Alexander Adelmann BSc.

Context Search Helper

A context based resource recommender system.

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

eingereicht an der

Technischen Universität Graz

Betreuer

Kappe, Frank, Univ.-Prof. Dipl.-Ing. Dr.techn.

IICM

Institut für Informationssysteme und Computer Medien

Graz, September 2016

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Datum	Unterschrift

1. Table of Contents

Abstract

The Context Search Helper System is a context aware software to help the user get his work done. The system analyses the used device, which could be a notebook or an office PC running a Microsoft Windows [a] operating system and also takes into account what the user is currently working on, to provide useful documents, emails or links from the user's device, his mail account and the web. To accomplish this task, the system crawls the user's documents, emails and browser history and queries these data sources according to the currently detected context.

The frontend of the system is developed as a Microsoft Word [b] 2013 Add In. The modular architecture of the system makes it very easy to implement other clients for the system as the important calculation is done in the background.

Das Context Search Helper System ist eine Kontext basierte Software, deren Hauptaufgabe die Arbeitserleichterung für den Benutzer ist. Das System analysiert das Gerät auf dem es läuft, was normalerweise ein Arbeitscomputer oder Laptop ist, der das Microsoft Windows Betriebssystem verwendet. Auch die aktuellen Aktivitäten des Benutzers auf dem System werden in die Analyse mit einbezogen. Auf diese Weise versucht das Context Search Helper System herauszufinden woran der Benutzer gerade arbeitet. Dann wird eine Reihe von Hintergrundberechnungen vorgenommen und anschließend mehrere Datenquellen durchsucht. Zu diesen Datenquellen zählen unter anderen das lokale Microsoft Outlook [b] Postfach des Benutzers oder die lokal gespeicherten Dateien. Auch die Webbrowser Historie wird durchsucht und eine Websuche über die Suchmaschine des Know Centers [h] durchgeführt.

Die Ergebnisse der Suche werden dem Benutzer dann in einer Sidebar in Microsoft Word angezeigt. Auf diese Weise wird der Benutzer bestmöglich bei seiner Arbeit unterstützt.

Die modulare Architektur des Systems bietet eine einfache Möglichkeit der Erweiterung von Client Applikationen, welche die Suchergebnisse darstellen.

1. Theoretical Background

In this section the current situation and its problems will be discussed as well as solutions for it. The connection of the theory field and the Context Search Helper application will be shown and why these theoretical ideas and concepts were used during implementation. Also a related system will be discussed and compared to our approach.

1.1 Current Situation and Problem

This section will focus on how users do their work and what can be improved to help them to be more efficient and find information easier and faster.

Working in a company as a white collar worker or researching for educational purposes like writing on a highly specific scientific thesis or paper requires a user to work with multiple programs at once. The user needs to switch between these programs very often and the information about the current work is not stored in one single window, but is very fragmented and located in different applications and locations.

To explain the current situation of working on a computer and the problems that are present nowadays, the following example will help to make the situation easier understandable.

A user is working in a large software company in the research department. This means that he is doing a lot of research on the web and has lots of meetings with colleagues about current projects and details of these projects. The company has different information and collaboration channels which are used by all employees. The employees communicate using applications like Skype4Businness [b] and Yammer [c]. But also older technologies like emails are used to exchange information between users. A lot of important information about running projects and the discussions about these projects is stored in Microsoft OneNote [b].

The user is currently working on a research project on how to improve and optimize the data flow and amount in large distributed systems. The communication between parts of the system is done with WCF Services [hh].

A normal work day for the user is reading lots of scientific papers about the topic and doing research on the web and browsing multiple blogs and other sites which information about the topic. The user also chats with other employees over S4B (Skype4Business) and Yammer because some of them also worked on similar projects and may help him with a few details. Not all colleagues are available over Yammer and S4B, so the user sends also a few emails and discusses project relevant topics also in emails.

The user needs to write a paper to finish the project and while he is working on the document, which he is writing using Microsoft Word 2013, he does not know what to write next. He forgot details about a subtopic which he remembers having discussed with a colleague. The user also saved a few PDF papers about the topic on his laptop.

Because he does not know what to write as he forgot the details about the topic he starts searching for the PDF papers he read a few weeks ago. The user stores documents like this in a few folders on his system so he searches for the documents in the folders where he usually stores papers. After a few minutes of searching the user found the PDF papers.

Unfortunately, the information he was looking for is not stored in the documents. The user also remembers talking to another employee in the company about it, so he starts searching for old Yammer posts and S4B conversations well as emails he may have sent or received by his colleague. After another 15 minutes the user found an email conversation he had with the other employee. Finally, he knows how to proceed writing his paper and what do describe next.

This example shows what the problem is for white collar workers. Information is not stored at one single location but is highly distributed. Also a lot of different applications are used and each application may hold parts of the information a user may be need at the moment.

1.2 Motivation

As mentioned in the previous section the main problem with today's work style of white collar workers is that there is no single place where all the relevant and important data is stored. The information is stored in many different places and applications. All the information is fragmented and users need to search for the information they are looking for.

As users need to use more and more applications the problem grows with every application that is added to the computers. This does not only affect the efficiency the users are working with, but also has impacts on the profit of companies. Every minute a user is searching for information in the company network or on his computer is a minute he is not working for projects. Another problem is that the mentioned problem leads to frustrated employees which also leads to less productivity.

The main aim of this thesis and the Context Search Helper system that has been implemented is to address these problems by lightening the users' workloads. Detailed information about the Context Search Helper system can be found in section Context Search Helper.

This is achieved by doing all the searching for relevant information for the users so they do not need to waste their time on search for documents and so forth. The system analyses the operating system it is running on as well as the current activity of the user which is currently logged in. The application runs in the background and gathers all the data and tries to extract the user's current working topic from all the data gathered in the background. With all that information the Context Search Helper then queries various data sources for relevant results according to the analyzed context of the user, his machine and work tasks. The results are then presented to the user in a structured and user-friendly way.

This leads to a user who does not always need to switch between applications and search for information that is stored somewhere on the computer or network. The user can stay in one application and all the relevant and important data regarding his current task is presented to him. With the help of the Context Search Helper the user's productivity is improved and wasted time is minimized. At the moment the system is implemented as a single user application, but future releases are likely to support a multiuser environment.

1.3 Related Work

In the following section related projects will be shown. These projects have similar means and try to achieve the same result as the Context Search Helper or use similar functions in the background.

One project that will be shown and discussed in detail is the Aposdle project [f]. This project has been developed under the leadership of the KNOW-CENTER GmbH [h]. There are also projects like Prolix [ss] and EmployID [rr] which will not be discussed in this thesis.

All of the mentioned applications are kinds of e-learning applications that focus mainly on learning in a business and working environment. Although the Context Search Helper system is not a learning system for users or companies, projects like Aposdle use similar techniques and ideas in the background which is why these applications will be discussed in the following chapter.

<u>Aposdle – a Work-integrated learning (WIL) system:</u>

The application supports three important roles of the users, namely the worker role, the learner role and the expert role.

Users can learn during the execution of their work tasks with Aposdle and therefore become more valuable for the company. Users also have the possibility to collaborate and communicate with the assistance of the Aposdle system. Users that have difficulties with their tasks are provided a list of experts in the company that already had the same problem or can offer help. To achieve this Aposdle analyses the user's current context and skill level.

The Aposdle project is a very powerful WIL (work-integrated learning) application which targets mainly technology companies with lots of knowledge workers. The main aim of work-integrated learning applications is to seamlessly integrate the working and learning Environment as Lindstaedt et al. [16] state. Seamlessly here means that the application or the system should integrate in the working day of the users and not be a different application which would distract the users from their work. In contrast to standard e-Learning applications where the users spend the learning time in the e-Learning application, WIL systems to not need the users to stay in one application while learning. They try to not attract the user's attention and only provide information or help if the user need it. As a result of this kind of behavior Lindstaedt et. al. [16] state that testing is not an option. The users interact with lots of applications and there is no central learning system [18].

Furthermore [18] state that from a user's perspective a WIL application is spontaneous and appears randomly. Learning is only a by-product of the time the users spend at the company at their desk or workplace. This leads to the need that a WIL system support is embedded in the user's daily work tasks and routine at work. This logic and behavior of a WIL application is different to a standard e-Learning application. It is essential for a functioning WIL system to use the company's real data, like documents, projects, excel sheets, reports, presentations and so forth, and use these real data documents for learning. For a system to work like that the users need to continuously improve their skills at the workplace. Users need to have a basic knowledge of the topics that are being learned and are able to guide their own learning process and to not require the learning system to do that for them [19].

1.3.1 Apodsle Project

apodsle
learn @ work

Advanced Process-Oriented Self-Directed Learning Environment

Figure 1. The logo of the Apodsle system

One of the projects that are very similar to the Context Search Helper system, is Apodsle. Detailed information on this particular project can be found on its website. The site also provides all the papers that were written during the research and implementation of the project. The URL to the site is [f]. The Apodsle project is a system that was developed from March 2006 till February 2010, so the runtime of Apodsle was 48 months.

According to the project's website the research and the implementation of Apodsle was supported by the European Union under the Information Society Technologies (IST) priority of the 6th framework programme.

The system's aim was to provide a way for users working in large companies to learn at the workplace together with colleagues while working on projects. The name Apodsle is short for Advanced Process- Oriented Self- Directed Learning Environment.

Lindstaedt et. al. [3] found out that according to users, who were asked what the most important development and training initiatives for organizations are, the improvement of workforce productivity is very important. In fact, two third of the asked users had that opinion. Furthermore, only 7.5% of the users that participated in the survey said that the company they are working for at the moment satisfied the progress on the mentioned initiatives.

1.3.1.1 Main idea

The main aim of the Aposdle project is to help companies to continuously improve their process steps and not fall behind in the very fast developing area. The project gives companies a possibility to train their employees while working and executing their tasks for work. Therefore, the expert's knowledge is improved while working. Another problem that can be solved with Aposdle is the fact that in technical companies, employees often leave and take their knowledge with them. This can be a serious problem and Aposdle helps to reduce the impact of such a change in a team.

The users which are targeted by the system are so called knowledge workers. According to Lindstaedt et al. [3] knowledge workers are employees who have knowledge as their main resource. Knowledge workers, also called workers by Lindstaedt et.al [5], are the users that are in the main focus of the Aposdle project. These employees continuously need to improve their main value, which is knowledge. Aposdle is a perfect environment to improve one's skills and knowledge and therefore a perfect system for knowledge workers.

Aposdle consists of three main parts which interact and together built the powerful Work-Integrated-Learning (WIL) tool. These three parts are described in the following paragraph.

<u>Learn during work:</u> This is the part of the system that focuses on the user's activities. The system analyses how a user interacts with applications and documents and this user behavior is then matched with interaction patterns the system already has learned from the users. The agent, which is analyzing the user, can use the matches to figure out what task the user is currently working on. The user is then continuously presented documents from the company's repositories which match the user's task. The system not only shows files that match the users current work task, but also analyses the user's skill level and uses this Information to present the user even better matching documents from the company database. With the information provided the user is constantly learning during the execution of his work tasks.

<u>Learn collaboratively:</u> When a user tries to accomplish a work task there is always the chance that he runs into problems that stops him from proceeding with his work. In this case the user would need to do some research on the internet for example to figure out what the problem is and how to eliminate it. The Aposdle project helps the user to find the information that he needs in only a few clicks. A user is shown a list of other employees of the company that have the required skill level and who have knowledge on the topic the user is working on and currently having troubles. The user can then easily connect to one of these users and the two colleagues can then interact and discuss the problem in a chat window that can be opened up. When the user has solved the problem with the help of the experts provides by Aposdle, the user can then create a document which describes the problem he had and the solution for it.

Learn from resources: As Aposdle analyses the activities from the users it can provide them with personalized learning documents on the topics they are working on and could improve their skills. When the provided learning document helped a user, he gets the chance to rate it at the end. If he rates the document positively it will help the system to improve and therefore other users can get even better learning documents.

1.3.1.2 Aposdle's Flow

The system contains three main goals and areas to help users and companies. The system provides knowledge workers

While users are working on task a system running in the background tries to analyses what the current user is currently working on. The system then shows the user suggestions of similar information found in the company's document repositories. The underlying background agent which analyses the actions and tasks of a user, also takes the user's skill level into account and refines the shown documents with this information. This behavior ensures that an employee will always be presented documents and information which match his level of knowledge and therefore the user is not over challenged by the information presented.

To enable such a behavior, the Aposdle project creates user profiles of each employee using the system. In this user profile information about the user and his tasks as well as his skill level is stored and is read and analyzed while the system is running.

Another very important feature of the project is that a user is not only shown relevant documents matching his current tasks, but also a list of experts on the current task is available. The system analyses the current task and skill level of the user and then presents him a list of experts that have knowledge on the needed field and may help the user. With very few clicks the user can contact these experts and then they help to get the work done.

Once a problem was solved with the help of one of the suggested experts, the user is asked to write a documentation of the situation. What where the problems and how were they fixed so that the task has successfully been done.

This is a very important step because if a similar problem occurs in the future the system can then present this solution to the knowledge worker having the problem while working. The system is constantly improved this way and get more effective in helping to optimize the flows in the company it is used in.

The third main feature of Aposdle is the possibility for knowledge workers to improve their expertise with the help of the project. The users will be presented personalized learning modules. As the system knows exactly what the users is working on and it also has information about the skill level of the user, a learning module can be created and presented to the user to help him to improve his skills and knowledge about relevant topics for the work. It is important to know that the user is not forced to go through these presented learning modules.

The system just offers the possibility and users can decide if they want to improve their skills or not.

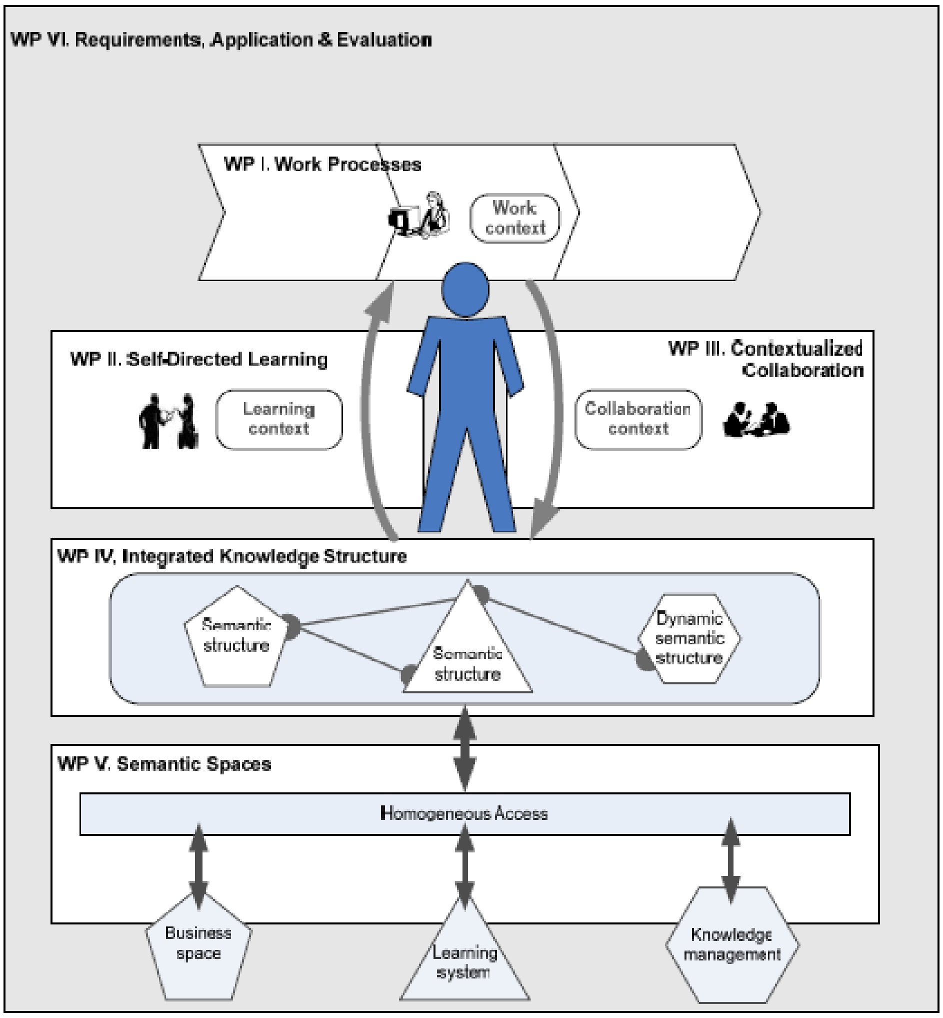


Figure 2: The three spaces which are supported by the Aposdle System.

