first and defined afterwords.

One crucial problem in spontaneous networks of semantic peers is the *vocabulary*. *Anton* was interested in talking about *soccer*. Unfortunately, it is just an English word. Other peers would call it *futbol*, *Fussball* etc. pp.

Ambiguous words and names are a well-known problem even in Web 2.0: *Tagging* is used to describe the topic of e.g. an article. Tags are words. Words are ambiguous. *Java* could mean an island, a coffee or a programming language. Java is a homonym in this case: The same word denotes different concepts. There are also synonyms: Different words denote the same concept. Moreover, names can change. St. Petersburg in Russia was also called Leningrad, Petrograd and St. Petersburg. New York was formerly known as New Amsterdam, Chemnitz in (East-Germany) was called Karl-Marx-Stadt between 1953-1990 etc.. Multilanguage support can be seen as variant of handling synonyms. Spain is also called Espana or Spanien. The same country has different names in different languages.

Semantic Web deals with such problems. Topic Maps [GM05] (ISO 13250) defines a represenation format for knowledge. A central concept is the *topic* which was foundation of the *semantic tag* in DCS.

A topic represents a thing (a *subject*) in the world, e.g. the city Cordoba. Unfortunately, Cordoba is a homonym. There are cars which are named after this city. Topic Maps solves this ambiguity by means of *subject indicators (SI)*. A subject indicator is a reference to information (e.g. a text, a website) which describes the concept. An SI for Cordoba could be an entry in Wikipedia or a link to a Website of the city. Each topic can have an arbitrary number of SIs.

SIs are a kind of wordbook of the language. SIs describe the meaning of the topics. Topics are the vocabulary.

Wordbooks can even be mixed because each topic can define its meaning by referencing different sources. The page http://www.cordoba.es/ describes Cordoba as well as a page in the English Wikipedia[7]

Thus, a peer *A* that understands Spanish and English could use both SIs to define its topic for Cordoba. Another peer **B** might not speak Spanish but English and lets say German. It could use the same link to Wikipedia and additionally a link to a German website about the city. Finally, *A* could learn a German web page that described Cordoba and **B** could learn a Spanish Web page for the same topic.

ISO 13250 topics have some more features. DCS was designed to be as slim as possible. Therefore, a semantic tag has been defined to have two parameter: at most one name and an arbitrary number of SIs. A semantic tag is an enhanced Web 2.0 tag: It has a name like a tag but also additional subject indicators.

[7] http://en.wikipedia.org/wiki/Cordoba_Spain

Peers are described by semantic tags. Peer's SIs can be links to business and/or private homepages. It can be a unique number. It can be anything that is unique to a person.

Fig. 1.2. Peer semantic tag for Anton (example)

In DCS, peers must have another feature: They must have an address to which KEP message can be send. Therefore, the **peer semantic tag** was introduced. It is an semantic tag with name and SIs and with an arbitrary number of (ISO layer 2-7) addresses. There are also derived semantic tags for location and time but this is out of scope of this paper.

Semantic Tag Sets

The W3C standard RDF[W3C04b, W3C04a] uses a so-called triple as basis for a knowledge base. There are nodes which stands for concepts. One node can reference another node by means of a predicate. A RDF predicate is a directed, labeled reference from a node (which plays the role *subject* in this case) to the another node *object*.

A network of semantic concepts is called *ontology*. An ontology represents a domain of discourse and is used e.g. for automated reasoning.

DCS adopts the concept of RDF. A semantic tag can reference each other. A single reference type is defined in DCS: the super/sub relation. Other application specific reference types can be defined. There are three kinds of ontology (**semantic tag sets** in DCS speech) types:

Plain Semantic Tag Sets are sets of semantic tag which are not referenced at all.

Hierarchical Semantic Tags Sets are taxonomies: A tag can be *super* tag of other *sub* tags.

Semantic Net Sets allow arbitrary references between tags. The references are application specific.

1.2.4 Context Points and Learning

Information are stored in a knowledge base. The knowledge base has seven dimensions. Points of the axis are semantic tags. The axises spawn a context space. Points in the context space are called **context points (CP)**. A context point has seven coordinates. Information can be attached to context points. **Knowledge** is a set of context points with the attached information.

