Building Context Aware P2P Systems with the Shark Framework

Thomas Schwotzer
FHTW Berlin (University of Applied Sciences)
thomas.schwotzer@fhtw-berlin.de

Introduction

Knowledge management is an inherently distributed process. Knowledge is not created in a company, a think tank or whatever. Knowledge is initially created in an individual's mind. Usually people decide to share knowledge. Such newly created knowledge will usually be discussed in groups. It will be exchanged, combined, modified and maybe forgotten. Knowledge always flows in and between groups and between individuals. Groups can be official organizations, e.g. companies but also ad hoc groups or other not official networks of people.

Knowledge management systems (KMS) must support knowledge sharing. Topic Map is a knowledge representation standard. It supports knowledge sharing. The standardized *merge* operation defines how knowledge from different sources can be integrated. Topic Map Query Language but also Topic Map API can be used to take parts (fragments) from Topic Map. Such fragments can be exchanged between Topic Maps. There are some examples of distributed Topic Map applications.

Thus, there are means to combine knowledge but also to take parts of knowledge out of an existing knowledge base that is based on Topic Map. Developers of distributed knowledge management systems (more specific: distributed Topic Map systems) need additional functionality. A protocol for knowledge exchange is required and application must decide if and what parts of knowledge are allowed to be exchanged. Currently, this functionality must completely implemented in the application. Of course, there is no need to implement any marshelling / serialization code line by line. Middleware systems, Web services etc. pp. can be used. Nevertheless, the the whole exchange logic is application specific.

Shark framework is an open source project. It provides a stateless P2P protocol called KEP (Knowledge Exchange Protocol) and an API for building distributed P2P applications. Shark is independent from a specific knowledge representation format. The Shark concept has its roots in Topic Maps and therefore most core concepts and ideas are inspired by and complaint to Topic Maps. A core concept of Shark is the Knowledge Port (KP). A KP can be compared to a TCP or UDP Port. It is – hopefully – as easy to define and open as a socket in e.g. Java. Knowledge ports exchange knowledge particle (more specific: Topic Map fragments if Topic Maps Engines are used). KPs contain the KEP protocol engine and form the interface between the P2P protocol and the used knowledge base.

Aim of this paper is twofold. First, the Shark framework shall be introduced. It will to be seen that an useful P2P communication can already be defined by just a few lines of code.

Second, it is a call for participation. There is already a theory on Shark, see e.g. [SG02][MS05] and referenced paper. The Shark framework is new, though. Currently it is written in Java and runs with J2SE and J2ME and supports UDP and Bluetooth L2CAP. There are just a few applications. Shark will be ported to compact .NET, Android and maybe to iPhone. Sourcecode is available under sourceforge [SharkFW].

Building Distributed Applications with XTM, TMQL, TMAPI etc.

Topic Maps comprise a whole family of formats and languages. Some of them are standardized. The ISO Topic Maps standard [TM] describes data model and its semantics. It is basis of the following formats and standards. XTM [XTM] is a XML schema for Topic Map representation. It is part of [TM]. Topic Map Query Language [TMQL] is used to retrieve parts (fragments) of a Topic Map based on a query. Finally, TMAPI is an (not standardized) API for management of Topic Maps. We have everything what's to implement a Topic Map application. The following figure illustrates the relationship between components of a distributed TM application.

The application specific code is on top of the diagram. It uses TMAPI or TMQL to access and manipulate the underlying Topic Map(s). The Topic Map itself is stored in a component that is called *Knowledge Base*. There are no constraints how Topic Maps are actually stored. The TMAPI and TMQL implementations hide KB specifics from TM applications and its developers.
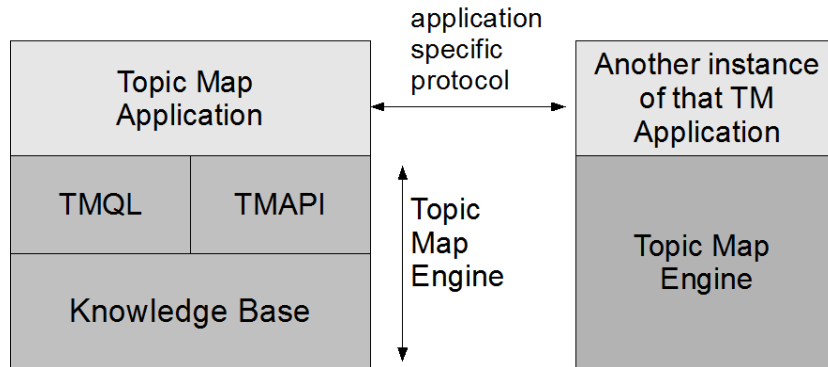