If a user chooses to learn more about a topic using the learning modules which are created for him, he gets the possibility to rate the learning module. This ranking information is then used to further improve the learning mechanisms of the Aposdle system.

The three spaces model from figure 2 shows the main distinction of the Aposdle approach. Compared to other e-learning applications, Aposdle will support all three roles a knowledge worker can take. According to Lindstaedt et. al [3] the three roles for a knowledge worker are the role of a worker, the role of a learner and the role of an expert. All three get addressed and supported in the Aposdle approach.

The figure also shows the „3spaces“ concept that is used for the implementation of the Aposdle project. Mayer et. al [6] stated in 2005 that Aposdle is using this concept and therefore has the ability to address the environments of business space, knowledge management and learning systems. Furthermore, they mention that normally these three spaces are implemented by separate applications and not one single application. With the use of the „3spaces“ concept the Aposdle approach is able to analyze three different environments of a knowledge worker’s workplace and tasks as one entity.

To make use of all three spaces the Aposdle project applies various existing semantic web technologies like the Web Ontology Language (OWL), Resource Description Framework (RDF), Extensible Markup Language (XML) and other peer to peer technologies.

Aposdle focuses on companies with mostly knowledge workers, also calls workers, as employees. According to Lindstaedt et. Al [4] a knowledge worker is an employee that creates profit for a company mostly with his expertise and knowledge. Normally companies which hire a lot of knowledge workers have products which therefore also consist of knowledge. The workers usually work through a computer or laptop screen to do their work.

The three spaces or roles of a worker that are used and analyzed by the Aposdle application are worker, learner and expert. The following paragraphs discuss these roles in a little bit more detail.

<u>Worker – Work Processes and Context:</u>

A user fulfills the role of a worker with applying knowledge to tasks and therefore create value for the company. Aposdle uses this role of a worker to take care of several work contexts of the user which means the application analyses the work situation the user’s skills and competencies as well as the application domain. Workers, which is meant by the „Worker“ role of knowledge workers in this paragraph, are constantly analyzed and then made aware of learning possibilities and documents from the company’s repository. The Aposdle application makes it possible for the user to access content from various knowledge sources without even having to change the environment. To analyze the user’s context three types of models are needed according to Lindstaedt et. al [3]. They state that these three models are the „learning domain“ as mentioned by Guarino in 1998 [8], „work process descriptions“ as Karagiannis and Telesko described in 2000 [9] and the mapping of work situation to competencies that are needed to accomplish the current task as Ley and Albert showed in 2003 [10].

<u>Learner -Self Directed Learning:</u>

The learner seeks for new information to gain knowledge and tries to learn more about the topics he is working with. According to Lindstaedt et. al [3] a learner also seeks help from experts or other workers. Aposdle is the perfect environment for a user’s learning role. The user is provided with documents and information for self-directed learning and exploration of the topics he is working on. The Aposdle application analyses the user’s current work task and then presents related documents and information from the company’s repositories. Included in the presented learning material are also documents and information that were not originally intended for learning. This kind of learning approach is according to Garrick in 1997 [11] called informal learning or as Berry stated in 1997 [12] as well another term for this learning approach is implicit learning.

Expert – Contextualized Collaboration:

A knowledge worker's expert role is to offer help or advise to other employees and colleagues in the company to help them with their problems. The user is an expert in his field because he learned a lot about the topic and therefore can help and assist others. Lindstaedt et. al [3] state that the most effective learning transfer can be made during communication. Especially when a worker talks to an expert. Therefore, this role is a very important one in the Aposdle application. The system eases the knowledge transfer within a company which reduces bottlenecks, like an expert being on holiday and nobody else can do his work. The definition of communication and collaboration methods that are used by Aposdle are built on the concept of scripted corporation by O'Donnel in 1999 [13] and Dansereau & Johnson in 1994 [14].

[24] mention that according to [25] users can dynamically switch between the three roles depending on the current business process. The border between the three roles are blurred and the users can switch from one role to another. If a user starts working on a certain task and he has no skills and knowledge about the topic which he starts working on, the user automatically becomes a learner. If the user successfully completes his task, then he gains knowledge for this kind of tasks and if another user has to do similar tasks but has no knowledge it is now possible to help this user and become an expert.

1.3.1.3 Architecture of Aposdle

1.3.1.3.1 SOA

The Aposdle system is built with a so called Service Oriented Architecture (SOA). According to [27] applications implemented with a SOA use multiple components that interact with each other using a defined communication protocol. An architecture like that provides a very good scalability of a software as well as an improvement in maintenance of the whole system. Of course an advantage of SOA is that the modules do not need to be executed on the same machine. Typically, the various components communicate over a network. Load balancing is therefore greatly improved in such systems. For example, components that do need a lot of computational power of lots of RAM can be run on a server or machine that can provide the needed resources. If the components would all run on the same machine, a CPU intensive task would reduce the system's overall performance. Another advantage over a „conservative“ one-machine architecture is that in a SOA based system the modules do not necessarily need to be implemented with the same toolset or language. As long as the interfaces are implemented correctly, the module can be programmed in any language. In the chapter Context Search Helper it will be explained that our approach uses a SOA design as well.

According to the Open Group [29] this is the standardized definition of SOA:

> Service-Oriented Architecture (SOA) is an architectural style that supports service-orientation.
>
> Service-orientation is a way of thinking in terms of services and service-based development and
>
> the outcomes of services.
>
> A service:
>
> Is a logical representation of a repeatable business activity that has a specified
>
> outcome (e.g., check customer credit, provide weather data, consolidate drilling reports)
>
> Is self-contained
>
> May be composed of other services
>
> Is a "black box" to consumers of the service

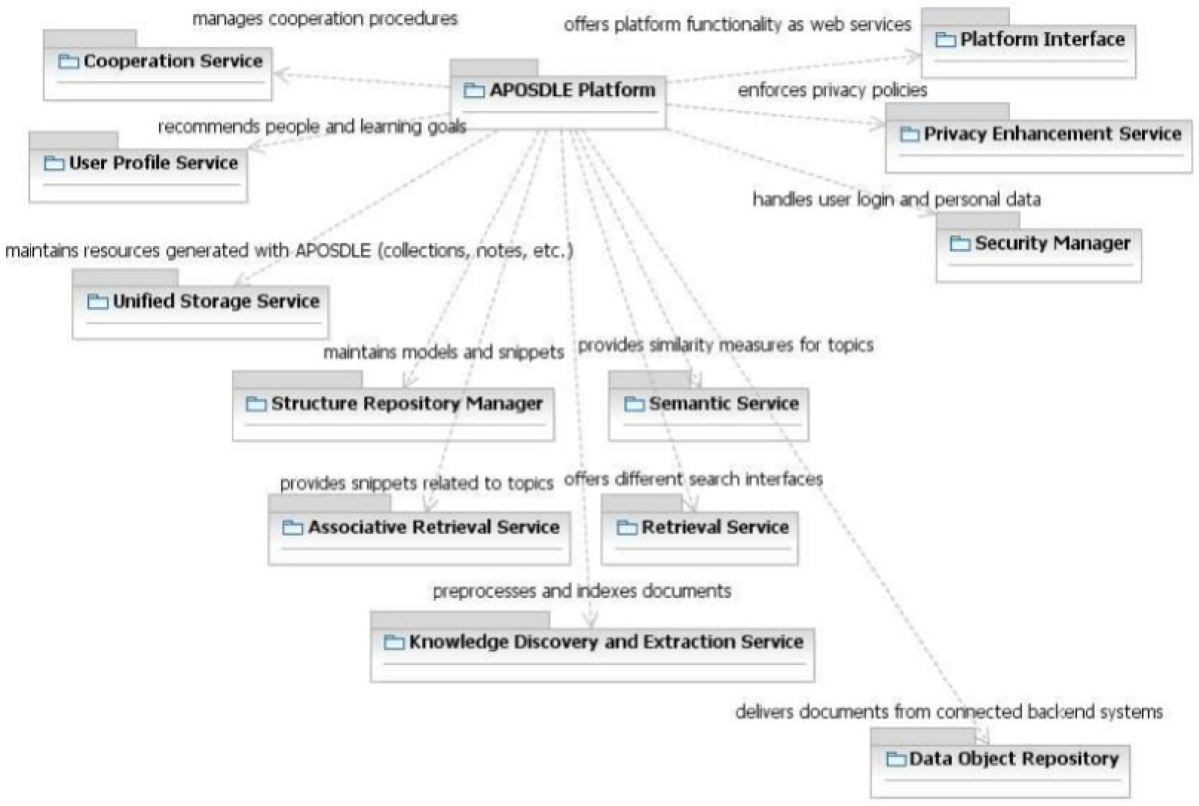

manages cooperation procedures
Cooperation Service
APOSDLE Platform
offers platform functionality as web services
Platform Interface
enforces privacy policies
Privacy Enhancement Service
recommends people and learning goals
User Profile Service
handles user login and personal data
Security Manager
maintains resources generated with APOSDLE (collections, notes, etc.)
Unified Storage Service
maintains models and snippets
Structure Repository Manager
provides similarity measures for topics
Semantic Service
provides snippets related to topics
Associative Retrieval Service
offers different search interfaces
Retrieval Service
preprocesses and indexes documents
Knowledge Discovery and Extraction Service
delivers documents from connected backend systems
Data Object Repository

Figure 3. Shows the high level overall structure of the Aposdle System.

The figure above shows all the parts that are used by the Aposdle approach. As the project is very powerful, complex and therefore very large with lots of modules and submodules each of the parts of the shown figure can fill lots of paragraphs. This would go beyond the constraints of this master thesis. For detailed information about each module of the system refer to [2].

The modules or services that are the most interesting in context of this thesis are the services that are responsible for the detection of the user context and the suggestion of experts to the user. Both services act in a context aware manner, which is one of the fundamental goals of this work.

1.3.1.4 Designing the WIL based user model

The way Aposdle designs the user model is descried in this section. The user's knowledge that is stored in the userprofile of the user can be collected directly by the user but it is also possible to collect the necessary information from inferences made about the userprofiles [20]. To determine the current user's context, the Aposdle project tries to track the so-called naturally occurring actions. Naturally occurring actions are actions performed by the user with a system or software that do not have the purpose of revealing information about the person performing the action. Possible actions that would be classified as such naturally occurring actions are contacting a colleague in the company via a chatting application like

Skype4Bussiness or just scrolling down a webpage [18]. According to Lindstaedt et. al [18] there have been a number of implementations of systems using such a user model trying to detect a user’s navigation behavior or the user’s interest in web pages [21]. According to Lindstaedt et. al [18] it is a problem of the approach described in [21] and others that the implementations are hard to use for adaptive work integrated learning environments such as the Aposdle system.

The Aposdle system therefore tries to create the user model by observing the naturally occurring actions of the user. This implementation is called knowledge indicating events (KIE) by Lindstaedt et. al [18]. An observation that can be made is that KIE are based on usage data.

During research the authors of the [18] paper have found that many companies create and maintain different types of formal models. These models are called enterprise models. The most used models are called:

MODELNAME	DESCRIBTION
WORK DOMAIN MODEL	This model is very often represented as an ontology.
PROCESS OR TASK MODEL	A task or process model is typically represented as a workflow or some kind of process model.
COMPETENCE MODEL, SOMETIMES CALLED A SKILL MODEL	This kind of model is normally represented as simple list or a matrix like structure in a data source.

According to [22] KIE have some limitations. These limitations are that they tend to be imprecise and very difficult to interpret. To draw conclusions based on KIE the Aposdle system uses a hybrid approach. The system uses available semantic information and structures like the enterprise models mentioned in the table above as well as methods like heuristics to guess the actions made by the user. The implementation of this hybrid system can be found in [23].

Lindstaedt et. al [18] found out that there are four types of services needed to fulfill the basic needs of an adaptive WIL system and environment.

SERVICENAME	DESCRIPTION
LOGGING SERVICES	The logging services are the basis for all other services in the system as they update the WIL user model with new KIEs that have just been observed. Sensors within in the system send the data they track, like tasks executions, opening up applications or browsing web pages to the logging services which then update the user model with the

	new data. It is also possible to pre-process the reported user actions to make sure they are stored in the correct format or just add a timestamp or other system related information. The concept of logging services is also implemented in our approach, the Context Search Helper. In our application we named the logging services the Sensor Module.
PRODUCTION SERVICES	The production services are used to provide data for other services which can consume the filtered or aggregated KIE. Production services provide views for the data and other services can get the data formatted or processed in the way the data needs to retrieved. A possible view which can get retrieved by another service in the system would be a list of all tasks the user executed in the last week. This list could then be visualized by another service in the system.
INFERENCE SERVICES	These services are used to process the KIEs as well as interpret the acquired knowledge. The inference services try to interpret the given data and KIE and draw conclusions about the users, like the user's level of expertise. There are possible ways to lead to inferences. One possibility is to apply heuristics on KIE to get aggregated information about the user. The exploitation of KIE would be another way. The two methods could also be combined in a hybrid implementation.
CONTROL SERVICES	According to Lindstaedt et. Al [18] control services are used to control the KIE that are stored in the user model. This is important in the context of privacy. The KIE must be controlled as it important to apply privacy policies of companies. This could be a timestamp that makes sure the data gets deleted after a defined time.

1.3.1.5 The user’s skill level

As mentioned earlier the Aposdle system calculates and stores the skill level of the company’s users. The skill level is useful and important for the expert suggestion feature of the Aposdle approach. If a user works on a specific task, then the user is presented a list of experts within the organization that are skilled in this particular field. The user than has the possibility to contact one of the users to get help with his current work task.

As the paragraph above makes clear calculating the correct skill of a user is an important feature of the application. Lindstaedt et. al [18] calculate the skill of a user in the following manner.

If a user is working on a task, the system queries the user model to retrieve a learning goal vector for this task. In the vector r all learning goals of the domain are stored, no matter if they are required or not to accomplish the task. Another vector k is queried from the user model that represents the user’s skill level of the goals for the first vector. This vector k, which stores the knowledge levels of the user consists of numbers or counters for all the learning goals of the vector r, which are the domain learning goals. Each of the rows in the vector k represents the knowledge of user for a learning goal. The higher the counter in the row is, the more the user knows about this learning goal and is therefore more skilled.

The system then calculates the difference between the knowledge needed to accomplish a task and the skill the user already has. The results are stored in another vector g. The values in this vector are normalized and help the system to search for people with the needed skill for the task the user is currently working on. This is done by the people recommender service.

The user’s skill of a certain task is, as mentioned earlier represented by a counter. Every time a user completes a task of the same field the counter for this particular field is incremented. This way the user’s progress is handled. The next time a user has to do a task he previous successfully accomplished, the vectors retrieved from the user model have been incremented as the user already succeeded in accomplishing this kind of task before.

Of course the higher the skill counter of the user for a specific task gets, the more likely he will get displayed and recommended as an expert for other users in the organization.

