A Domain Independent System Architecture for Sharing Experience

Kerstin Bach **Meike Reichle** **Klaus-Dieter Althoff**

University of Hildesheim
Institute of Computer Science
Intelligent Information Systems Lab
D-31141, Hildesheim, Germany
{bach|reichle|althoff}@iis.uni-hildesheim.de

Abstract

We propose SEASALT, an architecture based on the CoMES approach on developing collaborative multi-expert-systems using case-based reasoning and software agents technology. SEASALT is built on a modular structure and will allow implementing intelligent information systems for different kinds of application scenarios based on the architecture we are presenting. Introducing SEASALT furthers knowledge intensive services based on distributed knowledge sources. In our approach we integrate "knowledge work" in a community eliciting new information. Hence, in SEASALT agents act alongside with human beings on a community platform proving and receiving information.

1 Introduction

The SEASALT (Sharing Experience using an Agent-based System Architecture LayouT) project is intended as a long term project that aims to develop a domain-independent and versatile architecture for intelligent information systems and subsequently specify and develop its individual components and finally implement it for different application scenarios. Our work concerns the topics of knowledge engineering and knowledge management by describing an application independent architecture for intelligent information systems aimed at distributed multi-expert environments. Its main contribution is in the field of collaborative knowledge maintenance, knowledge acquisition from natural language communication, and agent technology in knowledge acquisition and maintenance. Our focus lies on domains that come with a surrounding community or can easily initiate an according community. This means that the overall domain should have a certain social relevance as well as it should be useful for the individual members of the community. Examples for such domains will be presented in this paper in section 5.

Since the knowledge managed by our systems is held by a community rather than by one or a few single experts we will aim at designing the systems' underlying architecture to involve collecting the knowledge from the community members and motivating them to provide their knowledge. The system is assembling and transforming the communities' knowledge into a well defined structure so it can easily be maintained and queried.

As a model for our architecture we use the Collaborative Multi-Expert-System (CoMES) approach as described in [Althoff *et al.*, 2007]. We have chosen the CoMES approach since we consider it to be both, easily applicable to different domains and also fitting in its focus on collaborative multi-expert environments. (In our opinion a community can be regarded as collaborating experts.) Our architecture has been developed in a bottom-up procedure by applying the CoMES approach to a first application scenario to receive an architecture draft. The draft was then subsequently generalized by mapping the draft to other scenarios and extending the architecture so it can also be applied to the new scenario. The scenarios used to develop the architecture were taken from already existing research projects, the criteria on which the scenarios are chosen match are described in section 4.1, two of them are presented in section 5.

By modeling the according domain in a modularized way we can easily split it in individual topics that can then be maintained by individual case factories as described in [Althoff *et al.*, 2006]. This distributed approach allows us to more easily incorporate information collected within our community, since this information mostly does not provide complete cases of the desired domain but rather partial information that has to be assembled and combined before it can be used in a conventional information system. Modeling the system's individual components as agents additionally contributes to its flexibility and helps us to accommodate the complexities of community provided knowledge.

In this paper we will firstly present prior work that has already been done in connection to advancing the CoMES approach. Next we will further specify the CoMES approach by concretizing it and transforming it into a first architecture layout and giving a detailed description of its individual components. Then we will demonstrate the architecture's adequacy by applying it to different application scenarios that have been developed within our research group. Finally we will give an outlook on the further proceedings of our work.

2 Collaborative Multi-Expert-Systems

Collaborative Multi-Expert-Systems (CoMES, see [Althoff *et al.*, 2007]) denote a new research approach that is both, a continuation of the well-known expert system approach and a research direction based on the ideas of case factory and knowledge-line [Althoff *et al.*, 2006; 2005].

In the *Knowledge-line* concept we systematically apply the software product-line approach [van der Linden *et al.*, 2007] from software engineering to the knowledge of knowledge-based systems. This enables the necessary "knowledge level modularization" for building potential variants in the sense of software product-lines. The modularization can be achieved by making use of multi-agent systems [Burkhard, 2003; Weiß, 1999] as a basic approach for knowledge-based systems. An intelligent agent – as a first approximation – is implemented as a case-based reasoning (CBR) system [Althoff, 2001], which, besides case-specific knowledge, can also include other kinds of knowledge. Each CBR agent is embedded in a case factory that is responsible for all necessary knowledge processes like knowledge inflow, knowledge outflow as well as knowledge analysis.