Lets make a (lengthy) example:

Assume a peer A is interested in space flight which is a topic. It can declare an interest:

```
peer: A, originator: any, remote peer: any, time: any,
location: any, direction: in, topic: space flight
```

Note, all these facets contain semantic tags. The names (e.g. *A*) are placeholders for the whole semantic tag.

Peer *B* might strongly belief that the US landing on the moon was a fake. *B* might also have collected "proofs" of this thesis which are stored in its knowledge base under the topic *landing on the moon* which is sub tag *of space flight*.

Another peer *C* might also be interest in the old moon program but strongly belief that astronauts were on the moon. It has pictures and old articles from the news etc. It has also put the whole stuff under the topic *landing on the moon* which is sub tag *of space flight*.

B and *C* might also described an interested in discussing *landing on the moon*.

What happens if *B* und *C* meet? Both find out that they are interested in the same topic. Both would exchange information. *B* would receive pictures etc. from the landing. Where would it be stored?

B would create a new context point with axis *topic: landing on the moon* and axis *originator: C*. It wouldn't interfere with its own information because *B's* information are stored with *B* on the *originator* axis. There are two different context points. They have the same topic but different originator coordinates.

The same would happen with *C*. What happens if *A* meets *B* after all? First of all, *A* would find out that there is a sub topic of *space flight* called *landing on the moon*. *A* can now decide whether to learn this new topic or not.

A could also retrieve knowledge about *space flight* from *B* with up to two context points. Both have the same topic coordinate: *landing on the moon*. The originator dimension would be different (*B* and *C*). *A* could now have two more context points. One holds documents that try to proof that nobody was ever on the moon. The other would hold historical information about this event.

It is just one scenario. It could be different. *B* could e.g. throw any document away which is from *C*. So could *C*. Thus, *B* would never tell *A* about C. *A* could refuse to talk to *B* or *C* at all...

This example illustrates some impacts of the concept:

- A single knowledge base can store *contradictions*. Yes, the world, is full of contrary meanings: What is false, what is true? It isn't clear very often. DCS allows to keep contrary documents in the same knowledge base but at different points in context space.
- Any peer is free to deliver whatever information it wants.
- Peers can learn from each other. Learning takes place in two ways:
  - Semantic Tags can be learned. *A* can learn that *landing on the moon* is a sub concept of *space flight*. It can enhance its topic dimension.
  - Peers retrieve knowledge which are context points and attached information. Peers can integrate (*assimilate*) retrieved context points into their knowledge base but they don't have to.
- Peers can adopt information. In the example above, *A* has no personal information about the *landing on the moon*. It knows what *B* and *C* think about it. Sooner or later, *A* could decide to join one side. *A* could say: Yes, I believe *B*. From DCS point of view, it could move (or copy) by changing the originator axis: If *A* replaces *B* in this axis with *A* it would change semantics: *A* would now state, that *A* beliefs in all documents which are attached to this CP. *A* beliefs in it. *A* does not claim to be the creator of the documents but it commits to the opinion.

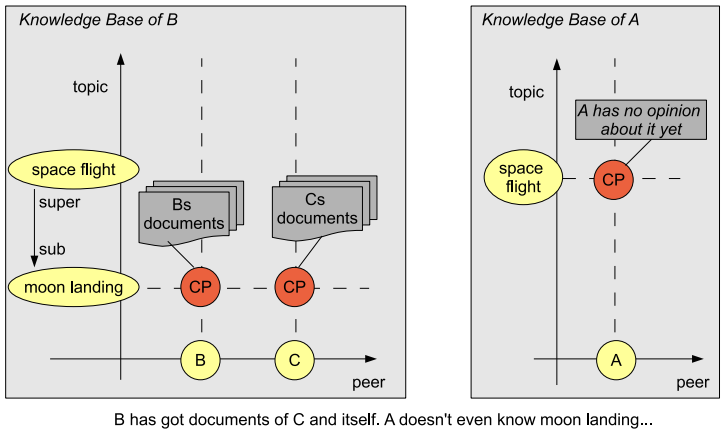

Fig. 1.3. Learning during knowledge exchange

1.2.5 Background Knowledge and Contextualization

This section gives a more formal and detailed view on the process of interest and knowledge exchange. In this chapter, the interaction between the user

and the software is described in more detail. In the remaining paper, peers are understood as a both - the owner of the system and his/her software. Only in this section an explicit distinction is made between the software (DCS engine) and its user.