1.3.1.6 Process Modelling

It is crucial for systems like Aposdle to implement good process modelling, because this provides background and additional information for figuring out which task a user is working on. Another information that can be gained from process modelling is the tasks that a certain user already has accomplished. [19] state that business companies have applied process modelling in the last few years as part of the company workflow management system. They further say that because of this development over the last decade, embedded learning applications and system have to consider process modelling as a very important and essential part of a real life application scenario. Van Welie [26] is writing that a „task“ is a necessary activity to achieve a specific goal. Task modeling can be done in various ways. One way to model tasks is the so called Business Process Modelling Notation (BPMN) [27]. Another way for this kind of modelling are Petri nets [ii] which are used by the Aposdle application.

Business Process Modelling Notation – BPMN [27]:

The BPMN is very similar to UML modeling and diagrams. There are groups of activities which are logically related to each other. The groups are connected with lines to show the logical relation of the groups. Furthermore, BPMN is an event based way of modelling where events and activities are defined as continuous events [19].

Petri nets:

A petri net is a mathematical representative for distributed systems. There are several mathematical ways to describe models but the fact that the Aposdle system uses Petri nets this is the most important for us. If Petri nets are used for modelling, a discrete distributed system is represented as a directed bipartite graph with additional annotations. Because of this there exist object like place- and transition nodes in Petri nets. Directed arcs are used for connecting place nodes with transition nodes. Petri nets are not event based like BPMN, which has been discussed in the previous paragraph. Petri nets use states for modelling. Thus the current state is represented with token assignments.

To detect the user, task or work context there are more indicators of the user’s context that can be used in addition to the formalized work process model described above. There are several systems that have done a lot of research in this area. For Task detection the CALVIN [28] system for example defines the task context as a term-vector-description. The PINPOINT system [29] is another approach for task modelling. This system provides document recommendations for the current work task. The difference to the Aposdle system or our Approach Context Search Helper is, that the PINPOINT system does not provide automated task recognition. The user selects the task he is working on manually from a list which has been created from domain experts.

The user context is usually more than just the current work task so there has to be put more effort in recognizing it automatically. Not only the current task has to be taken into account, but also information like existing qualifications or skills have to be analyzed. Many systems which have been implemented and try to recognize the user context for expert recommendations are based on domain specific heuristics [19].

Mcdonald et. al [30] designed a flexible system architecture and could therefore benefit from application and domain heuristics in their expert recommender system. Their system requires a supervisor who maintains the profiling, identification and selection. The supervisor has to create and administer the user profiles in the system. To accomplish this task, he uses configurable modules and various data sources. A selection supervisor for example has to filter the list of experts according to dynamically changing strategies and preferences.

The Aposdle system uses a similar architecture than Mcdonald [30] and improved his work so that the recognition of contexts can be done automatically.

1.3.1.7 Task detection in Aposdle

This section will discuss the idea and implementation about the task detection of a user. In the previous section the user context recognition was explained and how [24] implemented it.

[24] see the task detection as a prerequisite of a robust expert recommendation as it is crucial to know what a user is working on to recommend expert on this particular field. Furthermore, they state that detection the users current work task is very difficult as it is considered a machine learning task.

Machine learning in this scenario means that at the beginning of the learning progress the system is untrained and therefore the user needs to manually specify the tasks he works on currently. The user gets presented a predefined list of business tasks, which have been manually selected, and the can choose the task or tasks he is trying to accomplish.

During this first phase of the task recognition detection, the system uses a logging service, as explained in the previous section, to monitor and record every action done by the user. According to [24] such actions can be the pressing of key on the keyboard or the launch of an application on the system as well as reading documents or browsing the web. That way it is possible to tag training material of the user’s working tasks. As soon as there is enough material available, the Aposdle system starts to train a machine learning (ML) model of the user’s working tasks. The results get better when the user continues to work and does not need to select the business tasks manually any more. The better the system is trained the easier it is to notice a change in the working task of the user. The system then loads the new list of experts if the user changed the topic of his task.

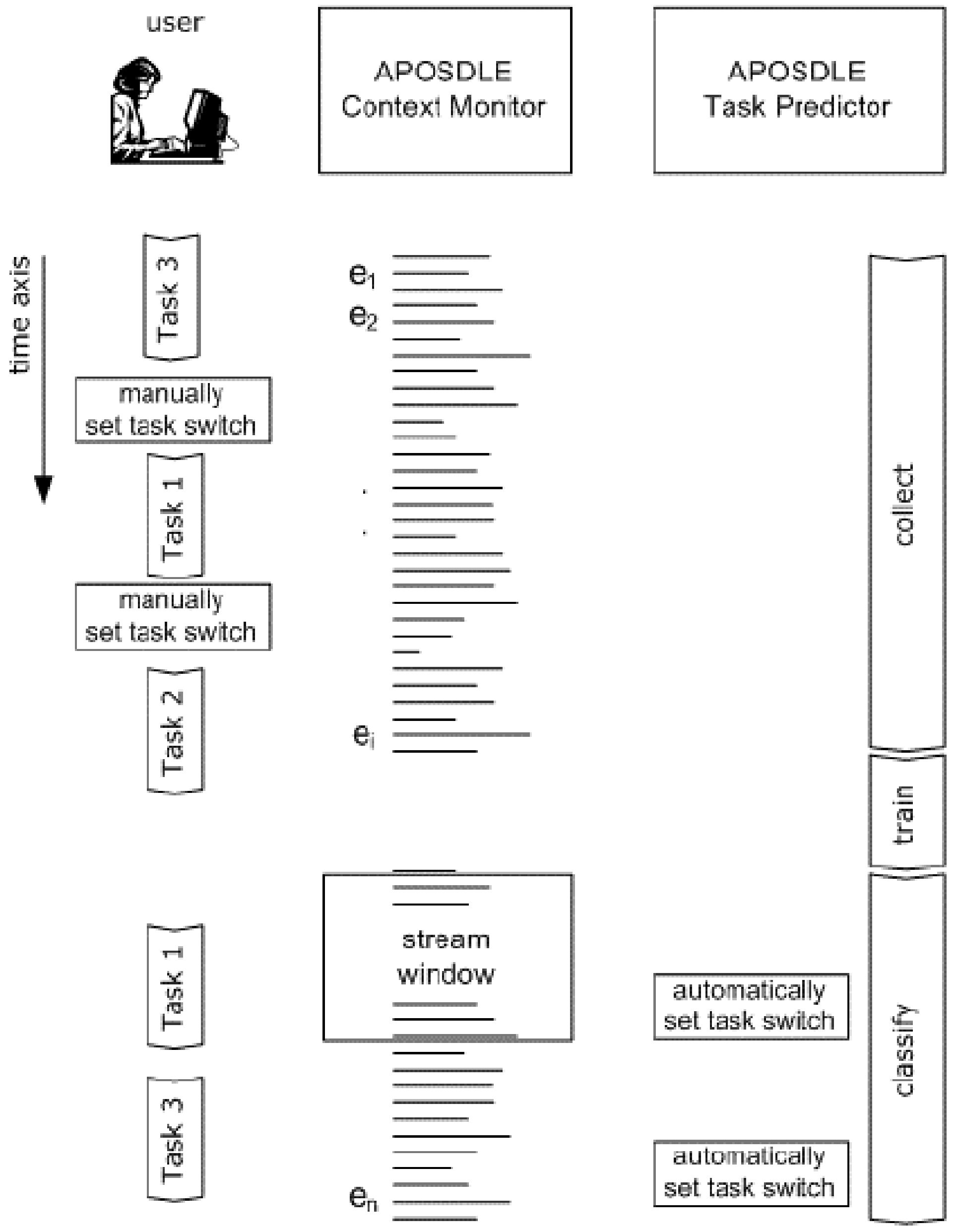


Figure 4. The figure shows the task recognition over the time. First the user has to manually select his task and after a while with enough learning material the system starts to automatically detect the task.

[24] write that the ML algorithms that has been implemented by the developers of the Aposdle system are decision tree learning, rule learning, Naive Bayes and support vector machines (SVM). They also mention that the reason for choosing the Naive Bayes (NB) algorithm was its good overall performance [31]. According to [32] Support Vector Machines (SVM) are a good choice for machine learning when the task is to categorize text. As this is the case in the Aposdle System, SVMs were used with a slight modification to Platt's [33] implementation. [31] say: „SVMs in general are assumed to find a good trade-off between overfitting and over-generalization". The SVM algorithm that was used by Aposdle is the Sequential Minimal Optimization (SMO) which was originally shown by Platt [33].

For the tree learning algorithm, the ID3 implementation by Ross Quinlan [34] has been used. This decision was made because Quinlan's [34] algorithm is known for avoiding over fitting. An early learning technique was the so called rule learning [35]. The algorithm also generates easily readable classification rules and its efficiency is very high according to the authors. The used and evaluated algorithm was the Incremental Reduced Error Pruning (IREP) algorithm [36].

The authors of the Aposdle system evaluated the four mentioned learning algorithms. For the evaluation they used a modelled business process of an application domain. The tasks they decided to choose were tasks like market analyses, product design and specifications, find contact suppliers and contract placement [24]. They formulated the process in the YAWL description language. YAWL is based on Petri nets, that is why it was chosen as description language.

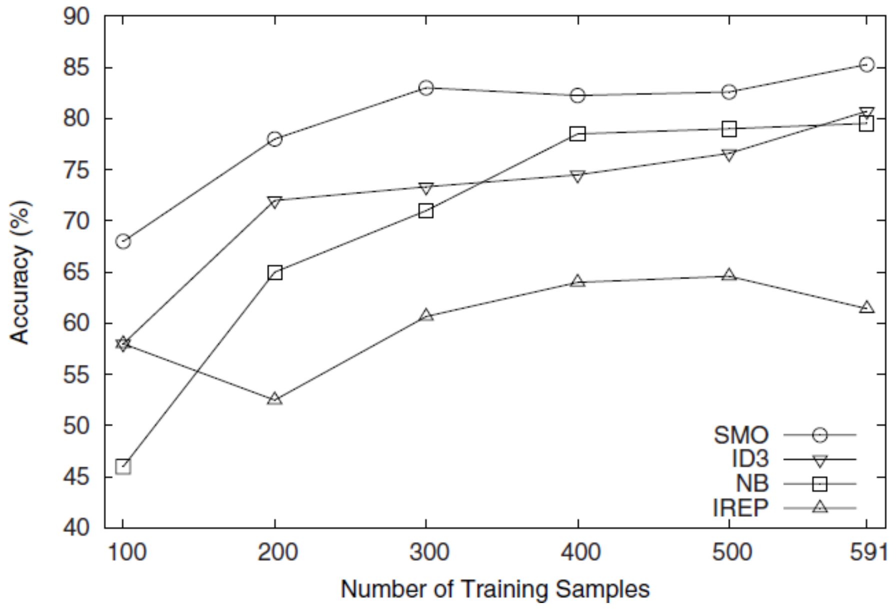


Figure 5. The figure shows the accuracy of the number of training samples for the four evaluated algorithms.

[24] found that the most improvement in accuracy can be seen for the Naive Bayes algorithm. The algorithms that are the least affected by the number of training samples are IREP and Euclid. Figure 5 also shows that starting from 200 training samples, the relations between the algorithms are stable and do not change much. The SMO algorithm performs the best as so be seen in the figure 6 as well. SMO's performance is very good. [24] show with their evaluation that it only needs 20 minutes of recording per task, target labeling for example, for the SMO algorithm to get an accuracy of 83%.

1.3.1.8 Differences to Context Search Helper

The Aposdle project and the system we implemented have a lot in common. Both applications try to improve a knowledge worker’s productivity by analyzing the current working task of the user and provide relevant information about that topic.

Aposdle, as it is much bigger with a much higher budged and more experts working on it, is more advanced in a lot of aspects. The main difference that is probably the most important one is that the Context Search Helper system is not a learning application. Aposdle is based on three parts which focus on work-integrated learning (WIL) and collaborative working. This means that Aposdle uses a company‘s social network to help users connect to others who can help them getting tasks done and solve problems.

The Context Search Helper system on the contrary is strictly a single user approach which does not focus on the learning aspect. The main idea for that system is that a user gets presented relevant information for his current work task and not learn during work. Furthermore, as the Context Search Helper is a single user application, there is no social interaction and therefore the currently logged-on user cannot ask anyone for help if he get stuck during work.

Aposdle also makes use of the skill level of a user and uses that information for processing the data and searching for documents with a matching skill level that are then presented to the user. This information is stored in user profiles. A similar feature is completely missing in our system. The Context Search helper presents documents to the user that match his work task without taking the needed skill level or the user’s skill into account. In a single user environment this feature is less important as the data that is shown is mainly from the user’s computer so we supposed that they match the user’s skill level.

The third part, which is the „learn from resources“part, is also a feature that is not implemented in our system. The users in Aposdle have the opportunity to improve their skills with personalized generated learning documents. In our system we spared to implement a feature like this as we did not want to create a WIL application. The Context Search Helper shall more be seen as a context based resource finder or recommender for the users instead of a learning application.

Another difference between the two systems is the fact that Aposdle can be seen as kind of a living system. This means that it learns from the users who use the application and therefore the guesses for possible matching documents which get presented to the user get better with time. The system constantly improves and the quality scales with the amount of users that use it and the information they enter into the system. When the system is freshly installed in a company environment the users have to manually enter the tasks they work on so that the system can learn from that input. The longer the system is used, the less the users have to enter themselves as the Aposdle project improves with the input.

Our system can be seen as kind of static compared to the Aposdle application. The Context Search Helper does not learn from data or user inputs. Hence its behavior does not change over the time. It always just analyses the task the user is working on and then searches for relevant documents and information. There is no storage for user profiles, which store information like skill levels, or user entered data. As soon as the user logs off from the computer our application clears all buffers and queues so that it starts completely clean the next time the user works on the computer.

As a conclusion it can be said that our system could be seen more like a background agent for Aposdle project or a sub system. The Context Search Helper could provide the information about the current work task of the users and Aposdle uses this information for its calculations and analyses.

1.3.1.8.1 Integrating the Context Search Helper System and Aposdle

As mentioned in the previous section, the Aposdle system is much more complex and therefore offers many features that are not implemented in the Context Search Helper system. Both of the systems are built with a SOA (Service oriented Architecture) and consist of many modules.

This fact makes it easy to integrate the two systems in each other and make the user experience and the features even better. The strength of the Context Search Helper is the fact that it analyzed the user and his current work tasks in „real-time". Real-time here means that the information is never „older" than just a few seconds. If the user switches his context and starts working on a different, new task, the Context Search Helper system adapts to it very quickly and seconds (usually 1-3, depending from the hardware) later, the system has analyzed the new context. There is no need to use learning algorithms and take the users overall work tasks into account. Another feature of our system is, that the context is not only analyzed very fast, but also the relevant resources are shown the user quickly in a non-disturbing way. The Context Search helper system integrates very good with Microsoft Office technologies and applications a like Outlook or Word. When both systems, the Context Search Helper and Aposdle are integrated and work together, the value of the presented information to the user will be improved.

Aposdle uses userprofiles and machine learning algorithms to detect the users' tasks. As mentioned in the previous sections the accuracy is not bad and the system gives good results. We think that Aposdle would profit from a real-time task detection system like the Sensor module of the Context Search helper to improve the already good results. The Context Search Helper Sensors module would analyze the current work task in real-time and would hand this information to the Aposdle system which could take the information into account. The Context Search helper could also be used to quickly detect task changes and could provide the new task context and some metadata to Aposdle.

Another reasonable integration of Aposdle and the Context Search Helper would be in the presentation layer. Aposdle does already provide forms and client applications to show the users important context related resources, experts from within the company or learning material that is personalized for the currently logged in user. Aposdle is designed to help the user in a non-disturbing way. Despite the fact that we also designed our system to assist the users and try not to attract the user's attention too often, the Context Search Helper is a little bit more present than Aposdle.