A *Case Factory* (CF) is an organizational unit that emulates the well-known experience factory approach [Basili *et al.*, 1994] from software engineering. Each role within an experience factory motivates the introduction of one or more software agents for carrying out automatable (sub-)tasks more and more independently. Like the CBR agents, the associated respective CF agents are intended to learn from experience. For example, they could implement machine learning techniques for analyzing, evaluating, and maintaining the case base of the CBR system agent. Usually each software agent has a human coach, namely the one being responsible for the role (in the sense of the experience factory approach), jointly taking over the respective assigned tasks. The human coach provides case-specific knowledge for the case base of the assigned CF agent(s) as well as feedback to suggested decisions. The coach's motivation for providing knowledge is to make the CF agents as competent as possible to transfer more and more routine tasks to these CF agents. Of course, overall responsibility and control remain with the human decision maker.

While many early (and also some current) expert systems had the problem of acquiring and maintaining their knowledge, the underlying idea in CoMES is to "develop CoMES where knowledge is produced": for example in knowledge communities already available in the World Wide Web. CoMESs do not necessarily try to integrate knowledge from different sources/experts but based on many experts. As a consequence, they do not have the immediate goal to formally represent the whole knowledge necessary to solve problems associated with a given task. Instead the idea is to learn step by step based on prior experiences or case-specific knowledge provided by cooperative authors. Another idea is to keep the resulting learning scenarios/tasks as simple as possible, thus having more agents and having each one learning in a rather simple way. In this context we are thinking of computer science techniques focusing on experience like case-based reasoning, experience management, case factory, machine and human learning, cognitive architectures, etc.

3 Prior Work

In this section we present prior work which supports realizing SEASALT. We will introduce techniques and explain how they can be applied to further the implementation of SEASALT. The approaches described in this section will be specific and concentrate on a single facet.

3.1 Domain Modeling for Unstructured Texts

In SEASALT we aim to process contributions given in web communities to model the provided knowledge. Since we do not know what kind of data can be expected, we have to deal with unstructured texts of different domains. We decided to use Textual Case-Based Reasoning (TCBR) to capture the given information and formalize it [Lenz *et al.*, 1998]. Before a TCBR-System can be set up, a domain model has to be defined to ensure that the data on which the system is based can be accessed. Hence, analyzing unstructured texts requires a domain model containing terms to represent texts. The terms contained in a vocabulary have to be both, domain specific and domain independent [Roth-Berghofer, 2003]. Since we do not know what kind of text (topic, used terms, etc.) is used in the contributions we have to create a vocabulary repository containing both types of terms. When dealing with large databases containing unstructured texts, building a domain model is a very time-consuming task. [Bach, 2007b] describes how domain modeling for TCBR can be realized for a case base containing more than 9.500 cases with 2.2 million words. The paper describes how heterogeneous repositories are merged to build a vocabulary repository of general and domain specific terms to facilitate the detection of unknown words. Therefore vocabulary repositories based on GermaNet[1] and Web Services provided by *Projekt Deutscher Wortschatz*[2] were used.

Since we expect SEASALT dealing with a large amount of unstructured texts we will need a semi-automatic support for searching and modeling new terms as well. In [Bach and Hanft, 2007] the Textual Coverage Rate (TCR) is introduced, which is a method to determine the IE coverage of unstructured texts using a given vocabulary. This empowers a knowledge engineer to decide which parts of the corpus should be modeled first and how much should be done to achieve a certain quality of modeling which means coverage of unstructured text through IEs. Another feature of a vocabulary repository is the provision of synonyms which allow connections to find similar words. In SEASALT we receive information of a World Wide Web community and we expect dealing with misspelling/mistyping of terms. Resolving misspelled words we will have to create a misspelling repository in order to recognize and suggest corrections. Misspelled words and their corrections will be stored during the modeling process and during semi-automated processes analyzing unknown texts, former corrections can be used to suggest the correct word to the knowledge engineer. In automated processing the vocabulary is used to find correctly spelled words and continue using them.

Although the domain modeling for unstructured text is only one part to formalize unknown unstructured texts (semi-)automatically, it will help us to gather information of the community. We still miss the definition of connections between terms beyond synonyms to find similar documents.

3.2 Combination of Distributed Information

Prior work on the combination of distributed information on a complex domain, such as Free/Libre Open Source Software (FLOSS) has been done in [Reichle, 2007]. Here information about FLOSS applications is gathered from different sources such as public FLOSS directories, GNU/Linux distributions, collaboratively maintained software tags and bug tracking systems. [Reichle, 2007] describes how to combine and unify these different information sources and transform them into a general model for FLOSS, that contains not only the most relevant information on a FLOSS application but also weights the different attributes and provides individually adapted similarity measures, so the model can be used to set up a CBR system's

[1] http://www.sfs.uni-tuebingen.de/lsd/

[2] http://wortschatz.uni-leipzig.de/Webservices/

case base.