Fig 1: Components of a Distributed Topic Map Application

There is no explicit support for building distributed applications with Topic Maps. Application developers are free to use arbitrary network protocols to e.g. exchange Topic Maps with XTM or TMQL queries and their results. The communication issues

are implemented in the application itself. Thus, it is an applications specific protocol that enables communication between remote Topic Map Engines. TMShare [TMShare] is an example of such a distributed Topic Map application.

P2P applications

P2P applications are a special class of distributed applications. There is no common definition of peers. In [Sc08] a model of autonomous context-aware peers are proposed – the ACP model. The basic ideas of the model are straightforward. Peers have the following features:

- A peer has its own knowledge base. There are no constraints on knowledge representation formats used by the knowledge base. Of course, in this context a knowledge base can be assumed to be a Topic Map Engine

- A peer observes its environment. Changes of the *environmental context* are recognized and can lead to an activity, e.g. delivery of a message, change of internal status.

- A peer can send messages to other peers in its *environment*. The definition of environment remains very vague in the ACP model. It can be a local area network but also the WWW.

- Peers have (not necessarily unique) identifiers.

- Peers are autonomous. They can autonomous decide under which circumstance (based on current environmental context, current connections to other peers, already exchanged message, status of the knowledge base etc. pp.) messages are send to other peers and what information are delivered.

Whenever peers take notice of each other they can decide to exchange messages and finally to exchange knowledge. Two processes have to be distinguished. Knowledge e*xtraction* is the process of taking knowledge from peers knowledge base in order to send it to a remote peer. Knowledge *assimilation* is the process of retrieving knowledge and (partially) integration in the local knowledge base.

More formal, both process can be defined as functions (in a Java like syntax):

Knowledge extraction(recipients, environmental context, status);

void assimilation(sender, environmental context, status, knowledge);

Extraction generates a knowledge particle (in this context a Topic Map fragment). Extraction is influenced by identity of the potential recipient, current environmental context and the status of the peer. Assimilation integrates (parts of) retrieved knowledge. This process is influenced by the senders identity (if known), current environment and internal status.

There are, deliberately, no algorithms defined for any of the functions in ACP. This is up to an ACP implementation. The simplest implementation of assimilation is a Topic Map *merge*. The easiest implementation of extraction would be a usage of a

static TMQL query. Both implementations would ignore the environment and identity of the potential communication partners and would lead to a kind of distributed Topic Map but not to a network of autonomous context aware peers.

Shark Framework – an implementation of ACP

The Shark Framework [SharkFW] is an implementation of the ACP model. This supports implementation of autonomous peers which can exchange knowledge in described manner. The framework is written in Java and is currently available for J2SE and J2ME. Launch of version 1.0 is scheduled September 2008. It is an extensible open framework with only a few requirements for underlying knowledge bases and used communication environment. A knowledge base must implement function extract and assimilate. An environment must allow sending and retrieving of message and should optionally be able to recognize changes (e.g. appearance of peer). Version 1.0 comprises a UDP-Environment and a Topic Map with J2SE and a Bluetooth-Environment and a very simple knowledge base based on J2ME.

Main features of the Shark core are an API for autonomous peers and implementation of a protocol engine supporting the P2P Knowledge Exchange Protocol (KEP).

Knowledge Exchange Protocol (KEP)

KEP is the Shark specific P2P protocol. There are four KEP commands. KEP was influenced by software agent protocols [KQML], [ACL]. In the following the four KEP commands will be briefly described.