If the is working in a Microsoft Office Application like Microsoft Word, the user is shown useful resources by our system. We use an Office Sidebar which can be installed as Add In in every Microsoft Office application for the visualization. The sidebar is integrated seamlessly into the Office Products and does not attract too much attraction from the user. The information is only shown the Context Search Helper detects a pause in the work of the user so the resources are only presented, when the user is not typing for example.

Our sidebar is very loosely coupled with the rest of our system and can easily be used by other applications or systems like Aposdle. As more and more companies worldwide use the Microsoft Office Suite, we think that integrating the Context Search Helper Office Sidebar would improve the UX (user experience) of Aposdle.

The last few abstracts we talked about the integrating of Context Search Helper system feature and modules into Aposdle. This is of course also makes sense the other ways around. The Context Search Helper lacks persisted information about the users and their skills. Using this feature from Aposdle and integrate it into our application would definitely lead to improvements in finding the right documents for the user. As our system is capable to search company repositories not all the documents and resources found on such fileshares help the user if he simply has not the correct skill level. If our system would make use of that feature of Aposdle we would be able to present the user even better matching results as his skill and the needed skill for the resources is also taken into accounts.

Another part that would be a useful extension to the Context Search helper is the expert finder which is part of Aposdle. If a user needs assistance in completing his current work task the Aposdle system provides a list of experts to the user which can help him. It would be a good idea to integrate this feature into the Context Search Helper system and show the experts in the Office Sidebar. The Sidebar could show the list of experts to the user or mention the authors of the presented results. If a user clicks a resource in our Sidebar, then the user could be provided a list of experts in that document.

As the previous paragraphs show integrate the two system, Aposdle and the Context Search Helper system, would be a great way improve both of the systems.

2. Context Search Helper

This section will focus on the Context Search Helper application. The idea, the technical background as well as the software architecture and design choices will be discussed in detail. Also the general flow of the application and the connection of all the used modules are described.

2.1 Architecture

In this section the architecture of the Context Search Helper system is explained. First an overview is given, to make it easier to understand the main ideas behind the chosen architecture. Then a much more detailed explanation will be given to fully understand all details of the Context Search Helper system.

2.1.1 Overall Design

The main idea behind the overall Design of the Context Search Helper system is, to be very easily extendable and scalable. The systems make heavy use of a Plugin structure to ensure that all parts are only loosely coupled. This design makes it possible to extend the system's functionality without knowing all the details. It is also not necessary for a developer to rebuild or update the whole Context Search Helper if one of the parts is changed. The architecture is similar to the SOA design used by the Aposdle system.

The mentioned plugin structure is achieved with a module which checks a configured directory for dlls. These dlls are then loaded. If they implement the correct interface the system loads the classes by using reflection. The module which checks for plugins contains a logic of loading the dlls during runtime and merge the return values of the plugins.

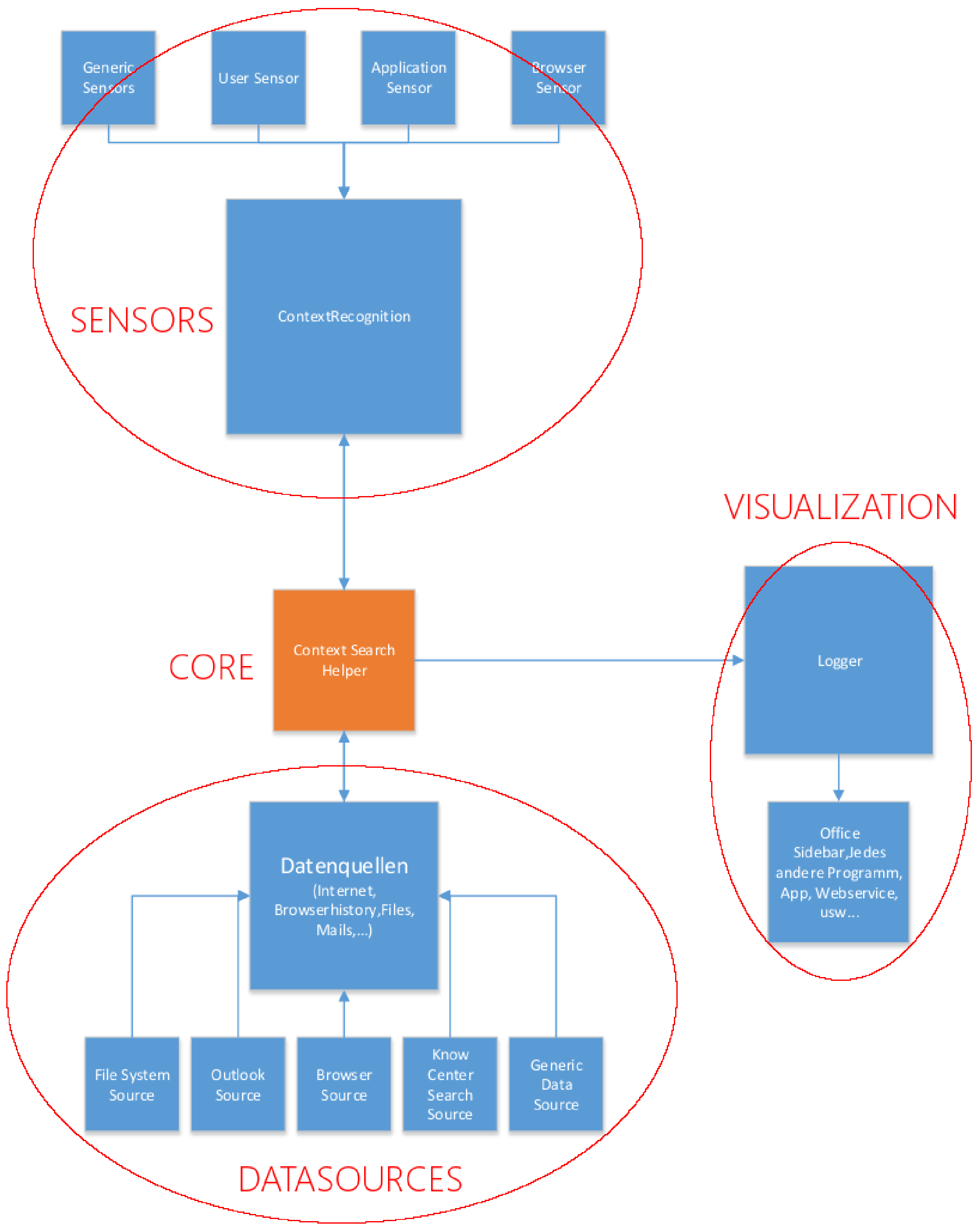


Figure 6: Overall structure of the Context Search Helper system

The Context Search Helper consists of four main parts. The Sensors block, the Datasources block, the Visualization block and the Context Search Helper core. These four main modules are connected by the Context Search Helper block, which is shown in figure 6 as the orange block in the middle of the figure. This block communicates with the other modules and therefore can be seen as the core of the system.

The following tables explains the task of each of the modules.

MODULE	DESCRIPTION
CONTEXT SEARCH HELPER CORE	The core component connects all the other modules. The core component starts up all the other modules and then gives them instructions and fetches data and information from them. In the current version of the Context Search Helper, which is a prototype version, the core module is implemented as a console application and logs a lot of debug information to the standard output. In later releases the core will be implemented as a Windows service.
SENSOR MODULE	This module connects to n sensors which are loaded dynamically during runtime using reflection. Each of the sensors provides information about exactly one part of the user's context. This means that there is one sensor for the user object provided by Windows, one sensor for office applications, one sensor for web browsers and so forth. The sensor module gathers the information from all the connected sensors, combines and normalizes the results and hands them back to the core module.
DATASOURCE MODULE	The Datasource module is used to search for resources once the user's context has been determined. The way it works is the same as the sensor module. There are up to n Datasources which can be connected to the Datasource module. Each of the sources provides its own results. For example, there is a source for web content or one for emails, one for documents on the computer and so forth. The results are then again combined and normalized by the Datasource module and handed back to the core module.
VISUALIZATION	This module is used to display the results coming from the Datasource module. At the moment the visualization is done by a Microsoft Office Side bar which shows the context sensitive related resources to the user. This is done in a way that does not disturb the user.

2.1.2 The Core Component

This component is implemented as a Microsoft Windows Service. It starts automatically when the user boots up his Laptop or PC and then runs in the background without the user even knowing that the system scans the state of the operating system all the time. The Core component runs in an infinite loop and executes the code every few seconds, depending on how much system resources are available. The Core component thread gets scheduled automatically by the operating system, it could vary a little bit on how long the thread has to wait to get computing time of the CPU again.

On low performance machines with few or only single core CPUs it is possible to change the loop iteration time of the core's main loop. If the iteration wait timeout is differed from the default value, which is 0ms, then a few considerations must be taken into account. The longer the thread waits to analyze the system again the higher the risk is that the current context is completely different from the one it was at the time of the last loop iteration.

The core component interacts with all the other modules of the Context Search Helper. In each loop iteration it tells the Sensor Module to check all sensors and report the relevant information and key phrases of the current context. The core then hands the relevant key phrases and named entities it got from the Sensor module to the Datasource Module. The Datasource module searches all sources and returns the results back to the Core. The Core then logs all the data to a queue, which is implemented as SQL table in a Microsoft SQL Server 2012 R2. In this queue the current context, which is represented with key phrases and named entities, the search results from the Datasource module as well as a timestamp are stored.

This information can then be visualized by a client like an Office Add In.

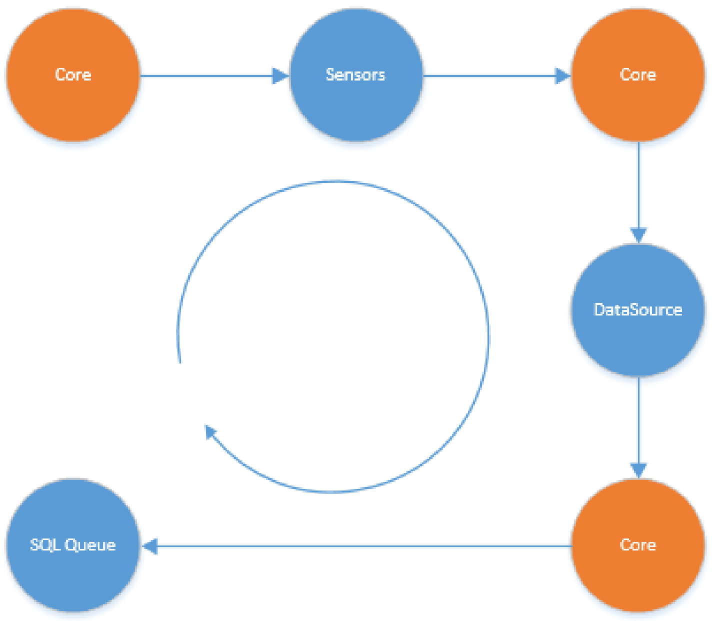


Figure 7: The data flow of the Context Search Helper System. This circle runs endlessly till the application is exited, which is normally the case when the user shuts down the operating system.

In figure 7 the flow of the data in the Context Search Helper system can be seen. The flow shown in the figure represents one loop iteration. The core triggers the Sensors to analyze the system. The information is passed back to the Core module, which then triggers the Datasource module to search for files, mail, etc. The results are returned back to the Core module which then logs everything to the DB.

2.1.3 The Sensor Module

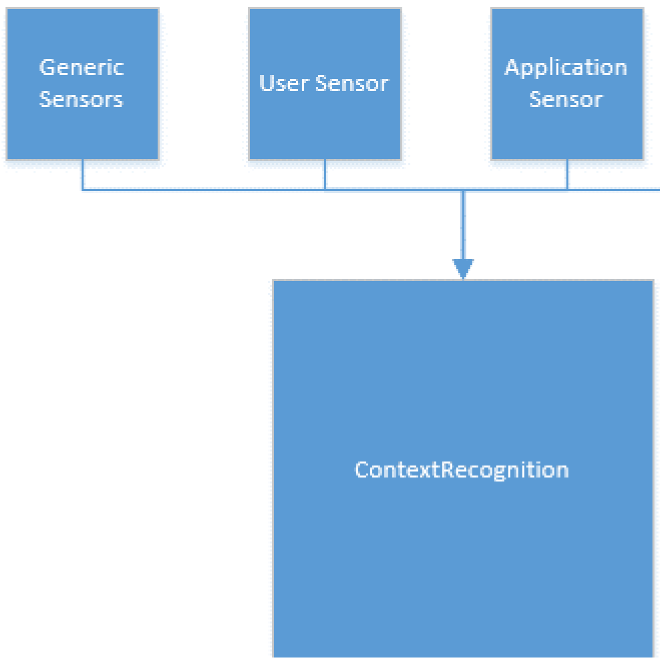


Figure 8: The Sensor module's architecture. The figure shows that the ContextRecognition block retrieves all the information needed by the sensors.

This module is used to analyze the user's operating system's state and figure out what the user is currently working on and what is the current work task. For this purpose, it uses multiple sensors. Each of the sensors has a specific task. In figure 8 it can be observed, that the Sensors module consists of two parts.

2.2 The ContextRecognition block

This block is the major part of the Sensor module and includes all the logic. The ContextRecognition class checks for Sensors in a directory which can easily be configured and tries to load the Plugins in this specific location. The location could either be a folder on the local hard drive or a network share within the company's network or even an ftp location. Additional modules can easily be added to the system by an administrator so all users get the new module. Each of the plugins, represents a sensor which is used to get a specific piece of context from the user or the operating system. If a new sensor is required, it is very easy to add it to the system.

2.2.1 Sensium by Know Center GmbH

The values which are returned by the sensors are then sent to Sensium. Sensium is a natural language processing tool developed by the know center. Sensium is able to process normal texts and language and extracts a lot of very important information from it. It is for example possible to extract a summary of texts or websites. Another example would be the extraction of relevant keyphrases or named entities of a given input text or URL. This is exactly the feature which is used by the ContextRecognition block. All the information gathered from all the sensors are sent to Sensium. After this step only the relevant key phrases and named entities are left as that is the information that gets returned by Sensium.

2.2.2 Ranking

Sensium gives every keyphrase a ranking. This ranking provided by Sensium is also taken into account by the Recognition block. The mean ranking value of the results is calculated. Only keyphrases with a ranking higher than the mean value, are considered to be relevant key phrases. For URLs there is even a stricter value used to separate the important and unimportant phrases from each other. For URLs the mean plus half the mean value is taken as threshold. This is done because websites contain much more information than the other sensor data. So websites from the browser sensor would indirectly be more important. To prevent this the threshold for the browser sensor data is set to a higher level.

More information about Sensium can be found here [a].

2.2.3 The Sensor Interface

For a sensor it is important to implement the interface so the ContextRecognition class can use the sensor without really knowing what is going on inside. A sensor plugin needs to consist of at least one class with the name Context in a C# file called Context.cs. This is the class the ContextRecognition is looking for when loading a Plugin. The Context class must include a function called `GetContext`. The signature of this specific function is as follows: `public string[] GetContext()`. An important requirement for the plugin class to be called is also that the namespace of the class is exactly the same as the library name. Otherwise the method will not be found by the ContextRecognition block.

The code snippet for loading a plugin looks like this:

```
//parse the dll name here
01: string dllname =
pathdoClientSearchdll.Substring(pathdoClientSearchdll.LastIndexOf("\\")).Replace("\\",
String.Empty);

02: dllname = dllname.Substring(0, dllname.Length - 4);

03: var asm = Assembly.LoadFile(pathdoClientSearchdll);
04: var type = asm.GetType(dllname + ".Context");

05: if (type != null)
06: {
07:    MethodInfo methodInfo = type.GetMethod("GetContext");

08:    if (methodInfo != null)
09:    {
10:          object classInstance = Activator.CreateInstance(type, null);
11:          object[] parametersArray = new object[] { };
12:          string[] res = methodInfo.Invoke(classInstance, parametersArray) as
             string[];
```

Listing 1. This listing shows how the Context Search Helper system loads the sensors and data sources dynamically with .NET Reflection.