While this work already offers useful insights on how to combine different information sources in this rather complex domain, it lacks the flexibility that the SEASALT architecture provides. In the approach presented there, all information is merged by different scripts and parsers beforehand and then inserted into a database and subsequently imported into the case base of an empolis e:IAS system. This way the combination of the different information sources is already predefined and can not be adapted to for example different kinds of questions or information needs. Also such a huge, monolithic case base is harder to maintain, especially if automatic case base maintenance is to be used [Roth-Berghofer, 2003]. An architecture like this makes it harder to automatically insert new information. This is especially true if this information does not come in complete case descriptions but is collected from a real live community and thus maybe fragmented or incomplete. The SEASALT approach of storing information in a modularized way can make use of this work's findings but also extend them to suit a more flexible system architecture.

4 The Architecture of SEASALT

4.1 Individual Architecture Units

This section describes each unit of the SEASALT architecture. The descriptions follow the logical order starting with a given task which is processed using the architecture.

The architecture can be vertically split in two parts as can be seen in figure 4.1. On the left hand side the knowledge provision and on the right hand side the knowledge acquisition. First we will focus on the knowledge provision and explain how a question to the system will be processed. A user enters a question using the *Interface* which passes the question on to the *Coordination Agent*. The *Coordination Agent* analyzes the question, looks up the matching *Topic Agent(s)* and sends its requests to them. A response based on the existing case base is created by each *Topic Agent* and passed back to the *Coordination Agent*. Finally, the response of the *Topic Agents* is used by the *Coordination Agent* to compile an answer.

To exemplify this, imagine the *Knowledge Line* as a cabinet with the *Coordination Agent* as its chancellor or president. Each member of the cabinet is concerned with a different department and has a large staff, that provides them with information, updates it and make simple decisions in the name of their respective cabinet member. In our architecture the members of the cabinet would be the *Topic Agents*, their staff would be each *Topic Agent's* individual *Case Factory*.

The knowledge acquisition process is illustrated on the right hand side of our architecture diagram. It consists of two parts: a platform that supports the community and offers communication services where community members can discuss and exchange experiences and a knowledge engineering part in which contributions are analyzed and processed. Based on the *Topic Agents* we will place *Collectors* belonging to a *Topic Agent* in the community to collect information. Since the information given in the community is not structured, a *Knowledge Engineer* has to support the *Collector's* work. The *Knowledge Engineer* will receive and formalize contributions that the *Collectors* classify to be useful for their individual topic. In the beginning this has to be a human being, but the *Knowledge Engineer* trains an *Apprentice* agent by providing it with a document set, consisting of the source contribution and the structured document. Thus the *Apprentice* will soon be able to do at least basic pre-processing for the *Knowledge Engineer*. The formalized contribution is passed on to the *Case Factory* to include the new cases and make them available to the system. The original contributions that are provided with the new cases can be used by the system to support its answers. Additionally the community platform which is used by the *Collectors* to collect information also offers agents for intelligent services that make the platform more usable, ease the communication and accelerate conclusions on topics.

It is necessary that the information provision and the community platform are presented and perceived as a single entity. Thereby the platform's intelligent services can be an additional motivation for the community members to not only passively let the system collect information, but to directly provide it with complete cases or feedback on the knowledge it already has collected. This is especially important, since it may be assumed that the system's knowledge provision component itself will mainly be used by those community members that have less knowledge on the domain, while those who are experts on the domain and have much knowledge about it (the community's *regulars* or *core group*) will use it less often. Those regulars however will make more use of the community platform and thus they will also more appreciate its intelligent services and be more motivated by them to actively provide the system with their knowledge.

Application Scenario

The motivation of users joining and working in the community is crucial for a successful application of SEASALT. Therefore, application scenarios in which an intelligent information system using our architecture is implemented should fulfill the following two conditions:

- *Socially relevant*, the topic should be relevant enough to ensure a community which addresses enough people who can join and share their knowledge.
- *Individually useful*, so each member of the community can ask their own questions and receives satisfying answers and is thus motivated to further contribute to the system's knowledge base.

Defining the scenario's tasks and domains should focus on both conditions to create a good running community.

The application scenario's knowledge domain should fit the following characteristics:

- Suitable for community maintenance: all information contained is accessible to everybody,
- Modular structure of knowledge to facilitate assignment to *Topic Agents*,
- Well-defined range of topics with typical questions.

Interface

The communication between the user and the knowledge providing components is enabled using a Human Computer Interface (HCI), for example, containing forms or text boxes. The *Interface* can be either a website or a client application which passes the question on to the *Coordination Agent*. The *Interface* depends on the structure of the domain data and supports the structuring of questions ensuring the *Coordination Agent* can handle them. After processing the question the *Interface* will be used to display the answer.