- The *interest* command is submitted by a peer to indicate its *retrieving interest*.
  A peer can define what kind of information is willing to retrieve information / knowledge. In Shark, this definition is simply be done by naming a number of topics. Of course, XTM or LTM are preferred representation formats.

- The *offer* command is submitted by a peer to indicate its *sending interest*. A sending interest is the counterpart of a retrieving interest. A peer describes kinds of information it is willing to send.

- The *accept* command is similar to the interest command but with slightly different semantic. Accept delivers a retrieving interest. Sender of an accept command expects to get a knowledge particle in reply.

- The *insert* command submits a knowledge particle, e.g. a XTM document. A sender will *extract* a fragment from its local knowledge base and send it with a KEP *insert* command to (a) recipient(s). Recipients will *assimilate* retrieved knowledge. Both, extraction and assimilation algorithm is application (class) specific.

KEP is a stateless protocol. There is not even an implicit defined order of commands. Thus, KEP can easily be implemented with UDP, Bluetooth L2CAP and other datagram protocols.

Each KEP command contains the name of the sender (or *anonymous*) and names of potential recipients (or *anonymous*). There are some common usages of KEP sessions.

KEP scenarios – Internet peers

In the first scenario it is assumed that two peers with huge knowledge bases can establish a stable communication channel e.g. an TCP based connection in the fixed Internet. In the first step both peers can negotiate a *mutual interest*. This can be done by an exchange of interest/offer messages.

A example will explain the approach. Peer A has information about latest music bands and movies. Peer B may be interested just in music. Thus, A would describe its *sending interest* with *music, movies*. (Note, this is an abbreviation. *Music* should be read as e.g. a topic standing for the concept of music. The string music can be a basename of this topic.). B would describe its *retrieving interest* with *music*. Now, A could send a *offer(music, movies)* command to B or B could send a *interest(music)* command to A.

If A retrieves an *interest(music)* it can decide if and what to offer to B. In this example it would probably send an *offer(music)* to B. If B would retrieve an *offer(music, movies)* from it would learn that A has music information an would probably reply with *interest(music)*. At the end of both sequences, B knows that A offers music information. A knows that B is interested in music. Music is of *mutual interest*.

Now, B could send *accept(music)* to A. A would extract music information and send *insert(musicKnowledge)* back to B. Alternatively, A wouldn't wait for an *accept* and directly send a *insert* command.
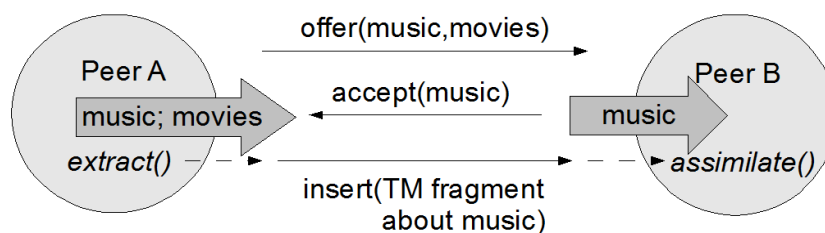

Fig 2: Knowledge Exchange with Shark's Knowledge Ports

KEP scenario – Peers in spontaneous networks

Spontaneous networks are networks of mobile nodes which tend to enter and leave a network frequently. Moreover, a peer that was in a spontaneous network cannot be assumed to enter it again. A knowledge exchange strategy must be adopted to these characteristics.

A spontaneous network could e.g. be a network of two Bluetooth applications e.g. running on mobile phones. It takes several seconds to establish a connection. Bluetooth mobile phones can usually communicate within a radius of 10 m. Imagine two pedestrian (~ 3 km/hour and their mobile phones in the jacket) would pass by. As soon as the distance is smaller than 10 m a spontaneous network could be established. The BT channel is of course dropped as soon as the distance is again over 10m. In this example, both mobile peers would have 12 seconds to establish a connection and to exchange information. Establishing a BT connection can already take about 10 seconds. There is no time for lengthy negotiations.

Another strategy should be used: Whenever a peer "sees" another peer in a mobile environment it should try to send relevant information as fast as possible. It could either send a (small) knowledge particle or an (retrieving or sending) interest with a different (e.g. IP or E-Mail) address for replies.