User A can define an new interest with its DCS engine. This can be done by denoting semantic tags which are already in the knowledge base. This tags are called **anchor**. User A can furthermore define **fragmentation parameter** for each dimension. This is done by each dimension of the context space. The newly created interest is called a **local interest**. It contains anchor and fragmentation parameter for each dimension.

A **remote interest** is calculated by means of local interest and the current knowledge base. A **contextualization**[8] algorithm is used. Principles of the algorithms will be explained now. At first an empty context space is created which is called *remote interest*. It has seven dimension but has no entries. Note, no entries means *anything*. A non specified interest covers all thinkable context points. The following algorithm shrinks makes this unspecified interest more specific.

1. A list of semantic tags is created. It contains all tags which are in the anchor set and also in the knowledge base. All tags in this set are added to the set in the remote interest.
2. Fragmentation parameter contain two parameters: A non negative number called depth and a list of allowed association types. In a second step the **corona** of each anchor is created. The corona contains all semantic tags in the dimension which can be reached via allowed association types in the semantic tag set an which have a distance not longer than *distance* to the anchor. The corona is added to the remote interest.

The center (or seed) of each corona are the anchor tags. The corona is also called **background knowledge**. Remember the previous example: B was interested in *landing on the moon* which is a sub tag of *space flight*. User B could mark *landing on the moon* as anchor tag. B can also define fragmentation parameter, e.g. B could define allowed association type as *super type* and the depth might be one. Now, the corona would have a maximum diameter of one and only tags would be added which have a super relation to the *landing on the moon*. The corona consists only of *space flight* in this case.

This process can be formalized: $K * IL = I$. Knowledge base K is contextualized with local interest IL. The result is I. I is a remote interest an can be exposed to another peer.

Lets say B gets an interest of A (I_A). B has also a remote interest (I_B). B has received A's interest a tries to find out, if both have a mutual interest.

[8] The termin *contextualization* was already used and probably invented by Theodorakis[The99]. He developed a mathematical model to split a knowledge base. There are hardly similarities between his concept and DCS contextualization.

This can also be done by contextualization: $I_{BA} = I_A * I_B$. All tags in I_B are interpreted as anchor tags and the fragmentation parameter are assumed to be null. The algorithm find concepts which are of interest for both sides. I_{BA} is also called the **effective interest**[9].

An interest is a sub context space. A's interest as well as B's interest is an sub context space. The effective interest in the intersection of both.

If B - the recipient - is willing to send knowledge it can extract it from the knowledge base. The sub space can calculated by contextualization: $K_{BA} = K_B * I_{BA}$. The knowledge base is contextualized with the effective interest. In other words: All context points are found which are in the intersection of the interest of A and B. The result is knowledge which can B send to A (K_{BA}). It is also called the **exchange knowledge**.

A receives K_{BA} from B. KEP is an asynchronous protocol. A might not remember the fact that is has already send an interest. Moreover, the A's interest can be changed over time. DCS Engine A should now take the knowledge and has to find out what to insert into its own knowledge base. This can be done by its own interests:

$K_{assimilate} = K_{received} * I_A$

Any received knowledge is contextualized with an interest. The result is knowledge which is of interest for the recipient and can be added to the local knowledge base.

1.2.6 Interests - a knowledge layer addressing schema

The ISO-OSI reference model defines seven protocol layers for open systems interconnection. Each layer has its own addressing schemes and protocols. OSI uses the term communication endpoint which is an entity that actually performs the layer n protocol. Endpoints on the same layer communicate. Nevertheless, the actual layer n protocol data units are wrapped into layer *n-1* protocol data units and send over a *n-1* protocol which is wrapped into a *n-2* layer data unit etc. pp. etc until they have reached layer 1. The physical layer actually transmits data.

KEP is a layer seven protocol. KEP addresses are interests. Interests describe who (peer) wants to communicate with whom (remote peer) about what (topic and originator), when (time), where (location) and in which direction. Interest describe a sub context space.

A DCS user defines its interests with its local context space. It uses a vocabulary which can potentially be understood even be unknown peers due to the semantic tags. Interests are an addressing schema that fit to loosely coupled spontaneous semantic networks. Even peers can bind which never meet before: DCS is an open system.