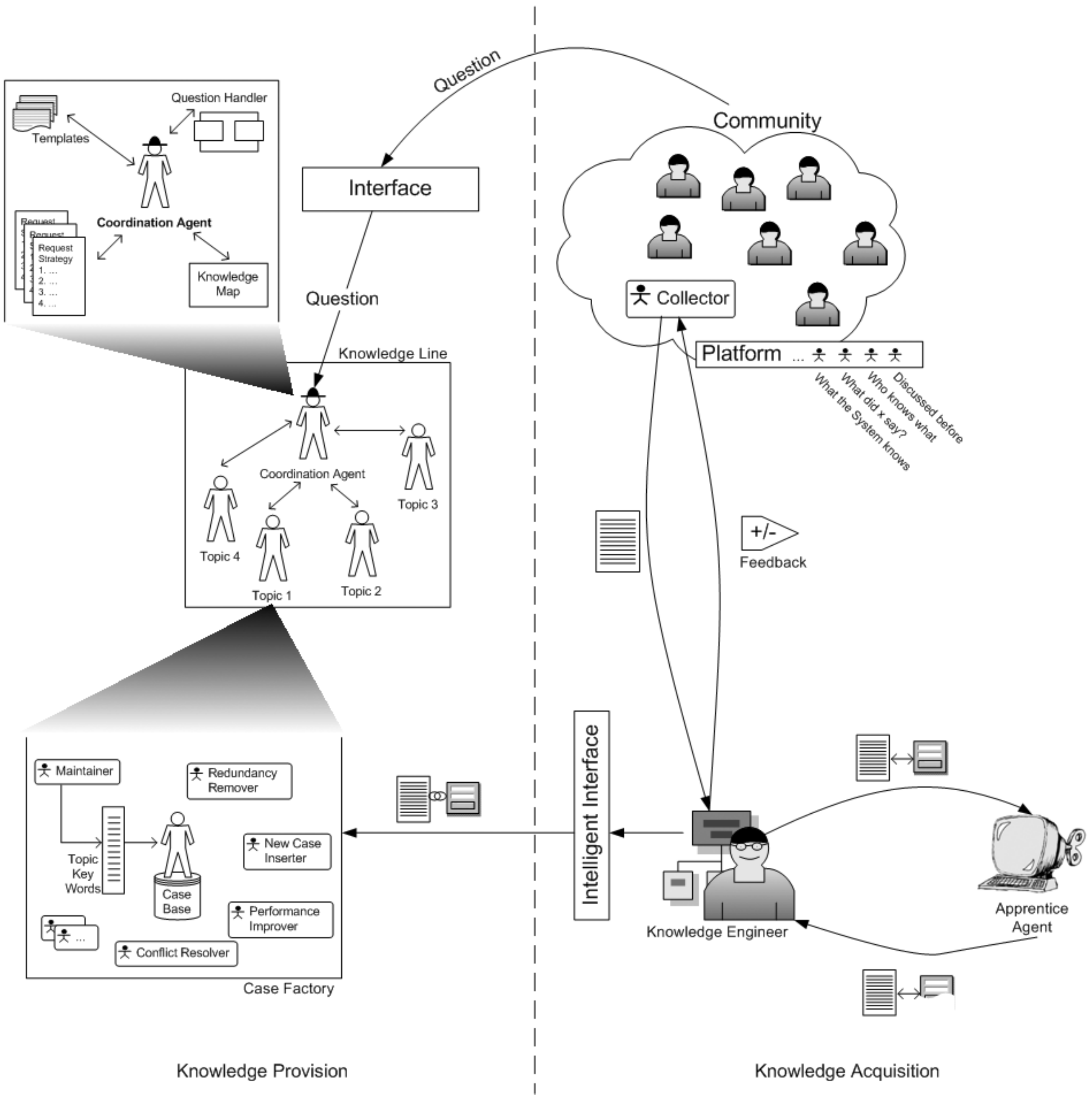

Figure 1: The SEASALT architecture

Knowledge Line
A knowledge domain that has the above mentioned characteristics allows building numerous *Topic Agents* that can be coordinated by one *Coordination Agent.* A clearly defined area of expertise for each *Topic Agent* is needed. Following a modular structure enables realizing a distributed multi-expert-system.

The Knowledge line consists of several *Topic Agents* whereas each agent is responsible for one individual topic. Each *Topic Agent* is implemented as a CBR system, has an own *Case Factory* to manage its knowledge, and is equipped with a list of topic key words to communicate its area of expertise to the *Collector* agent.