With the first strategy mobile peers would frequently get unsolicited insert commands. They would examine these knowledge particles and maybe assimilate parts of it.

With the second strategy mobile peers would just exchange their interest and addresses to longer lasting peers, probably Internet peers. Such strategy is useful in environments which combine mobile and fixed peers.

Peer API / Knowledge Ports

Shark Frameworks supports development of KEP based P2P systems. The following example code illustrates how a peer providing music information can be created.

```
Peer p = new Peer();

p.getKnowledgeBase().
        addKnowledge("music","new album from madonna", "music news");

KP okp = p.createOKP("music");

okp.setVisible();
```

The first line creates a Peer. New information is added to the knowledge base in the second line. Information contains of three parts, topic ("music"), creator ("music news") and informations itself ("new album from madonna"). Note, this is also just an example and illustrates knowledge base access by Shark. If a Topic Map Engine is used the code could be changed like this:

```
Topic Map tm = (Topic Map) p.getKnowledgeBase();
```

```
// do TM specific things, e.g. based on TMAPI or TMQL
```

The third line creates a knowledge port (KP). There are two knowledge port classes: incomming and outgoing knowledge port (IKP / OKP). An IKP is an object representing a set of information for assimilating information. An OKP is an object holding information for extraction process. The function above is just a convience function. The general KP constructor is defined as follows:

```
KP(KnowledgeBase kb, Peer peer, Context ctx, Context interest, PeerName
peers, boolean ikp, boolean okp)
```

The kb is the knowledge base which will extract or assimilate knowledge. The peer is the sending peer, ctx describes requirements for the environment, interest is either a sending or an recieing interest and peers describes names for potential communication partners of this port: Finally, two boolean values allow to define a knowledge port as IKP or OKP or both.

The convenience function in line 3 actually creates an OKP using the main knowledge base of the calling peer, the calling peer as sender, defines no constraints on an environmental context, defines just a single topic as interest ("music") and allows to communicate with any peer that will be detected.

Last line makes this newly created OKP visible. Depending on the used environment, the KP will be e.g. published in a service directory and/or a broadcast is sent into the spontaneous network etc. pp.

Defining an IKP is as simple as defining an OKP:

```
Peer p = new Peer();
```

```
KP ikp = p.createIKP("music");
```

Both peer will be ready to exchange knowledge after both code fragments has been executed. A KEP protocol session will be performed whenever both peers can establish a communication channel. Knowledge will be exchanged if mutual interest can be negotiated

As described above there are several KEP strategies. In version 1 just two are supported. Both have been described above. Default is the full negotiation. The KEP strategy of a knowledge port can be changed with following command.

```
void kp.setStrategy();
```

Shark Engine

The Shark Framework is an additional layer above a knowledge base, e.g. a Topic Map Engine and the application code. It provides a P2P protocol which is designed for loosely coupled systems, namely spontaneous network but which also works on top of UDP or TCP in IP based networks.

The example above illustrated that e.g. four lines of code are sufficient to create a peer, enter sample data and to open a port for knowledge exchange. Just two lines of

code are needed to define a peer that is interested in getting information about music. Shark hides P2P communication protocol as well as establishing a communication channel to other peers.
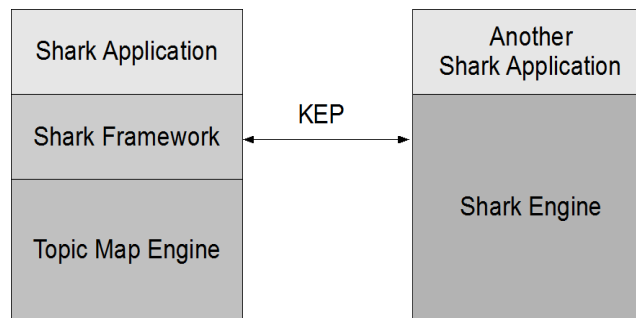

Fig 3: Shark Application Components

The figure above also illustrates a feature of most frameworks for distributed systems. Shark Framework already provides a protocol. The application code does not have to deal with protocol specific issues. It just has to define rules for knowledge exchange.