In line 01 and 02 the name of the plugins is parsed from the file system path to the dll.
Line 03 tries to load the plugin using the full path to the dll including its name. In line 04 the Context class is searched. Line 07 only gets executed when the Context class was found. If this was the case the GetContext method inside the Context class is loaded. If this does not lead to an error line 11 finally calls the method from the dll.

This class gets called each time the context is needed by the Context Search
Helper system.
As the code gets called every time the Context Search Helper Core starts a new loop iteration (see above for more details), it is possible to hot-swap modules. Also updates of module do not need more than just a replacement of the old with the new version of the dll.

Each of the sensors can implement much more classes and even contain complex structures and objects but the connection point between a sensor and the ContextRecognition block has to be `the Context.cs` file which has to contain the `GetContext` function.

It is not possible to pass parameters to the function and the return type of the method has to be a string array. The strings in the array also need to follow a strict format.
The Recognition block tries to parse the strings in a very specific way, so it is important that the strings are encoded correctly because otherwise the Context Search Helper would run into an error while trying to read the return value of the sensor. The error would then be logged with the methods described in section

Error Handling.

A string which is contained in the function's returned array has to have the following encoding: Type;Value.
At the moment two different types of Types are used; Phrase and URL. The value can then either be a phrase, text or sentence or an URL. This differentiation is needed because the ContextRecognition block needs to know how to handle the results of the sensors. A correct string would be "`Phrase;Max Mustermann`". Another one would be "`Url;https://en.wikipedia.org/wiki/Natural_language_processing`".

2.2.3.1 Encoded string vs Strongly typed objects

During implementation a decision was made to use encoded strings and not strongly typed object. There are of course pros and cons of this decision. The drawbacks are not as important as the improvements by this decision and this was the reason strings were used.

Using concrete objects, enums or structs, would require the plugins to know these object so they can create and return them as results. There are two possible ways to get the plugins to know the return object class. Either each of the plugins contains a complete copy of the object class, which could lead to problems when objects are compared for example. The .NET Framework would not recognize the objects as the same object and list and compare operations would lead to problems. Another problem would be the change of the return object. In case the structure or properties of the return object would change or be extended some time, then these changes would be needed to be applied to all the plugins using the object. This would not be a good solution so it was not implemented this way.

The other option is that the return objects are stored in a shared library which is known by the Context Search Helper system and each single plugin. This implementation would not cause any problems during compile or runtime but would have one huge drawback: dependency. All plugins need to include the shared library containing the return objects. Then the plugins would not really be independent from the Context Search Helper System. A developer needs to know the object, needs to include the shared library. The system together with the return object would need to be installed or at least stored on every developing machine. So the plugins would just be classes inside the whole system.

The solution which was chosen where encoded strings. As strings are a very basic, primitive datatype there is no need to share a library with object between the main system and the plugins. To develop a new plugin absolutely no knowledge of the context search helper system is needed. The only thing which must be kept in mind is the encoding, which is very simple to make the system very easily scalable and extendable. More information about the interface is mentioned in the section

2.2.3 The Sensor Interface. This solution also makes it possible to write plugins in other programming languages than C#. As long as the interface is used correctly the Context Search Helper can handle the plugin.

2.2.3.2 User Sensor

This sensor is one which is shipped with the Context Search Helper. This means that it is already included in the software. The sensor is responsible for getting information about the currently logged in user. The idea behind this sensor is that the Context Search Helper can be extended to a multiuser software with features like recommendation. In this case the system needs to also have information about the user currently using it. It is possible to return results and draw the user’s attention to the fact that also other users got the same context or results so the users could then get connected because they obviously work on the same topic. This feature is not implemented in the current version of the Context Search Helper system but it is very high prioritized on the future work and upcoming features.

But the user sensor also can add important context to a single user system without a recommendation feature. The user sensor reads basic data from the Microsoft Windows User object like the user name and the display name. This information can be used by the Context Search Helper to get documents created or changed by the current user.

Another important part of the user sensor is the Microsoft Active Directory (AD) account. In case the Context Search Helper system is installed on a computer which is part of a Windows network, then the user probably has an AD account. This account often has a lot of important additional information about a user. For example, the current user works as a doctor and this information is stored in the user’s AD account. The system then can take into account that the user is working as a doctor and this adds a very important context to the system. The profession makes a big difference in the context and thus results of the system shown to the user. To use a very classic problem of the semantic web here let’s consider a scenario in which the user is working on the topic Kiwi. It makes a huge difference if the user is working as a cook or as an ornithologist.

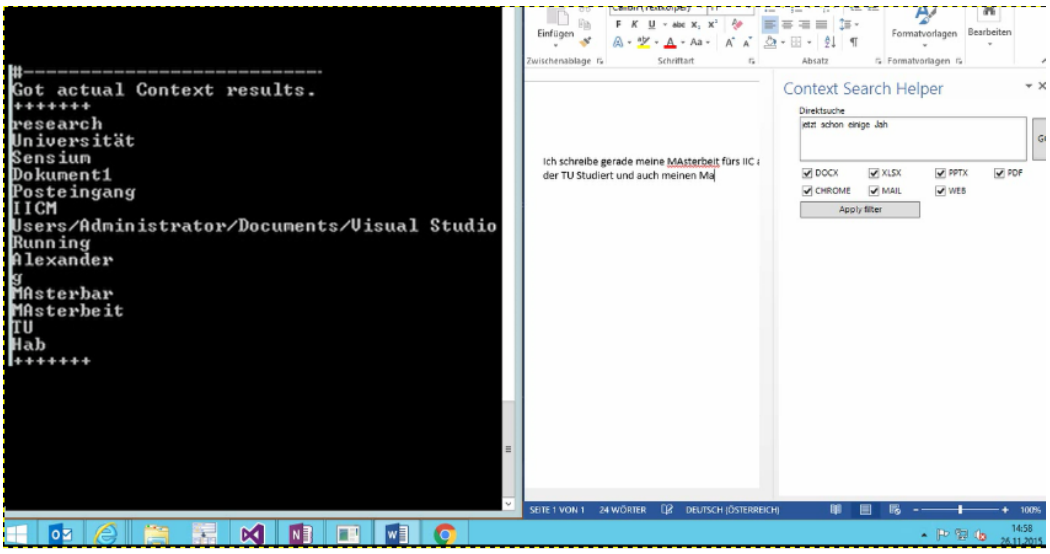
2.2.3.3 Application Sensor

A very important information on what a user is working on are the apps and applications he is using. To gather this high priority information, the Context Search Helper system also includes an Application Sensor plugin for the ContextRecognition module. This module checks the running computer for all active frontend apps and applications. Frontend means that Windows Services, background applications or drivers are not included. This would be totally unusable information for the system. It does not matter which Services are running in the background, because about 100 of them are absolutely equal on all Microsoft Windows machines. Therefore, this information is ignored.

The important and relevant applications are the ones the user is actively using. For instance, the user opens up Outlook and reads an email about some topic he got from a colleague. The application Sensor then gets the subject of the email, because it is opened in a window. In other words, it could be said that the application sensor checks for open windows and gets their title as well as the name of the executable.

Note that the application sensor does not read the context of the windows. In the example above the subject of the email would get read, but not the content. To accomplish this task a new sensor would be needed which would focus on outlook. Such a sensor is planned for the Context Search Helper, but not included at the moment.

The Sensor has no impact on the window titles are created. This is completely a matter of the Microsoft Windows operating system, the Context Search Helper approach just uses the information that is already there.



```
#----------------------------
Got actual Context results.
+++++++
research
Universität
Sensium
Dokument1
Posteingang
IICM
Users/Administrator/Documents/Visual Studio
Running
Alexander
g
MAsterbar
MAsterbeit
TU
Hab
+++++++
```

Ich schreibe gerade meine MAsterbeit fürs IIC
der TU Studiert und auch meinen Ma

Context Search Helper

Direktsuche: jetzt schon einige Jah

[✓] DOCX [✓] XLSX [✓] PPTX [✓] PDF
[✓] CHROME [✓] MAIL [✓] WEB

Figure 9: The screenshot shows the entered text in Microsoft Word is analyzed in real-time. As soon as a user types, the entered text gets analyzed and is used for further processing and resource matching.

2.2.3.4 Browser Sensor

In the section 2.2.3.3 Application Sensor it is mentioned, that the application sensor reads the window title of running applications. In case the window is a browser application, this is not much information. It depends on the implementation of the website which gets opened in the browser if the window title is set. This problem is solved with the browser sensor. The sensor detects open browsers and can extract the URLs of the browser tabs. With this feature a lot of context information can be gathered. If the user is working on some topic and is doing some research about it on the web, like reading Wikipedia [ll] articles or doing Google [kk] or Bing [mm] searches the URLs of the websites will get parsed and analyzed for relevant key phrases and named entities. At the moment the browser sensor supports two browsers.

Microsoft Internet Explorer and Google Chrome are the two supported browser applications. To get the information about the open tabs two very different approaches have to be implemented by the sensor.

To get the open tabs and their URLs from Internet Explorer two libraries by Microsoft have to be used. SHDocVW and MsHtml.

SHDocVW was introduced by Microsoft together with Internet Explorer (IE) 4.0 which is really a long time ago. Nevertheless, the library can still be used to automate the browser. A lot of actions that can be done through the Graphical User Interface of the application can be done in code written in C# and VB.Net. As the Context Search Helper system is completely written in C#, this was the language of choice for the browser sensor. According to the MSDN the SHDocVW can be described like this:

THE SHELLWINDOWS COLLECTION IS DESCRIBED IN THE INTERNET CLIENT SDK AS FOLLOWS: THE SHELLWINDOWS OBJECT REPRESENTS A COLLECTION OF THE OPEN WINDOWS THAT BELONG TO THE SHELL. IN FACT, THIS COLLECTION CONTAINS REFERENCES TO INTERNET EXPLORER AS WELL AS OTHER WINDOWS BELONGING TO THE SHELL, SUCH AS THE WINDOWS EXPLORER

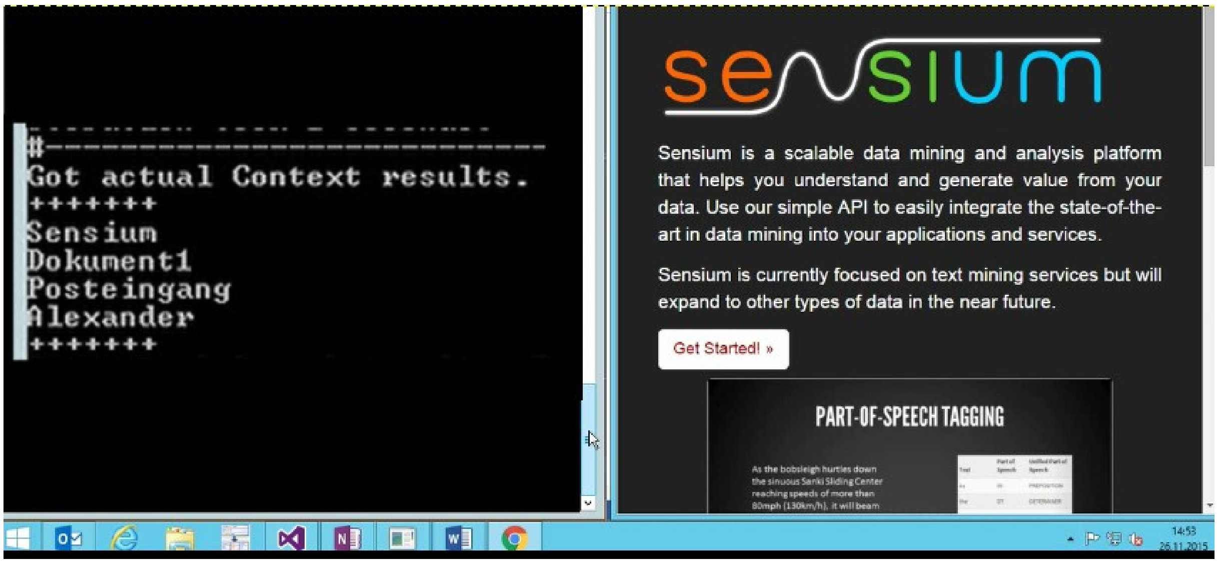
The MsHtml Library which is needed to get the browser tabs from IE is also a class library developed by Microsoft. The library is normally known under another name which is Trident, the Internet Explorer browser’s engine name as well. This dll is also needed to get the useful information like the URL.

To summarize the workflow that is needed to get the IE URLs, the SHDocVW library is used to get a handle to the browser application and the tab windows and the Trident/MsHtml library is then needed to get the URLs and texts from the tabs and windows.

The other browser that is supported at the moment is, as mentioned, the Google Chrome internet browser. To get the URLs of this application a completely different programming approach is needed.

The Microsoft Windows dll user32.dll is needed and because of the fact that this library is a native C++ written dll, the C# marshalling and interop API is used to execute functions inside the C++ code. A handle to the chrome application is retrieved using the mentioned library using marshalling. Then the main window can be access from the code. Note that not like the IE browser, only the currently active tab can be accessed with this method.

Firefox support is planned, but at the moment out of scope. To get this integration working correctly, a third party dll, NDde.client, would be needed.



```
#---------------------------
Got actual Context results.
+++++++
Sensium
Dokument1
Posteingang
Alexander
+++++++
```

Sensium is a scalable data mining and analysis platform that helps you understand and generate value from your data. Use our simple API to easily integrate the state-of-the-art in data mining into your applications and services.

Sensium is currently focused on text mining services but will expand to other types of data in the near future.

Get Started! »
PART-OF-SPEECH TAGGING
As the bobsleigh hurtles down the sinuous Sanki Sliding Center reaching speeds of more than 80mph (130km/h), it will beam
14:53
26.11.2015

Figure 10: The browser sensor in action. The currently active tab shows the Sensium web page. The debug output on the left side shows that Sensium is immediately recognized as context relevant and taken into account for further processing. It can also be seen that the other context phrases are the opened up applications like the name of the opened word document or the mail inbox.

2.2.3.5 Generic Sensor

The generic sensor is the last plugin which is included in the Context Search Helper. Simply put the generic sensor just reads entries from a MSSQL table which acts as queue and hands it to the ContextRecognition block.

To understand why the generic sensor is helpful an example might come in handy. Imagine the following situation:

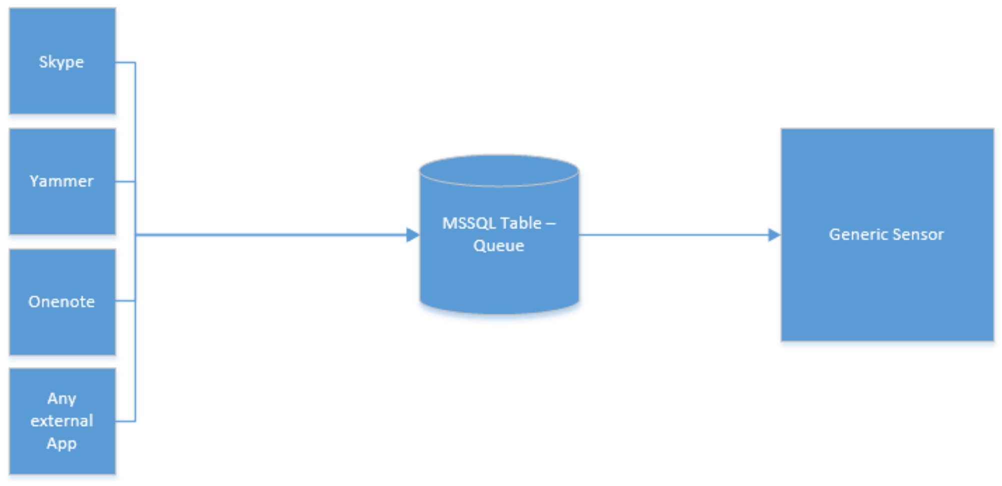


Figure 11: Workflow of the and architecture of the Generic Sensor that is implemented.