The context space concept allows describing the whole communication context in a semantic and thus less ambiguous manner. Moreover, the knowledge

[9] Note, contextualization isn't a commutative operation.

base makes the communication context persistent. Actually, the knowledge base is a persistent local context space owned and managed by a single peer.

1.3 Related work

Multidimensional addressing schemes are not new. Distributed Hash Table (DHT) is a multidimensional addressing principle for P2P systems which is e.g. used in Chord[Sto03]. A hash function creates n numbers for each information which has to be stored in the systems. Each number becomes the coordinate of the information. Each DHT peer covers a subspace in this n dimensional space. DHT allows to calculate which peer stores information with a given hash code.

DHT and DCS differ fundamentally. DHT peers are managed by the overall system. They get a subspace when entering the system. They have no choice - they have to store and restore information when asked. The n dimensional DHT space is based on hash codes and not on the semantics neither of the content nor of any communication constraint.

Kademlia[MM02] and BitTorrent[10] are other protocols which organize replicates and optimize retrieval of information in P2P system.

DHT, Kademlia, BitTorrent and other related P2P approaches are means to organize information. Peers are governed by the system. They store and restore information at command.

Autonomous DCS peers are autonomous entities, though. They store and offer what they want and they can describe when interest or knowledge exchange is allowed. DCS doesn't care about copies and replicated information like DHT does. Nevertheless, replications strategies *can* be build with DCS.

Diaspora[11] is a P2P social network. It is a Web application but the Web server runs locally. The P2P communication connects the peers. It is the same principle: Peers store their own data and communicate directly if they want. Diaspora has all drawback of the Web 2.0 tagging that was already defined. DCS is uses a semantic addressing schema and supports arbitrary underlying protocols.

Shark is a reference implementation of DCS. Shark hasn't the maturity of Diaspora yet. It is in an alpha version yet. But Shark based applications work on a broad range of systems: PCs, Android, J2ME, sensors which communicate via Bluetooth, TCP, HTTP and E-Mail.

Sowa [Sow00] made a general distinction between two general approaches to understand a distributed knowledge base. The first approach is called *world knowledge*. Each knowledge in the world is conceptually seen as a part of an overall knowledge base. There is a wish that sooner or later all contradictions are resolved and the overall knowledge base is consistent and finished. This

[10] http://www.bittorrent.com/
[11] http://www.joindiaspora.com/

was the dedicated aim of the CYC project. Some Web 2.0 enthusiasts seems to think the same and Wikipedia and / or similar systems would became the place where the final knowledge will be stored and the end of all times would have been reached, see also [Lan10].

The opposite approach is called *micro theory*. It states, that knowledge bases (KB) are independent entities which evolve. KBs can exchange (parts of their) micro theories (knowledge). There is no overall KB but a number of independent and communicating KBs.

There is no trend toward a reduced number of contradictions. Contradictions are part of the knowledge creating process. DCS follows the idea of micro theories.

1.4 Shark - Shared Knowledge

First ideas of Shark were published in 2002 [Sch02, SG02]. It was more or less a theoretical sketch. The concept of a multidimensional context space was developed between 2004 and 2005 [Sch04, MS05]. The seven dimensions and the structure of interests are new results in the project and hasn't been published.

In 2008, we have started an open source project (*Shark Framework*) which implements the DCS concept[Sch08].

Shark implements the DCS knowledge base in multiple ways: There is an in memory implementation as well as an implementation above a pure file system. It works on any Java device. Next steps will be an integration of Android SQLLite database and RDF knowledge base (Jena framework). A concept of mapping to the ISO Topic Maps is already published [Sch09b].

Implementation details of Shark cannot be discussed in this paper. But the principles of writing an Shark based application can be illustrated. Shark is written in Java. Currently, J2SE, J2ME and Android is supported. Objective C support is planned.