Application independent protocols are an advantage in general: An application specific protocol can potentially change whenever the application is changed. A Shark application isn't even able to change the KEP protocol at all. It can just handle received knowledge or interest in different ways.

Shark peers versus software agents

Shark peers have something in common with software agents which are used in the field of distributed artificial intelligence. Nevertheless, there are major differences: Software agents are meant to be entities that can fully replace human users in an dedicated application domain. Agents can simulate plans, strategies of human users as well as (in a reduced an limited manner) feelings and biases.

Shark peers are just container holding information and algorithms for knowledge exchange. Shark peers are parametrized and run on behalf of human users but they would and could never be seen as a *substitute* of a human user. From a very abstract perspective Shark peers can be compared to an intelligent filtering system but not to a replacement of personal strategies.

Shark peers versus distributed systems

A P2P system is a distributed system. A system based on Shark is a distributed system. The concept of autonomy makes it different from e.g. file exchange systems, music exchange platforms. A Shark peer decides (based on its algorithms) if and what kind of information shall be exchanged. In other P2P system users browse through collection of information and decide what to download. Peers are passive entities which makes their local information bases accessible to remote peers.

There are distributed systems that hide distribution. Distributed databases combine several databases and present it as a single virtual database to software developers and users. The distribution is hidden. Middleware systems like CORBA, EJB etc. are also hide distribution. Shark doesn't. Developers and users are aware of the fact of distribution. Thus, Shark can and should only be used for applications which are not meant to hide the fact of distribution.

Summary and outlook

The Shark framework is an implementation of the model autonomous context aware peers. It is an open framework. Shark core has just very weak assumptions on knowledge base features. The Shark protocol KEP is stateless and can easily be implemented on top of datagram protocols like UDP and Bluetooth L2CAP. Currently, Shark is implemented in Java (J2SE and J2ME). Nevertheless, Shark is far from being finished. Even version 1.0 can just be seen as a very first step.

Mobile P2P systems should support a broad range of hard- and software. In the next steps Shark will be ported to Google's Android and to Apple's iPhone. Furthermore, applications are needed to proof the concept and to give input for further revisions of the framework. This paper shall also be understood as a call for participation. Shark is published under LGPL in sourceforge. Shark is an acronym. It stands for Shared Knowledge. Let's share Shark!

Literature

[TMAPI]   *Topic Map API*: http://www.tmapi.org/

[TMQL]   *ISO/IEC 18048: Topic Map Query Language*.
            http://www.isotopicmaps.org/tmql/

[KQML]   Finin, T., Weber J., Wiederhold, G., Genesereth, M. Fritzson,R., McKay, D., McGuire,J., Pelavin, R., Shapiro, S., Beck, C.: *Specification of the KQML Agent-Communication Language - plus example agent policies and Architectures*; June 1993

[TMShare]   Ahmed, Kal: *TMShare – Topic Map Fragment Exchange in Peer-to-Peer-Application.* In: Proceedings of XML Europe 2003, London 2003.

[ACL]   FIPA Communicative Act Library Specification / Foundation for intelligent

physical agents. 2002 – FIPA standard

[Sc08]   Schwotzer, T.: *Ein P2P system basierend auf Topic Maps zur Unterstützung von Wissensflüssen*; Vdm Verlag Dr. Müller, April 2008, ISBN 978-3639008371

[SharkFW]   *Shark Framework – Shared Knowledge Framework*; Sourceforge: http://sourceforge.net/projects/sharkfw/

[SG02]   Schwotzer, T., Geihs, K. 2002. Shark - a System for Management, Synchronization and Exchange of Knowledge in Mobile User Groups. In Proceedings of the 2nd International Conference on Knowledge Management (I-KNOW '02), 149-156. Graz, Austria

[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

[TM]   ISO/IEC 13250: *Topic Maps*; December 1999

[WfMC]   The Workflow Management Coalition: http://www.wfmc.org/

[XTM]   Pepper, S., Moore G.: *XML Topic Maps (XTM) 1.0*; March 2001