A user works in a big company with multiple office locations and he works as a software engineer. Even the engineering team is split up and not all colleagues work in the same office. But to get all the work done and to work together as a team, the company uses two Microsoft products, namely Skype for Business [b] and Yammer [c]. The engineers discuss a lot of important topics about projects over these two communication channels.

The Context Search Helper is able to get the information about the fact that the users are chatting with each other over skype or discussing topics in yammer, but the sensors do not provide a possibility to get the information which is sharing in these two applications. For the company in the scenario this is very important information and therefore it would be very valuable to also include the Skype and Yammer messages in the context analysis.

Because of the plugin and loosely coupled architecture of the Context Search Helper, it is easy to add this valuable context to the system. The first possible way to achieve this would be a new sensor which can handle Skype and one sensor solely for Yammer. Due to the architecture this would not be a big deal but the Generic Sensor opens up an even faster and easier method to access the information in completely unknown external applications like Yammer and Skype4Business. To use the Generic data source not even programming skills are required. All that is needed is to add data to a SQL DB. Also web developers which do not have the skills to code in C# or Java can use this Data source from within their websites.

The idea is that Yammer and Skype need to store their information and data inside a SQL table. This can be done with a few lines of code in Yammer and Skype. These applications then store all the messages which are sent and received in the SQL table and the Generic sensor just needs to read this table. It does not matter for the Context Search Helper which external applications exist, it just extracts the information from the SQL table and gets additional context to process.

So the generic Sensor opens up a very easy way to add context from numerous completely unknown applications to the system. As a matter of fact, the Context Search Helpers Visualization module itself makes use of the Generic Sensor to make it possible to change the visualization client very easy. More information about this module and the use of the Generic sensor can be found in section 2.4 Visualization.

2.3 The Datasource Module

This part of the Context Search Helper system is the second powerful and complex module. It gets an array of relevant key phrases and named entities as input and then searches various sources for relevant results.

2.3.1 The SearchProxy class

The Datasource module basically consists of two parts. The main class which contains nearly the entire logic called `SearchProxy.cs` and an undefined number of Datasource plugins which can be used to search various sources for results. The architecture is exactly the same as in the Context Recognition block. The main module can be configured and given a path to a location where plugins are located. The `SearchProxy` class then tries to use all of the dlls or plugins at that location and use them to search for files, mails and other results.

The `SearchProxy` class gets encoded strings back and tries to parse them. It should be mentioned that it is absolutely crucial for the plugins to implement the interface and return correctly encoded strings, because otherwise the `SearchProxy` would get an error trying to read the information and thus the information of the sensor would be useless.

When the plugins have all returned their search query results, the scores need to be normalized. Not all search plugins return the relevance in the same range, like the Windows search returns relevance values in the range from 0 to 1000 and the Know Center search in the range from 0-10. So an important step done by the Datasource module is normalizing the results so that all relevance values are between 0 and 1.

Note that at the moment there is no merging of the results. That means that an email returned by the Outlook Datasource with a relevance of 1 is equally important as a web search result with the same relevance. There is no weighting of some sort implemented at the moment, but planned for future releases.

After all plugins have been loaded and the search results have been normalized, the Datasource module stores the results as well as the key phrases into a SQL table. This table is important for debugging and calculating the quality of the results. In the table the phrases and results are stored together with a timestamp for easy debugging.

The search results are then given back to the Core component of the Context Search Helper application. As mentioned in the section 2.1.2 The Core Component, the results returned by the Datasource module are then stored in a SQL table as well. This table is updated every time the module gets new results from the search plugins. On the used test environment this was about every few seconds.

2.3.2 The Plugin Interface

The Datasource module can be easily extended with plugins to access more search sources. To add a plugin, the only thing needed is to build and dll and place the dll in the folder where the Datasource Core component searches for all the available plugins. There is not even the limitation to C# and .NET. As the interface is very simple and basic, it is possible to add plugins developed in other programming languages like Java. This is only possible because the interface is implemented using primitive datatypes and has absolutely no dependencies. The idea is exactly the same as the interface used in the ContextRecognition block. The learn more about the reason why the interface is implemented using very basic language features, please have a look at the section

2.2.3.1 Encoded string vs Strongly typed objects.

Only copying the portable libraries, is not enough of course. The Datasource module tries to load the dlls at runtime using the .NET reflection feature und then searches for specific classes and methods in the loaded plugins. To make this mechanism working correctly it is very important that the specific plugins implement an interface which is knows by the Datasource module.

If a plugin shall be accessible from the Datasource module it needs to contain a least one Class named Search. This file and class is searched by the Datasource Core component. The Search class has to be in a specific name space. The namespace needs to be exactly the same as the name of the library, because otherwise it would be impossible for the system to load the plugin during runtime. In this class there needs to be at least one function. This function is called by the system. The signature of this specific method is: `public string[] DoSearch(string[] phrases)`. The methods get a string array with phrases as input. The values in this string array are the relevant key phrases and named entities which have been extracted beforehand and handed over to the Datasource component. It is very important that each of the plugins get these values as parameter, because otherwise it would not be possible to perform a search on the source. The method also needs to return an array of string. The strings which are stored in the array, also need to be encoded. Such an encoded string needs to be formatted in the following way: “Type;LocationPath/ID;Score;Displayname[optional]”.

2.3.2.1 Type

The type is needed mainly for one reason. It makes it possible for the visualization component to render a correct icon for the result set. How this is done can be seen later in this document in the section about the visualization section. There are a few types which are supported out of the box by the system. These icons result from the various plugins and result set types shipped with the application. The types are:

TYPE	DESCRIPTION
DOCX	This type represents Microsoft Word documents of the versions 2007, 2010, 2013, 2016
XLSX	This type represents Microsoft Excel sheets of the versions 2007, 2010, 2013, 2016
PPTX	This type represents Microsoft PowerPoint presentations of the versions 2007, 2010, 2013, 2016
MAIL	This result type is used for Emails found in the configured mail accounts in Microsoft Outlook of the versions 2007, 2010, 2013, 2016
CHROME	Results with this type have been found in the Google Chrome browser history of the user
PDF	PDF documents are represented using this type
TXT	Simple text Files use this type
WEB	All web search results are using this type at the moment.

2.3.2.2 LocationPath/ID

This part of the encoded string is probably the most important one. The part of the string stores either an email ID or a path to the File or URL. The visualization component can use this property to use Windows API built-in commands like `Process.Start();` Opening all types except email only need this one line of code. The parameter of the `Start` function can simply be a path to a word or pdf file for example. Microsoft Windows will look up which is the default application for this file type and will open the application and load the file. This workflow works perfectly for all supported types by the Context Search Helper application.

As mentioned above, emails need an extra handling in the visualization component. As it is not possible for the email data source to return .eml files, which would be portable email files, it is also not possible to open them with `Process.Start().` The Email data source returns only email IDs, which then have to be looked up with the Microsoft Outlook API. More details about this can be found in the section about the email data source.

2.3.2.3 Score

The score represents the relevance of a search results. It is important to return values between 0 and 1. Otherwise there would no usefully sorted result list be possible. If all sources return values between 0 and 1 the results are far better comparable. It was a design decision during the implementation that the normalization logic is done within each plugin and not in the data source core module. An important note is that results from different sources with the same score are considered equally important. There is no weighing in any kind between the sources. This feature was not meant to be in the first release version of the application but is on the to-do list for future releases.

2.3.2.4 Displayname

The display name is an optional property of the encoded strings, which is currently only used by the email data source. It is a way to change the text which is displayed on the result entry in the visualization component. As emails are handled which their ID and not a filename, like the other results, the email source uses this property to set a user friendly text for the results, so the user does not need to see the ID, which is very long and unreadable.

2.3.3 File System Source

This source searches for files which match the given search phrases on the local computer in the file system. The plugin makes heavy use of the Windows Search API, which is documented in the MSDN and can be found at the link [gg].

The Search APIs is implemented in a way, that it can be easily be queried with ADO.NET functions and SQL like statements sent to the Search API. Instead of using a normal SQL Database connection string the following connection string is used to connect the source to the Windows Search Indexing Database: `string connectionString = "Provider=Search.CollatorDSO;Extended Properties=\"Application=Windows\"";`

With this string an OleDBConnection Object is created and using this object the Windows Indexing Database can be queried.

The columns that are queried by the source are `System.ItemPathDisplay,System.ItemType,System.Search.Rank`. A full list of all possible columns to search can be found here [l].

Another important part of the query is the specified source to search. In the SQL like query used by the file system source this is the SystemIndex. The system index is built by windows and constantly updated. Depending on the system wide search settings specific file types can be ignored by the search or the Internet Explorer history is included in the SystemIndex. Also full texts can be indexed. The according settings can be found in the Control Panel or the advanced settings in the Windows 10 tablet system settings.

The next important part of a search query of the file system source is the `WHERE` statement. This is the place to filter the SystemIndex for specific file types or filenames and content.

The file system data source also makes use of the `Scope` parameter. This parameter is used to restrict the result files to a specific directory. If the parameter is not specified in the query, the whole file system is searched for results. Restricting the place to search improved the overall performance of this data source plugin. In a release version of the Context Search Helper system the Scope parameter should be configurable in a settings dialog so the user would have the possibility to restrict the crawled locations to a folder where all the relevant documents are stored instead of also crawling directories like C:\.

Before returning any results of the SystemIndex the file system data source also uses the `SORT` parameter of the SQL like Windows Search API language to return the results in a descending order according to the relevance. At the moment there is a little problem with the ranking information of items returned by the SystemIndex. According to the link [m] the rank of files that did match the filename or other attributes than full text content matches, is 1000. That means that documents which are found because of full text content matches are always lower ranked than documents which just match the name or part of the name. There is possible improvement in the implementation of this plugin at this area. By using the `Rank By` clause of the Windows Search API, the ranking behavior can be overwritten, but this is very difficult and may cause a lot of errors and problems so it was not done in the current implementation of the file system data source.

The following listing shows a complete example code for a query to search the SystemIndex:

```
SELECT TOP 1000 System.ItemPathDisplay,System.ItemType,System.Search.Rank

FROM SystemIndex

WHERE (System.ItemType = '.pdf' OR System.ItemType = '.docx' OR System.ItemType = '.xlsx'
OR System.ItemType = '.pptx')

AND

(

                    ( FREETEXT('Phrase1') OR FREETEXT('Named Entity') )

          OR

                   ( System.ItemName CONTAINS('Phrase1') OR System.ItemName
                   CONTAINS('Named Entity') )

)

AND SCOPE = 'C:\Users\Administrator\Desktop' ORDER BY System.Search.Rank DESC
```

Listing 2. This listing shows the SQL like query that retrieves results from the Microsoft Windows built-in Search Index DB.

2.3.4 Outlook Source

The outlook data source searches in Microsoft Outlook for emails matching the given phrase or named entities. To accomplish this task there are settings which have to be set correctly in Outlook. In the Trust Center of Outlook, it has to be configured that external applications can access Outlook data. This is needed because otherwise there would be a Popup asking the user if he really wants to access Outlook data each time the data source searches for emails. The mentioned security mechanism is implemented in Microsoft Outlook, that viruses or other malware cannot easily access email data from the user. It is important to mention that the Popup only appears if Outlook is not running on the client computer. If the user has Outlook opened, then there is no such problem.

This is because the data source check is a there is already an instance of Outlook running on the computer. If this is the case, then this Outlook application instance is used to search for emails. If no client application is opened the data source opens a new Outlook instance in the background and uses it for its purpose. The code for this logic can be found on the MSDN and looks like this:

```
Outlook.Application GetApplicationObject()
 {

     Outlook.Application application = null;

     // Check if there is an Outlook process running.
     if (Process.GetProcessesByName("OUTLOOK").Count() > 0)
     {
         try
         {
             // If so, use the GetActiveObject method to obtain the process and
             cast it to an Application object.
             application = Marshal.GetActiveObject("Outlook.Application") as
             Outlook.Application;
         }
         catch
         {
             //log to windows logs
             application = new Outlook.Application();
             Outlook.NameSpace nameSpace = application.GetNamespace("MAPI");
             nameSpace.Logon("", "", Missing.Value, Missing.Value);
             nameSpace = null;
         }

     }
     else
     {

         // If not, create a new instance of Outlook and log on to the default
         profile.
         application = new Outlook.Application();
         Outlook.NameSpace nameSpace = application.GetNamespace("MAPI");
         nameSpace.Logon("", "", Missing.Value, Missing.Value);
         nameSpace = null;
     }

     // Return the Outlook Application object.
     return application;
 }
```

Listing 3. Original code recommended by Microsoft as best practice for getting an Outlook instance using .NET and C#.

Once the data source has connected to an instance of Outlook, either a running or a new one, the default inbox folder is queried. A possible improvement of this data source would be that all folders, or manually specified folder are searched. As mentioned above at the moment the default inbox folder is used to search for emails.

To get emails and conversations matching the keyphrases and named entities which have been passed to the source as parameters of the `DoSearch` method, the `Restrict` function of an `EmailItem Collection` is used. This has the same effect as typing the phrases into the search box visible in Outlook. The function returns a subset of the current Item collection which only contains emails which match the query. The query which is used makes use of the `urn:schema:httpmail` Namespace of Microsoft. For more details about this Namespace please follow this link: [n].

The current implementation of the email data source uses these schemes: `subject`, `body`, `from` and `sender`. That means that the subject of all emails in the default location will be searched for matches with the given key phrases as well as all the bodies of the emails. Emails which have the property `from` or `sender` set to a matching keyphrase are also included.

An example query which is used to restrict items in the default email folder looks like this:

```
@SQL=

(

        "urn:schemas:httpmail:subject" LIKE '%PHRASE1 %'

        OR

        "urn:schemas:httpmail:body" LIKE '% PHRASE1%'

        OR

        "urn:schemas:httpmail:from" LIKE '% PHRASE1%'

        OR

        "urn:schemas:httpmail:sender" LIKE '% PHRASE1%

)
```

Listing 4. The SQL like query to retrieve search results from Outlook and query mails, contacts or other PIM objects.

An important note to mention here is that the current implementation of the Outlook data source is that it is not making use of a ranking here. All emails found by the Datasource get the same relevance, which is hardcoded set to 1. This is a point where improvement will be done in future releases of the Context Search Helper system. Due to the flexible architecture of the system it is only needed to change or add code to the Outlook data source to enable ranking of emails.

2.3.5 Browser Source

The browser data source is used to search the history data of the system's browsers for information and websites which match the current context of the system. The source is meant to search the data of all browsers which are installed on the client. This is very difficult because each browser has a different API and the information is sometime protected by security mechanisms. That is the reason that at the moment, the current version of the browser data source only queries the history data of the Google Chrome browser. The Chrome browser was selected as the one used in the current release version of the data source because of the market share. As shown in figure 12 Google Chrome holds about 60% of the market according to [o].