1.4.1 User Interfaces

There is an SharkKB interface defining methods of the database. Shark developer can manage context points as well as each dimension in the local knowledge base. the following lines illustrate just a few methods for CP management. Have a look in on the sourceforge page for more details.

```
ContextPoint getContextPoint(ContextCoordinates coordinates);

ContextPoint createContextPoint(ContextCoordinates coordinates,
String[] dimNames);

void removeContextPoint(ContextCoordinates coordinates);
```

```
void assimilate(Knowledge k, FragmentationParameter[] otps,
LocalInterest effBackground);

Knowledge extract(LocalInterest subSpace,
FragmentationParameter[] otp,
FragmentationParameter[] fp);
```

The code fragment above contains declarations of the methods extract and assimilate. Both are based on the contextualization algorithm which was discussed in 1.2.5. Thus, Shark application developers don't have to deal with search function in complex ontologies.

Actually, application developers can use predefined knowledge base implementation and can focus on the application specific logic.

1.4.2 Knowledge Ports

An interest is created by means of the Shark knowledge base with the following method.

```
LocalInterest createInterest(AnchorSet as,
FragmentationParameter fp[]);
```

A local interest is just a data structure. A **Knowledge Port (KP)** is the KEP protocol engine - the communication endpoint in OSI speech. It takes a local interest, calculates the remote interest (as described in 1.2.5) and waits for incomming request. It also observes the environment for new peers. In can establish a connection for the first contact with a new peer.

Connection establishment etc. is managed by the framework. Application developers can use the business logic of predefined KPs. They can also define their own business logic. In this case, a new knowledge port class must be derived from the `class KnowledgePort`. Two methods must be implemented:

```
void doInsert(Knowledge k, KEPResponseFactory respFact);

void doExpose(ReceivedInterest i, KEPResponseFactory respFact);
```

This API looks (hopefully) similar to the Servlet API. There are two KEP commands, *expose* and *insert* which can be retrieved by an engine. It can either get an interest or knowledge from another peer. The respFact (response factory) is to create a KEP response. The underlying protocols are hidden from application developers. Actually, a request can come in with Bluetooth and the KEP response is send with TCP. This aspect is discussed in section 1.5.

1.4.3 Protocol binding

KEP is a layer 7 protocol and uses other protocols to actually be transmitted to other peers. There must be a protocol binding defined for each protocol. Currently, bindings are defined for TCP, Bluetooth (partly: some BT profiles), HTTP and E-Mail (POP/SMTP).

The framework is open for new protocol bindings. There is a concept of **protocol stubs** in Shark. It is simply an interface which has to be implemented. The interface for connection oriented protocols shall be shown to illustrate the idea.

```
public interface StreamStub extends Stub {
    public StreamConnection createStreamConnection
           (String gcfAddr) throws IOException;

    public String getLocalAddress();
    public void stop();
}
```

There must be a way to establish a StreamConnection which is a combination of `InputStream` and `OutputStream`. Each protocol stub must understand the Java Generic Connection Framework (GCF) addressing schema. Shark uses this schema to define addresses e.g. in the parameter `gcfAddr`.

The local address is the address of the local endpoint of the protocol, e.g. the TCP server name including the TCP port. The String must be a GCF address. Method `stop()` asks the protocol engine to stop which means that it shouldn't accept no further messages.

A Shark engine can have an arbitrary number of protocol stubs. A singleton, the KEPStub, manages the underlying specific protocol stubs. KEP messages are send firstly to the KEP stub. It checks the recipient address and forwards the message to the appropriate protocol stub. This makes the protocol specific adoptions and sends the message. Incoming request are retrieved by a protocol stub and send to the KEP stub. The KEP stub delivers valid KEP messages to all active Knowledge Ports which perform the business logic. KEP responses are send through the KEP Stub and so on.

1.5 iSphere - a Shark based social network

ISphere is an application based on Shark. There are two version: The full version runs on J2SE and Android. A reduced version runs on J2ME devices. Shark runs an each platform but the reduced version became necessary due to the limited screens on J2ME phones. Neither the datastructure nor protocols created a problem - it is just the GUI. This underlines the thesis in section 1.1 that memory and bandwidth are no limiting factors nowadays.

ISphere illustrates the component based character of Shark. There is a core which includes all DCS algorithms and the knowledge base. It is written in Java 1.3 compliant code and runs on all Java platforms. Platform specific protocol implementation became necessary due to the different protocol APIs in J2SE, J2ME and Android. These implementations are now part of the Shark framework and must not be reimplemented in further projects.