Coordination Agent
The *Coordination Agent* receives the user's questions from the *Interface* and communicates to individual *Topic Agents.* It possesses a Question Handler to distinguish the type of question and select a Request Strategy according to the question. Request Strategies indicate in what order to query individual *Topic Agents* and how to combine their results for creating a human readable answer. Furthermore the *Coordination Agent* has a Knowledge Map indicating which *Topic Agent* is competent on which topic and how to contact it. To generate the human readable answer, templates are provided which will be filled with the combined responses of the contacted *Topic Agents.*

Case Factory
The individual *Topic Agents'* case bases are maintained by several *Case Factory Agents* which serve different tasks such as inserting new cases, removing redundancies, resolving conflicts, improving the case base's overall performance, maintaining the topic keywords list, etc. Additionally each *Topic Agent* has a collector agent that is equipped with the keyword list and tries to identify relevant contri-

butions or contributions on the community platform which are delivered to the *Knowledge Engineer*. Each contribution passed on to the *Knowledge Engineer* is analyzed and the *Knowledge Engineer* will give a feedback about its actual relevance to the *Collector* so the agent can improve its collection strategy.

Furthermore, each *Topic Agent* has a case base log where every *Case Factory Agent* notes all actions performed on the *Topic Agent's* case base and why they were carried out, ensuring all decisions and actions can be retraced.

Collector

One *Collector* agent per *Topic Agent* is placed in the community. The *Collectors'* task is to collect community contributions that are relevant to their respective *Topic Agent's* area of expertise. Their choices are based on the respective *Topic Agent's* key word list. The contributions identified as relevant are passed on to the *Knowledge Engineer*. The *Collectors* then receive feedback on their choices from the *Knowledge Engineer*, thus improving their performance over time.

Knowledge Engineer

The role of the *Knowledge Engineer* connects the *Case Factories* and the *Community* enabling the exchange of knowledge between both of them. The *Knowledge Engineer* is an expert on the system's tasks and also takes part in the Community. He receives contributions collected on the community platform by the different *Collector* agents and gives each *Collector* agent a feedback on its decisions. Furthermore the *Knowledge Engineer* formalizes relevant contributions, so the according *Case Factory Agent* can insert the new cases into its *Topic Agent's* case base. The formalization is supported by an intelligent interface that offers features like controlled vocabulary, spell checks, synonyms, etc. as it is presented in [Bach, 2007b; Bach and Hanft, 2007].

The *Knowledge Engineer* is supported by an *Apprentice* agent. To train the *Apprentice* the *Knowledge Engineer* uses relevant contributions and their formalizations (document sets). In return the *Apprentice* is able to do at least basic pre-processing and can take more and more work off the *Knowledge Engineer* as its training continues.

Since these are only roles, the tasks of the *Knowledge Engineer* can of course also be executed by more than one person as well as the *Apprentice* can be more than one agent.

Apprentice

The *Apprentice* is an agent or numerous agents, that is trained with document sets by the *Knowledge Engineer*. We aim to automate as many tasks as possible. The *Apprentice* is at least able to do preparatory pre-processing and is meant to take more and more work off the *Knowledge Engineer*.

Community

The Community consists of a group of people who discuss several topics of a common domain. A communication *Platform* is used to discuss and exchange opinions. It is extended with several *Collector* agents, that collect relevant contributions on their topics. The people participating in the community will be able to use the given system.

Platform

The *Platform* is used by the *Community* to discuss and exchange opinions and by the *Collector* agents to collect relevant contributions. It offers agents of its own that perform intelligent services for the platform's users such as:

- „This has been discussed before" – Pointing out to users if similar discussions have been had before
- „X might be able to help you on this" – Learning which user is interested in or an expert on a certain topic. (Interested users start threads on topic, experts answer in threads on topic.)
- „What does X think about that?" – Selecting all contributions by a specific person according to a specified topic
- „What the system knows" – Can tell what data the knowledge-based system has on a specific topic

An example for such intelligent services is presented in [Feng *et al.*, 2006].

5 Practical Applications of the SEASALT Architecture

After introducing the architecture we will now describe two application scenarios which will be implemented using SEASALT. Both application scenarios should substantiate the architecture and clarify the usage of the CoMES approach. Further their implementation and testing will serve as a first evaluation and help to improve the architecture and detect flaws in the concept.

5.1 docQuery - An Intelligent Information System on Travel Medicine[3]

Nowadays it has never been easier to travel to different places, experience new cultures and get to know new people. In preparation for a healthy journey it is important to acquire high quality of reliable information about travel medicine prevention. Travel medicine is specialized on medical issues like diseases, vaccinations, etc. which might occur before, during and after a journey.