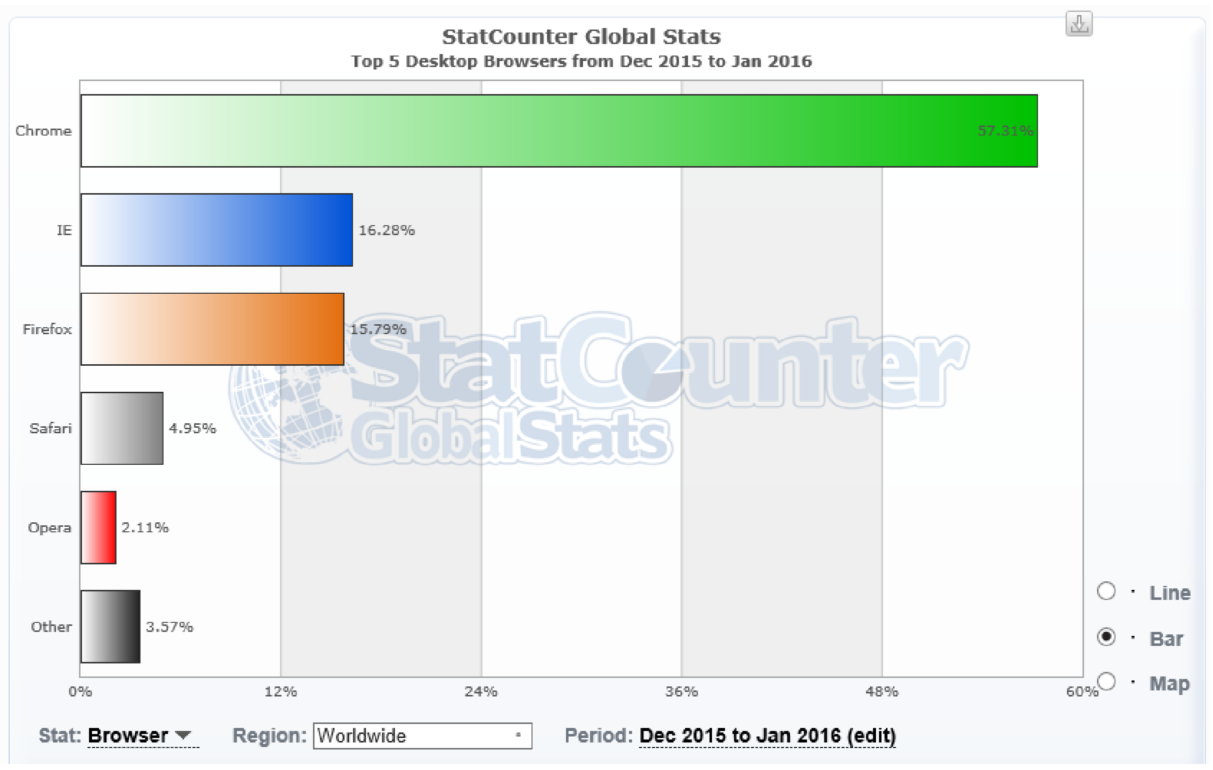


Figure 12: The market share information of desktop browsers from december 2015 to january 2016

In future releases support for other browsers is panned.

This can be achieved with two different ways. It is possible to enhance the functionality of the browser data source and include multiple classes. Each of these classes communicates with exactly one browser. Another class which acts like a kind of dispatcher or proxy would then be needed. This class would get the context phrases and calls each of the specific browser classes. Then the results would be merged and returned to the Context Search Helper. This design and implementation choice would lead to a very complex browser source which would contain multiple classes and logic.

The better choice for an integration of more browsers would be to rename the current browser data source to “Browser_Chrome_Source”. Each of the other supported browsers would get an own data source like “Browser_Firefox_Source” and so forth. This design choice would make use of the modular architecture of the Context Search Helper system. It would also be easy to append new browsers or change existing ones. There would not be one single data source with lots of code and logic, but small and lightweight ones for each browser.

In this paragraph the functionality and implementation of the Google Chrome data source will be discussed.

Google Chrome stores all kind of user data in a SQLite [q] database file. Inside this file there is stored all the user related information like downloads, site visits, the browsing history and so forth. This SQLite file can easily be read by using the SQLite API available for C# via the Nuget package management inside Visual Studio. The SQLite database file with the important information for the data source is stored inside the users “home” directory. The source copies the file from this location to a temporary location. This is done to prevent race conditions or file locks in case the user is using the Google Chrome browser at the moment the data source tries to read the information. The file is copied the following way:

```
string userpath =
                System.Environment.GetFolderPath(Environment.SpecialFolder.UserProfile);
string filetoCopy = userpath + @"\AppData\Local\Google\Chrome\User
                               Data\Default\History";
string fileAfterCopy = Environment.CurrentDirectory + @"\ChromeHistory";
File.Copy(filetoCopy, fileAfterCopy, true);
```

Listing 5. This listing shows how the SQLite file that holds all the Google Chrome user data is copied to the current directory where the application is running.

After the file has been copied the data source opens the SQLite file and searches it with a query formatted the following way:

```
select url
from urls
where
 (
            (title LIKE '%*PHRASE1%' or url LIKE '%*PHRASE1%')
      or
            (title LIKE '%PHRASE2%' or url LIKE '%PHRASE2%')
)
LIMIT 30
```

Listing 6. Shows the Query for retrieving the user's Google Chrome history data.

In the query above it can be seen that the table used to search is the “urls” database table. This table consists of the columns of the table below.

COLUMN	TYPE	DESCRIPTION
URL	LONGVARCHAR	This column contains the URL of the website.
TITLE	LONGVARCHAR	The title of the website in the history is stored in this column.
VISIT_COUNT	INTEGER	The number of site visits is stored. This information could be used for a better search query. Sites which have a higher visit count could be ranked higher that

		sites which were only opened once for example.
TYPED_COUNT	INTEGER	This column stores information about how often the user actually typed the URL.
LAST_VISIT_TIME	INTEGER	This information could also be used for a more optimized search query. Sites which have been visited more recently could be ranked higher than “older” sites in the user’s history.
HIDDEN	INTEGER	Gives information about the visibility of the history site. If the flag is set to true, the history entry is not displayed in the browser. This flag has no influence if the data is queried via C# code.
FAVICON	INTEGER	The favicon of the page.

As there is no ranking information stored in the SQLite database file, all search results get the same ranking, which is set to 1 hardcoded. An improvement for the Google Chrome data source would be a ranking which takes the last visited date and time as well as the visit counter into account. This feature is planned for future releases and is currently not available in the latest version of the Context Search Helper application.

2.3.6 Know Center Source

This data source is used to search the web for results according to the given key phrases and named entities. There were a few options to query the web for information matching the user’s context. A lot of different search engines exist and each of it has its own advantages and disadvantages over the other ones.

Obviously using Google Search would be a good guess for good web search results. To accomplish such a task like querying a big Search engine like Google or Microsoft’s Bing an Application Programming Interface (API) key is needed which is bound to a user account. Otherwise an application which sends requests every few seconds would be considered to be a bot within a few seconds. API keys with unlimited usage of data are not easy to get so Google or Bing were not used for the data source.

There is another simple reason why none of the “big” famous search engines were used in the web search data source. The know center has developed a very own search engine and API which can easily be queried from within C# code. All it needs are simple web requests.

The know center search also includes a web frontend and can be used to query data from news around the world regarding a wide range of topics. A lot of additional filters exist to restrict the query and improve the results. The location of the Know Center Search is [r] https://saas.know-center.tugraz.at/news-demo/#/knowminerSearch. To learn more about all the functions and filters as well as the API interface the given web site can be visited.

The following listing shows the code which is used to get information from the Know Center Searchminer search engine.

```
using (var client = new WebClient())
{
       try
       {
              client.Headers[HttpRequestHeader.ContentType] = "application/json";
              client.Headers[HttpRequestHeader.Accept] = "application/json";

              string toupload =  "{" +
                                  "\"indexName\": \"news\"," +
                                  "\"query\": \"" + phrase + "\"," +
                                  "\"searchSettings\": { \"chunkSize\": \"30\"  }" +
                                 "}";

              var response = client.UploadData("https://saas.know-
                                          center.tugraz.at/news-demo/api/search/search",
                                          Encoding.UTF8.GetBytes(toupload));

              var responseString = Encoding.UTF8.GetString(response);
              dynamic obj = JsonConvert.DeserializeObject(responseString);


              var sh = obj.searchHits;
```

Listing 7. Shows the code for calling the Searchminer and retrieving values form it.

The first important thing is to create a `WebClient` object. This is a basic class inside Microsoft’s .NET framework and can be used to make calls to websites or web services. There are a lot of properties functions and parameters which can be used when working with the WebClient object. The Know Center Search data source only uses a very limited subset of methods and properties. One of these properties is the `Headers` array. This array represents the headers which are sent to the web service or website to tell it what the data which is sent looks like and what format the response needs to be in to be able to be read. The Know Center data source communicates with the search engine in the JavaScript Object Notation (JSON) format so the headers are set to “`application/json`”.

The next thing which is important is the payload of the web call that gets sent to the service. As the headers tell the search engine that the payload is formatted in the JSON format, this the way the payload needs to be formatted. It consists of the name of the database to be searched. The Know Center search data source only looks in the “news” database for relevant results. Also included are of course the key phrases and named entities the Context Search Helper application has collected from the system.

The message is then sent to the search engine by using the `UploadData` method of the `WebClient` object. Note that it is important to encode everything in `UTF8`, because otherwise the WebClient and the search engine would not be able to communicate with each other.

The response of the service is then deserialized using the JSON helper classes of the Newtonsoft JSON product which is free to use and can be found at this location [t].

An important step of the processing of the web search results is the normalization of the relevance. The Know Center search engine ranks the results from 0-10. To be able to compare and merge these results with the ones from the other Datasource it is very important that the relevance is normalized from 0-1.

2.4 Visualization

This section is about how the data is stored and passed from the Context Search Helper subsystem to the User interface. The concepts of the transport and storage of the data will be discussed as well as the possibilities to visualize the data to the user.

2.4.1 Client applications

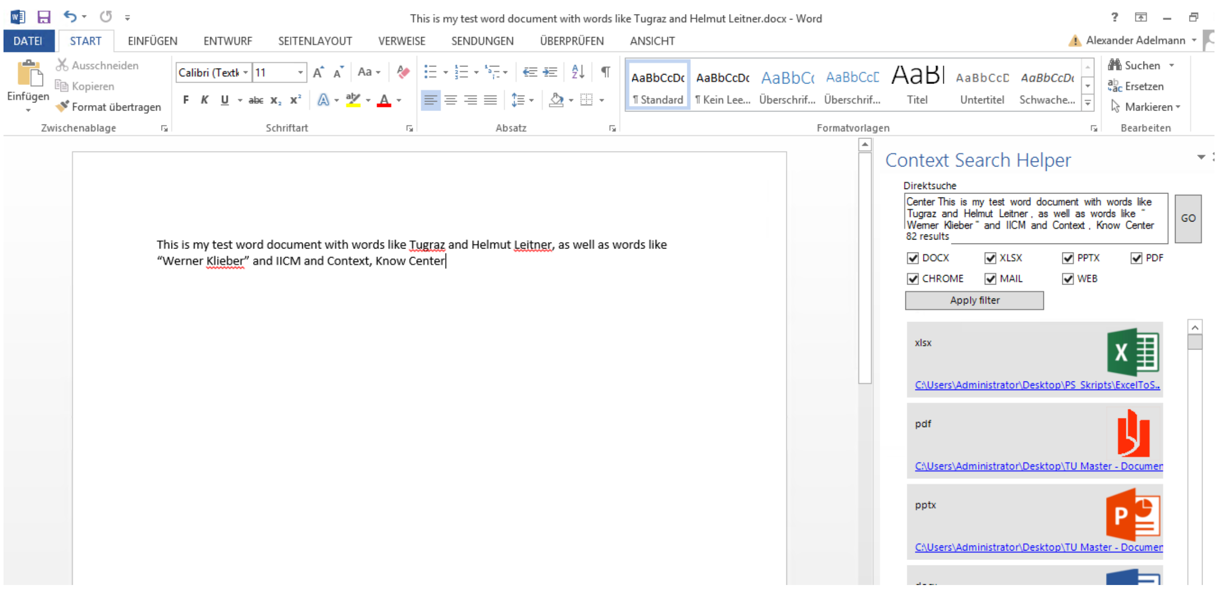
The architecture of the application is, as already mentioned, very modular and the Visualization module or block is not directly connected to the core application. This makes it possible to build client applications which only show the data collected by the system.

Possible client applications could be standalone Windows Forms [nn] or WPF [oo] applications. This application would only need to use some database technology like ADO.Net or the Entity Framework [pp] to read the data from the MSSQL database tables and display them. As the information in the database tables is always up to date each time a client application queries the table the returned information represents the current context and state of the system.

Another possibility for a client application for visualization could be some kind of webpage that displays and all users in the company could perform searches on the data to get to know on which topics the colleagues are currently working on.

2.4.2 The Microsoft Word 2013 AddIn

The client application which was been developed and included in the first version of the Context Search Helper System is a Microsoft Word 2013 AddIn. The AddIn consists of a sidebar which is visible by default. It can also be hidden and only shown when needed, but at the moment this has to be done by the user. In future releases it is possible to integrate a logic that hides and displays the sidebar automatically depending on the amount or quality of the search results. A simple approach would be to hide the Sidebar whenever less than N relevant elements have been found with a score higher than M. Of course this approach is a very simple one and can be improved.



This is my test word document with words like Tugraz and Helmut Leitner, as well as words like "Werner Klieber" and IICM and Context, Know Center

Center This is my test word document with words like Tugraz and Helmut Leitner , as well as words like " Werner Klieber " and IICM and Context , Know Center
82 results

C:\Users\Administrator\Desktop\PS_Skripts\ExcelToS..

C:\Users\Administrator\Desktop\TU Master - Documen

C:\Users\Administrator\Desktop\TU Master - Documen

Figure 13: Microsoft Word 2013 with the visible Sidebar of the Context Search Helper AddIn and results.

The Sidebar is currently showing also debug information as the application is still in a development state. The debug information shown in the sidebar is the textarea at the top. It shows which text snippet is used from the current document for analysis.

It would be too much information to take all the text of the opened document and analyze it. For example, the user has opened a very large document with a lot of text. Not all paragraphs would have the same topic, so taking all the text would result in a mix of multiple topics. This would lead to poor context recognition because there are too many different topics.

Therefore, the Word AddIn checks where the cursor is positioned and only takes the current paragraph. It is considered the current paragraph if the cursor is located in it. The idea behind this is that the current paragraph is the most important one, because the user is currently working on this part of the document. This is not the only information that is stored in the SQL database queue. A timer is running in the background of the AddIn and every 3 seconds the difference between a collection of words 3 seconds ago and the current word collection is taken as additional context input. Again this approach is considered helpful in the context detection as the last few words or the last sentence is potentially the most important one to the user in the whole document. So each time a search is triggered by the Word AddIn, the current “difference words” and the current paragraph are used for context recognition. This leads to an even stronger weight on the current “difference words” as these phrases occur twice in the search phrase.

During implementation of the sidebar it became obvious that it is not a good idea to always perform a context update and all the calculations. There are two reasons for this. The first problem with continuously calculation the phrases and writing to the database is that this would cost too much performance. Even on the development environment which has a very fast processor with a lot of cores and lots of RAM this lead to freezes in the GUI of Microsoft Word. The fact that the sidebar is running in an own background thread, does not change much about it.

The second reason is that always updating the sidebar results would cause a distraction and instead of helping the user, it would prevent the user to focus and concentrate on the document he is writing. That would lead to the complete opposite scenario of what the Context Search Helper System is meant for. As a result, the users would not use the sidebar and probably close it, so there would be no sense in using the system at all.