ISphere specific business logic had to be implemented. It was implemented in Java 1.3 compliant code again and is cross platform compatible. This fact should be coined out: Application specific code should be written in Java 1.3. It runs on each Java platform. Shark hides platform specifics from Shark application developers.

Only developers who work inside the Shark framework has to create platform specific code. Application developers don't have to.

There are three different GUIs, though. They had been written from the scratch with different technologies. It was also to proof the concept of the framework. The PC GUI is a Web application based on the wicket[12] framework. The PC shark engine is started with a web server. Users interact with a Web browser with the local knowledge base. The Android GUIs has been written with no additional tools, J2ME used the MIDP high level GUI API.

PC and Android GUIs are strongly inspired by existing social network app. Users see their own picture and can define their interests and profile information. The mapping is different to ordinary Web 2.0 apps of course:

Each user is represented by a peer semantic tag. It contains a name (that is only valid in the local knowledge base), a set of subject indicators e.g. referencing a Web pages of the user and an arbitrary number of addresses (addresses of Shark engines in GCF style).

Personal information like private telephone number, address of the user, picture etc. are stored in the knowledge base with a context point. There is a iSphere specific *topic* called *profile*. An user A stores its profile with a context point with the coordinates (peer/originator: A, topic: *profile*).

Each user can define who is allowed to get her/his personal profile. This can be done by a a list of remote users or by naming a user group. User groups can defined locally in the knowledge base. User groups never leave the local knowledge base.

Interesting topics can be defined in the topic dimension in the local knowledge base. Semantic tags are used. Topics are not visible for other peers without an interest for communication.

Anything that should propagated to other peers must be described in one or more *interests*. Defining an interest is pretty simple: Users pick topics of interests and define communication partner. They can also declare time and place for an allowed communication and if they want to send or retrieve data.

Publishing data requires three steps:

[12] http://wicket.apache.org/

1. Information (semantic tags and if wished information and context points) must be created in the local knowledge base.
2. Interest define a context in which a communication shall take place. Knowledge ports contain the application logic. They are activated with an interest. Note: A knowledge port can also be passive. It can just wait for incoming requests. Thus, it is possible to define a communication interest which isn't visible to other peers. Other peers cannot *see* anything from the local knowledge base. They are only aware of the fact, that there is a peer. They can send a KEP message or not. Such interest are called **silent** or **hidden** interests.
3. User can also decide to (*expose*) an interest. In this case, other peer can *see* the interests of a peer and not only the peer. The interest becomes **visible** in this case.

ISphere can be configured. It can be decided what communication protocols shall be used. In this paper, just one configuration shall be explained. It shall illustrates the multi-protocol approach of Shark.

PC based engines communicate with E-Mail (POP/SMTP) with arbitrary remote peers. There is a fully fledged GUI. User can define interests. Users address in the peer semantic tag is an E-Mail address.

PC based engines can communicate via Bluetooth with J2ME phones. In this configuration, it is assumed that each iSphere user has a J2ME phone and an PC (or Laptop). There is a special knowledge port on both sides which synchronizes both knowledge bases. Thus, when leaving, both KBs are in sync.

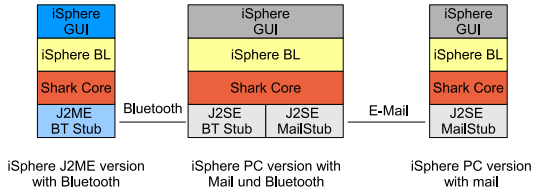

Fig. 1.4. iSphere versions

The J2ME devices communicate via Bluetooth. Thus, these little mobile devices can exchange interests and knowledge as any other Shark engine but users cannot change the knowledge base. Screens are to small. But knowledge bases are synchronized with the PC again. Now, newly retrieved knowledge can be studied, thrown away or can be taken as a new part of the local personal knowledge.

Moreover, a mobile device can also run in a silent mode (or *harvesting* mode) which means it only collects interests from remote peers but doesn't send anything. After synchronization, the PC performs the interests and probably contacts other peers.

1.6 Summary and Outlook

Lanier is right[Lan10]. A serious number of Net application programmers are trapped in a *Lock In*. The only natural way of writing a distributed application seems to be a Web based application. This isn't true.