Currently the World Wide Web offers many websites, discussion forums and services where a traveler can gather information. Usually those websites do not contain all medical information a traveler might need and the editors are mostly unknown. Furthermore the information is spread over hundreds of websites and it is challenging and time-consuming to find appropriate information [Bach, 2007a].

Together with a team of certified doctors of medicine with a strong background of travel related medicine, we will use SEASALT to implement *docQuery*, an application to provide travel information based on travelers' key data such as travel period, destination, age(s) of traveler(s), activities, etc. [Bach, 2007a]. We aim at establishing a community of experts exchanging knowledge on their expertise and getting new information from their colleagues. *docQuery* will be supervised by experts and integrate the experiences of travelers using the given advices to improve the quality and early identify new issues.

Implementing *docQuery* using SEASALT will help both, creating a community to exchange knowledge and offering an multi-expert-system on travel medicine tasks. In

[3] This is a project in cooperation with Thomas Schmidt (mediScon worldwide).

the travel medicine domain various topics like medications, countries, diseases, etc. can be distinguished and will each be found in an individual *Topic Agent.*

5.2 FLOSSWALD – Information System on Free and Open Source Software

Free/Libre and open source software (FLOSS) has produced a large and diverse range of software which offers numerous and high quality alternatives to almost all commercial software applications. Popular FLOSS like Firefox[4], Thunderbird[5], OpenOffice.org[6], vlc[7] or the GNU/Linux operating system are steadily gaining users both in the private and commercial sector.

However the FLOSS community is a complex social and technical network that consists of tens of thousands of individual groups and projects that produce software in all degrees of quality. Research [Reichle, 2007] shows, existing FLOSS directories are mostly used by expert users and FLOSS insiders, while less experienced users prefer general search engine and the advice of friends when choosing software. The success of choosing software in this way is limited though. A general web search for software for a specific purpose will most likely yield the most popular software (or rather that with the most popular website) but not necessarily the one that is best suited for the given purpose and user. Asking friends or colleagues for software advice is also of only limited use, since those usually have the same level of knowledge as the seeking person and can thus not offer qualified advice.

The FLOSSWALD System [Reichle and Hanft, 2006; Hanft and Reichle, 2007] aims to be an intelligent information system on free/libre open source software (FLOSS), that offers the community's knowledge and experiences with different software to unexperienced users and can be queried using natural language and simple menus. To use the system, the user does not have to be an expert on software or computers in general. In order to achieve this the system combines information collected from different FLOSS directories by the FLOSSmole Project [Howison *et al.*, 2006] with data provided by GNU/Linux distributions (in this case the Debian Project's Package data). This knowledge can be enriched using the DebTags, a collection of collaboratively maintained tags covering different aspects of software [Zini, 2005]. Additionally more user-friendly attributes such as tasks that a software can be used for "vague attributes" like user friendliness, flexibility or stability that can be learned from the community. The combination of these different knowledge sources and the problem of keeping them up-to-date provides an adequate application scenario for the SEASALT architecture.

5.3 Implementation Details

Both application scenarios, along with others, will base on SEASALT and use basically the same technologies. Raw data will be contained in a DBMS like PostgreSQL[8] and the *Case Factory* will be realized using e:IAS.

e:IAS is an information and knowledge management suite developed by empolis[9], a subsidiary of Bertelsmann Arvato [empolis GmbH, 2005]. e:IAS consists of several different components for information and knowledge processing and management. Among these there is a rule engine, which can be used to model business processes and classification tasks (via rules). Hence, it contains a text miner that can be used for analyzing documents as well as it offers free text user queries, which can further be combined with a downstream pattern matching component.

e:IAS also includes a powerful CBR engine and the so called creator module, which is based on the free IDE eclipse[10]. The creator is used to model the cases for the CBR engine. A case is modeled as an aggregate of attributes. The creator is used to model the required attributes, their domains and also underlying concepts and taxonomies and the respective attributes' similarity functions. The model is stored using RDF respectively OWL, all further information is stored in an XML format. The case models in e:IAS can be filled with imported data from a multitude of sources. The data import and their further processing is done using a modular pipeline system in which the different functionalities can be freely combined using individual pipelets. Pipelets offer for example the import from simple text files, documents, databases or websites using an integrated crawler, and their subsequent processing such as breaking the input data into single values and assigning them to their respective attributes, analyzing texts with a text miner or spell checker or stripping input of html or xml elements. Once the data are imported and processed the system's knowledge base is ready and can be used by the e:IAS knowledge server. Pipelines are however not only used for importing data, but also for integrating the aforementioned functionalities and making them available to the Knowledge Server. e:IAS offers pipelets for text mining, the creation and application of rule sets, searching, automated classification and the generation of dialogs. Additionally the Knowledge Server is able to use external knowledge sources such as already existing dictionaries. Different types of clients including simple web clients but also rich clients or JavaBean applications can be used to access the Knowledge Server. Communication between the server and its clients can be implemented using various languages such as XML, COM or RMI.