ISphere proofs that social network applications can be implemented without a server. Shark also demonstrated that subject indicator from Topic Maps is an appropriate way to create a common vocabulary for P2P based loosely coupled mobile networks. Shark also proofs that the concept of DCS can be implemented. Actually, this wasn't very surprising. Developers can implement nearly any algorithm - it's just a matter of time.

More important was the fact that Shark also proofed that the necessary complexity of DCS can be hidden inside a framework. Simple Shark based applications can be written without any understanding of semantics or multidimensional spaces or all the theoretical background. Only the concept of tags and URLs must be understood. Writing Shark based applications is as simple (or complicated) as writing a Servlet.

The proof of concept exists. Shark is an alpha version, iSphere is a prototype. The Shark project is sponsored by the German Ministry of Education and Research. The project runs until 2012. At the and of the project, there will be a released and well documented Shark framework. There will be an iSphere tutorial which describes step by step how the social network app has been build. Other Shark sub projects has been started recently. We work on a system for location based information, a system for exchange of huge data like video streams, a sensor based system and a distributed e-learning system. All projects are based on Shark. All systems work without a server. The concepts are already there. Shark stands for Shark Knowledge and it is open source: Join us.

References

GM05. Garshol, Lars M. ; Moore, Graham: ISO 13250-2: Topic Maps – Data Model / ISO/IEC JTC 1/SC34. 2005. – Forschungsbericht

Lan10. Lanier, Jaron: *You Are Not a Gadget: A Manifesto*. Knopf (Publisher), 2010. – ISBN 978–0307269645

Len98. Lenat, Doug: The Dimensions of Context-Space / CYC CORP (www.cyc.com). 1998. – Forschungsbericht. – http:www.cyc.com/context-space.rtf.doc,txt

MM02. Maymounkov, Petar ; Mazires, David: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In: *Lecture Notes in Computer Science* 2429/2002 (2002), S. 53–65

MS05. Maicher, Lutz ; Schwotzer, Thomas: Distributed Knowledge Management in the Absence of Shared Vocabularies. In: *Proceedings of the 5th International Conference on Knowledge Management (I-KNOW '05)*. Graz / Austria, July 2005

Sch02. Schwotzer, Thomas: Context Driven Spontaneous Knowledge Exchange. In: *Proceedings of the 1st German Workshop on Experience Management 2002 (GWEM02), Berlin*, 2002

Sch04. Schwotzer, Thomas: Modelling Distributed Knowledge Management Systems with Topic Maps. In: *Proceedings of the 4th International Conference on Knowledge Management (I-KNOW '04), Graz*, 2004, S. 52–59

Sch08. Schwotzer, Thomas: *Shark Framework*. `https://sourceforge.net/projects/sharkfw`. Version: started 2008

Sch09a. Schwotzer, Thomas: A Mobile Spontaneous Semantic P2P System. In: *Proceedings of the IEEE International Conference on Communication Technology and Applications 2009 (ICCTA 2009), Beijing / China*, IEEE, 2009

Sch09b. Schwotzer, Thomas: Multidimensional Topic Maps. In: *Proceedings of the International Conference on Topic Map Research and Applications (TMRA'09), Leipzig*, Springer, 2009

SG02. Schwotzer, Thomas ; Geihs, Kurt: Shark – a System for Management, Synchronization and Exchange of Knowledge in Mobile User Groups. In: *J.UCS* 8, Issue 6 (2002)

Sow00. Sowa, John F. ; Kallie Swanson, Mike S. (Hrsg.): *Knowledge Representation: logical, philosophical and computational foundations*. Brooks and Cole (Thomson Learning), 2000

Sto03. Stoica, R.; Liben-Nowell D.; Karger D.R.; Kaashoek M.F.; Dabek F.; Balakrishnan H. I.; Morris M. I.; Morris: Chord: a scalable peer-to-peer lookup protocol for Internet applications. In: *Transactions on Networking* 11 (2003), S. 17–32

The99. Theodorakis, M.: *Contextualization: An Abstraction Mechanism for Information Modeling*, Department of Computer Science, University of Crete, Greece, Diss., 1999

W3C04a. W3C: *OWL Web Ontology Language Overview*. `http://www.w3.org/TR/owl-features/`. Version: 2004

W3C04b. W3C: *Resource Description Framework (RDF)*. `http://www.w3.org/RDF/`. Version: 2004