6 Conclusion and Future Work

6.1 Conclusion

In this paper we have presented an architecture that follows the CoMES approach and connects case-based reasoning, software agent technologies and the acquisition of knowledge distributed in a community.

We also described the collaboration of different experts as well as the integration of software agents in a (given) community. To further knowledge exchange we extend common community platform features with several intelligent services executed by software agents. Alongside integrating knowledge work in a community, we have described how to use the elicited knowledge in an intelligent information system. Developing our architecture we follow a modular approach by creating *Topic Agents* for each specific topic and combining them to create more complex answers. Hence we specified each unit of SEASALT and assigned tasks they have to perform. To evaluate our architecture we presented two application scenarios which can be implemented using SEASALT.

[4] http://www.mozilla-europe.org/en/products/firefox/

[5] http://www.mozilla-europe.org/products/thunderbird/

[6] http://www.openoffice.org/

[7] http://www.videolan.org/vlc/

[8] http://www.postgresql.org/

[9] http://www.empolis.com

[10] http://www.eclipse.org/

6.2 Scientific Contribution

In this work we focused on knowledge engineering and knowledge management by describing an application independent architecture for intelligent information systems aimed at distributed multi-expert environments. Agent technology is applied in the case factory agents, the collector agent, and the apprentice agent. The work of the knowledge engineer and its apprentice agent concerns the field of knowledge acquisition from natural language communication. Collaborative knowledge maintenance is realized in the implementation of the topic agents, case factory, and collector agent.

Our approach is characterized by our focus on communities and the modular structure of knowledge as represented in our topic agents which also distinguishes the SEALSALT architecture from general search engines and more monolithic approaches. Furthermore the architecture is more topic and community oriented. The use of case-based reasoning allows for easy maintenance of the knowledge base and powerful retrieval.

6.3 Future Work

Future work will go into the detailed specification and implementation of the described units of SEASALT to realize CoMES. We will first have to define general functions and units which have to be provided for any application, then implement those units and in a final step adapt them to the different application scenarios like FLOSSWALD and *docQuery*. As mentioned before, the *Case Factory* will be implemented using e:IAS representing one *Topic Agent*. The communication between agents will be realized using RMI which is already provided for e:IAS. Also the communication and tasks of the *Coordination Agent* have to be defined and implemented. We have also created the role of a *Knowledge Engineer* which trains the *Apprentice Agent*. To support the *Knowledge Engineer* by pre-processing the communities' documents we will use TCBR, but in advance we have to clarify which tasks a *Knowledge Engineer* has to fulfill and how agents can support it and adapt to those tasks. We will also have to undertake further research into issues arising from collaborative knowledge maintenance such as contradicting experiences, alternate solutions or incomplete data.

To ensure successful community work, community members have to be motivated to share their experiences and use the platform. In exchange platform services such as specific platform agents should ease communication. Hence, questions should be answered correctly so users can trust in it. Existing communities have to be introduced to our approach and the platform features need to be discussed.

References

[Althoff *et al.*, 2005] Klaus-Dieter Althoff, Jens Mänz, and Markus Nick. Integrating Case-Based Reasoning and Experience Factory: Case Studies and Implications. In Uli Furbach, editor, *Proceedings of the 28th German Conference on Artificial Intelligence Workshop on Knowledge Engineering and Software Engineering*, pages 1–12, Koblenz, Germany, 11. - 14. September 2005. Springer, LNAI 3698.

[Althoff *et al.*, 2006] Klaus-Dieter Althoff, Alexandre Hanft, and Martin Schaaf. Case Factory – Maintaining Experience to Learn. In Mehmet H. Göker, Thomas Roth-Berghofer, and H. Altay Güvenir, editors, *Proc. 8th European Conference on Case-Based Reasoning (ECCBR'06), Ölüdeniz/Fethiye, Turkey*, volume 4106 of *Lecture Notes in Computer Science*, pages 429–442, Berlin, Heidelberg, Paris, 2006. Springer Verlag.

[Althoff *et al.*, 2007] Klaus-Dieter Althoff, Kerstin Bach, Jan-Oliver Deutsch, Alexandre Hanft, Jens Mänz, Thomas Müller, Regis Newo, Meike Reichle, Martin Schaaf, and Karl-Heinz Weis. Collaborative Multi-Expert-Systems – Realizing Knowlegde-Product-Lines with Case Factories and Distributed Learning Systems. Technical report, University of Osnabrück, Osnabrück, September 2007.

[Althoff, 2001] Klaus-Dieter Althoff. Case-Based Reasoning. In S.K. Chang, editor, *Handbook on Software Engineering and Knowledge Engineering. Vol.1, World Scientific*, pages 549–587. 2001.

[Bach and Hanft, 2007] Kerstin Bach and Alexandre Hanft. Domain Modeling in TCBR Systems: How to Understand a New Application Domain. In David C. Wilson and Deepak Khemani, editors, *Proceedings of the 7th International Conference on Case-Based Reasoning (ICCBR) 2007, Workshop on Knowledge Discovery and Similarity*, pages 95–103, Belfast, Northern Ireland, 2007.

[Bach, 2007a] Kerstin Bach. docQuery – Reisemedizinisches Informationssystem. Internal project report, 2007.

[Bach, 2007b] Kerstin Bach. Domänenmodellierung im Textuellen Fallbasierten Schließen. Master's thesis, Institute of Computer Science, University of Hildesheim, 2007.

[Basili *et al.*, 1994] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Experience Factory. In John J. Marciniak, editor, *Encyclopedia of Software Engineering, vol 1*, pages 469–476. John Wiley & Sons, 1994.

[Burkhard, 2003] Hans-Dieter Burkhard. Software-Agenten. In Günther Görz, Claus-Rainer Rollinger, and Josef Schneeberger, editors, *Handbuch der Künstlichen Intelligenz, 4. Auflage*, pages 943–1020. Oldenbourg, 2003.

[empolis GmbH, 2005] empolis GmbH. Technisches White Paper e:Information Access Suite. Technical report, empolis GmbH, September 2005.

[Feng *et al.*, 2006] Donghui Feng, Erin Shaw, Jihie Kim, and Eduard Hovy. An intelligent discussion-bot for answering student queries in threaded discussions. In *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, pages 171–177, New York, NY, USA, 2006. ACM Press.

[Hanft and Reichle, 2007] Alexandre Hanft and Meike Reichle. The FLOSSWALD Information System on Free and Open Source Software. In Norbert Gronau, editor, *Proceedings of the 4th Conference on Professional Knowledge Management - Experiences and Visions*, pages 135–142, Berlin, March 2007. Gito Verlag.

[Howison *et al.*, 2006] James Howison, Megan S. Conklin, and Kevin Crowston. FLOSSmole: A collaborative repository for FLOSS research data and analyses. *International Journal of Information Technology and Web Engineering*, 1(3):pages 17–26, Juli - September 2006.

[Lenz *et al.*, 1998] Mario Lenz, André Hübner, and Mirjam Kunze. Textual CBR. In Mario Lenz, Brigitte Bartsch-Spörl, Hans-Dieter Burkhard, and Stefan Wess, editors, *Case-Based Reasoning Technology – From Foundations to Applications*, Lecture Notes in Artificial Intelligence, LNAI 1400, pages 115–137. Springer-Verlag, Berlin, 1998.

[Reichle and Hanft, 2006] Meike Reichle and Alexandre Hanft. The FLOSSWALD Information System on Free and Open Source Software. In Martin Schaaf and Klaus-Dieter Althoff, editors, *Lernen - Wissensentdeckung - Adaptivität Proceedings of the LWA 2006, FGWM 2006 Workshop on Knowledge and Experience Management*, volume 1/2006 of *Hildesheimer Informatik-Berichte*, pages 229–233. University of Hildesheim, Oktober 2006.

[Reichle, 2007] Meike Reichle. Entwicklung und Implementierung eines Modells zum Retrieval von Free/Libre Open Source Software unter Verwendung eines Case-Based Reasoning Systems. Master Thesis (Magisterarbeit), 2007.

[Roth-Berghofer, 2003] Thomas Roth-Berghofer. *Knowledge Maintenance of Case-Based Reasoning Systems – The SIAM Methodology, Dissertation*. PhD thesis, Universität Kaiserslautern, 2003.

[van der Linden *et al.*, 2007] Frank van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering*. Springer, Berlin, Heidelberg, Paris, 2007.

[Weiß, 1999] Gerhard Weiß, editor. *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1999.

[Zini, 2005] Enrico Zini. A cute introduction to Debtags. In Alexander Schmehl, editor, *Proceedings of the 5th annual Debian Conference*, pages 59–74, Helsinki, Finland, 10. - 17. Juli 2005. The Debian Project.