The User Experience (UX) design is meant to assist the user. It should not distract the user from his work. So the sidebar has an additional logic implemented that tries to figure out when the best moment for an update of the search results is. Therefore, another timer is running in the background. This timer is not running by default. It is started each time the user stops typing and does not continue for a few seconds. The timeout is set to five seconds. To achieve such a behavior this timerjob is started by the first timerjob, which is also used to calculate the difference between the words from three seconds ago and the current words of the document. This first timer, called the “WordDiffTimer” knows exactly when the user stopped typing. This is the case if the word difference is zero. Then the second timer is started. As mentioned above the timeout is set to five seconds. When the timer tick function is executed an update of the GUI is triggered. Then the timer is stopped again. Then it is check if the user started typing again. Only then the state of the sidebar is set to the writing mode again. If the user did not start writing again it would not make sense to trigger an update every 5 seconds as this would again disturb the user in his workflow and would lead to a bad user acceptance.

2.4.2.1 Microsoft Word sidebar I/O

This section focuses on how the sidebar sends its data to the Context Search Helper core system and how the results are displayed.

The overall flow is best described using a figure.

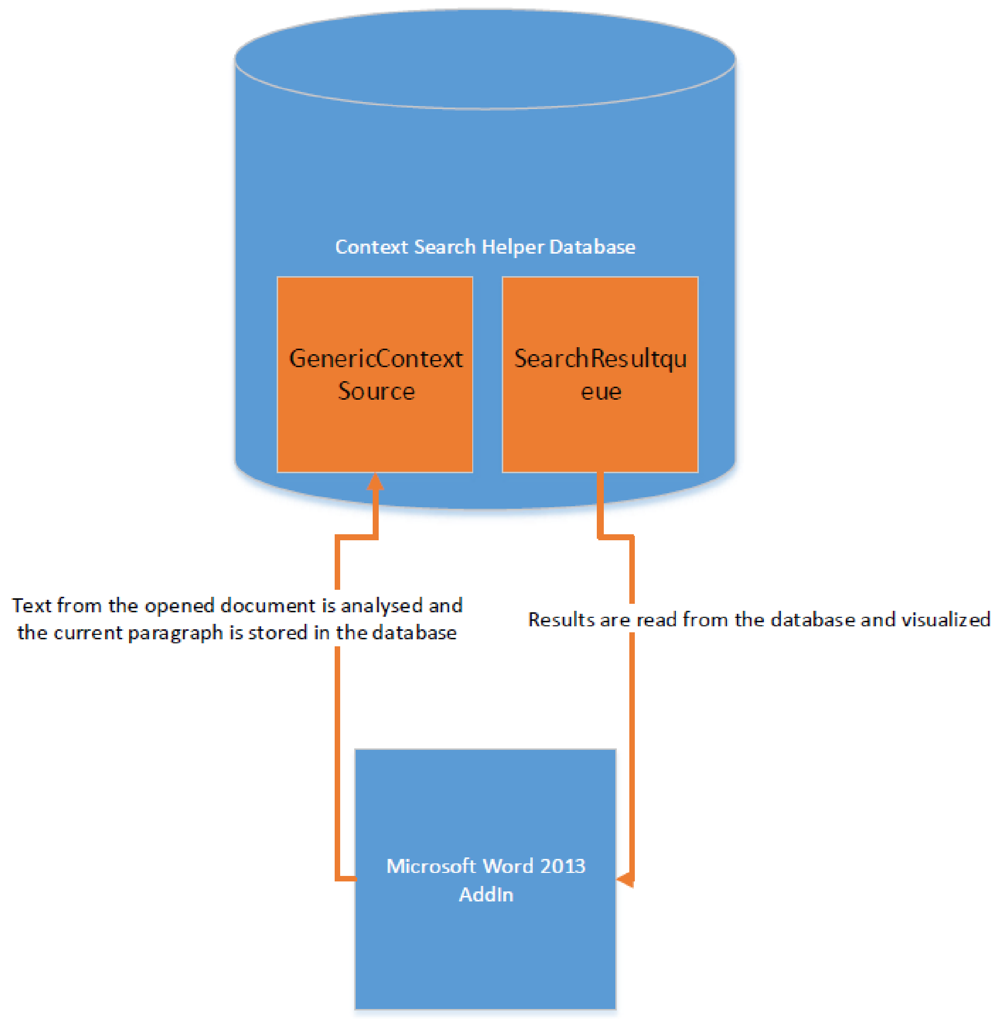

Context Search Helper Database
GenericContext Source
SearchResultqu eue
Text from the opened document is analysed and the current paragraph is stored in the database
Results are read from the database and visualized
Microsoft Word 2013 AddIn

Figure 14: The data flow between the Visualization module and the Context Search Helper Core.

As shown in figure 14 the communication of the Word AddIn and the Context Search Helper Core is two-way. The current paragraph and the so called “difference words” are stored in the database. The data is then read by the Context Search Helper Core module and processed. The process contains the communication with the Sensium web service and the search for the keyphrases in various data sources. Then the result items of the search are merged and stored in the database. Every time the Word AddIn notices that the user is not writing for more than five seconds the data in the database is read and cleared from the database table. The gathered result items are then displayed to the user.

The database table is cleared so the next time the sidebar reads the data, only the items which represent the current, new state of the system are stored in there.

If more than one client application would consume the data in the database queue at the same time, there could be problems with race conditions or deadlocks in the clients. As of the current version of the Context Search Helper there is only one client application which reads the data from the SearchResultQueue SQL table, there is no need to consider concurrency problems. In future releases when multiple clients access the data, the simple approach of just deleting the data in the result queue will not be a good idea. It would be possible to mark the entries with the client application's name and if all clients read the data, a delete operation would be triggered.

2.4.2.2 Icons and result Types

As of the moment the application supports 8 result types. A complete list of the result types which are supported is shown in the table below.

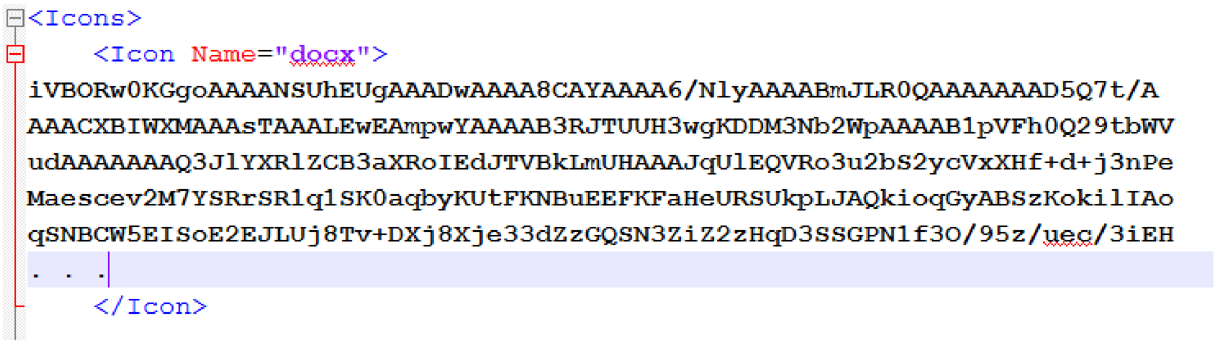
TYPE	DESCRIPTION
DOCX	This type represents Microsoft Word documents of the versions 2007, 2010, 2013, 2016
XLSX	This type represents Microsoft Excel sheets of the versions 2007, 2010, 2013, 2016
PPTX	This type represents Microsoft PowerPoint presentations of the versions 2007, 2010, 2013, 2016
MAIL	This result type is used for Emails found in the configured mail accounts in Microsoft Outlook of the versions 2007, 2010, 2013, 2016
CHROME	Results with this type have been found in the Google Chrome browser history of the user
PDF	PDF documents are represented using this type
TXT	Simple text Files use this type
WEB	All web search results are using this type at the moment.

Please note that this is the same table as shown in the "

2.3.2 The Plugin Interface" section. Of course the result types which are passed between the core module and the visualization module of the system have to support the same result types. This is part of the interface and it is very important for both parts to implement it correctly as the modules have absolutely no information about each other. This is one of the disadvantages of such a loosely coupled architecture. There is no type checking or other programming language features in the interface as everything is stored inside a database a strings.

To understand how the visualization is done by the sidebar it is easier to only focus on one result type. So as an example the docx type is analyzed and discussed. The sidebar gets all the existing types alongside with an icon from a XML based file which is stored on the file system. In the current version of the application the location of the file has to be the directory in which the main application is running. In future releases it is planned to be configurable easily in a fancy graphical dialog.

The structure of the file is as follows:



```
<Icons>
    <Icon Name="docx">
iVBORw0KGgoAAAANSUhEUgAAADwAAAA8CAYAAAA6/NlyAAAABmJLR0QAAAAAAAD5Q7t/A
AAACXBIWXMAAAsTAAALEwEAmpwYAAAAB3RJTUUH3wgKDDM3Nb2WpAAAAB1pVFh0Q29tbWV
udAAAAAAAQ3JlYXRlZCB3aXRoIEdJTVBkLmUHAAAJqUlEQVRo3u2bS2ycVxXHf+d+j3nPe
Maescev2M7YSRrSR1q1SK0aqbyKUtFKNBuEEFKFaHeURSUkpLJAQkioqGyABSzKokilIAo
qSNBCW5EISoE2EJLUj8Tv+DXj8Xje33dZzGQSN3ZiZ2zHqD3SSGPN1f3O/95z/uec/3iEH
. . .
    </Icon>
```

Listing 8. The structure of the icon and type file used by the Word sidebar for search results.

In the “Icons” node there can be a variable number of subnodes which have to be named “Icon”. These “Icon” nodes are the ones that represent a search result and are parsed from the Word AddIn. So if for example the Word AddIn reads a docx result from the database, it then checks its type and icon file if a “Icon” node with that name exists. Therefore, it is crucial that the values of the “Name” property of the “Icon” nodes match exactly the values written to the database from the Context Search Helper core. Again there is no typechecking so both parties have to strictly implement the interface.

Once an “Icon” tag with the matching name property was found, the icon is parsed. The icon images are stores ad Base64 [qq] strings in the configuration file.

When the matching result type was found in the file and the icon has successfully been parsed, a custom usercontrol object is created by the sidebar. This usercontrol is the actual result the users see in the graphical user interface of the side bar. The usercontrol has an icon a type and a `Linklabel` placed on it. The `Linklabel` is the control which makes it possible to actually open the found search result. The implementation of this is described in the next section.

2.4.2.3 Opening a result item

The Word Sidebar supports a very comfortable way to take a detailed look on the result items. It is possible to just double click on a result view and the document or email or chrome history website and so forth will be opened up in a new window.

New window means that for example double clicking a word result will open up a new instance of the word application and show the file. For this behavior to be achieved the Word AddIn needs to have to different functions implemented in code. This is needed because not all result types can be opened the same. Later in this section it will be explained why this is the case.

The function that is called whenever the user double clicks on a result entry or if the user clicks the Linklabel control at the bottom of each result entry is the following:

```
void lblLink_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
        if (lblType.Text.ToLower() != "mail")
        {
                try
                {
                        string toopen = e.Link.LinkData as string;
                        Process.Start(toopen);
                }

                catch (Exception ex)
                {
                        Utilities.Logger.LogError(ex);
                }
        }

        else //ismail
        {
                Outlook.Application myApp = Utilities.Helper.GetApplicationObject();
                Outlook.NameSpace mapiNameSpace = myApp.GetNamespace("MAPI");

                Outlook.MailItem getItem = (Outlook.MailItem)mapiNameSpace.GetItem
                FromID(lblHiddenValue.Text, null);
                 getItem.Display();
        }
}
```

Listing 9: The code function that is executed every time the user clicks the Linklabel on a result or the user double clicks somewhere in the result item. The function type is a standard Windows Forms event handler used to react to user input

The listing above shows how the opening of a result type is implemented. The first thing that can be observed is that there is an if-else construct used to differentiate between the result types.

2.4.2.3.1 Open result types

For nearly all result types the same code function can be called to show the result or open up the application and show the content of result. For these types the same logic can be used: docx, xlsx, pptx, chrome, pdf, txt and web.

The function is a built-in function of the .NET framework and is called "`Process.Start`". The function has several overwrites and the one used by the Context Search Helper System is the method with one parameter. The parameter passed to the function is a string which stores just a location to a file or a web address. For example, the Code "`Process.Start("c:\demo.txt");`" opens up a text file which is located at the C drive of the computer. The `Process.Start` function is part of the Reflection framework of the .NET framework. For a developer this function is very easy to use because the developer does not have to worry which application to open or anything. The `Process.Start` method communicates with the computer's operating system and gets the information of what the system wide default application for the given file type is. This very helpful function can be used to open up all result types except one: Mails.

2.4.2.3.2 Open mail results

As mentioned above it is not possible to open emails with the `Process.Start` function. This has a very simple explanation. An email is not a file type known to the system. Calling the `Process.Start` function would result in an error because the function would try to find a default application for the given type and as an email is no file type an exception would be thrown. An email is just an object inside an application. This application is Microsoft Outlook and only Outlook knows how to handle the email type.

In Outlook the emails have unique IDs and these IDs are used to identify and find emails in the application. Also the Context Search Helper Application uses the email IDs, these are stored in a hidden field of the result Usercontrol. If the result is not an email type the hidden field is simply not used, but if the result type is an email result, then the emails Outlook internal ID is stored.

In Listing 90, the else branch shows the code that is executed when the result type of the item which has been clicked by the user is an email. Then instance of Microsoft Outlook has to be used to search and open the email. The `Utilities.Helper.GetApplicationObject()` is used to get such an instance of Outlook.

Once the Outlook instance is ready, the email with the ID of the hidden field in the result entry is searched. Once the item has been found, the item is displayed.

For the user it makes to noticeable difference if the result item is an email or not, the result always opens up once he double-clicks it even if the code in the back and the logic is completely different. This was an important factor for the UX that the user sees no difference in the result types when he wants to open them and they can easily be shown by just double clicking the result entry.

2.4.2.4 Filter the results

The following screenshot shows the filter options of the side bar.

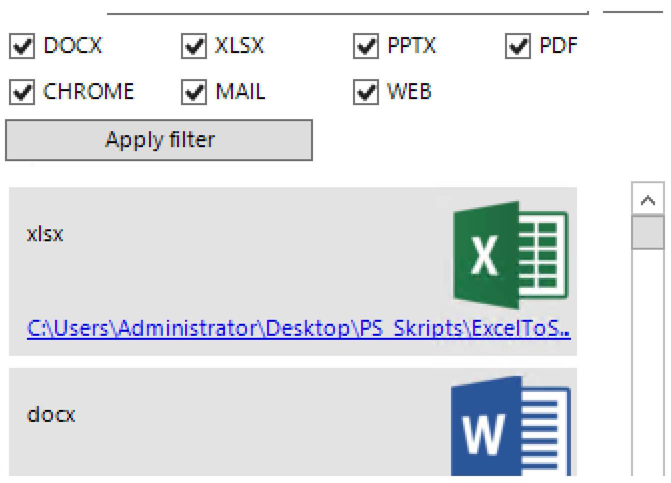


Figure 15: The filter abilities of the Word Sidebar

The sidebar gives the user the ability to filter the displayed search results. As there may be lots of results from various types like browser history entries alongside Microsoft Office documents, the sidebar is able to filter the displayed results. In the current version of the Word AddIn the filter possibilities are restricted to search result type filters. This means that the user can choose that he only wants to see Microsoft Word documents for example or documents from Microsoft PowerPoint or Excel. Maybe the user remembers that he had a conversation with a colleague in Microsoft Outlook and the emails are still in his inbox. Then it is possible to easily filter to only see emails.

In the background the filter does not change the code which gets executed or the logic which is used. The sidebar always queries all entries from the database and then applies the filter. This makes the filtering much faster. The results do not need to get fetched every time the user selects a filter. But this behavior also has its downsides. There reading of all the data takes longer that only getting the item which shall be displayed.

Because the user’s standard behavior is considered writing, then stopping to think about what next to write, and then going through the results displayed in the sidebar, the focus was on performance while changing the filters.

2.5 Database and Tables

The Context Search Helper system uses a few SQL tables stored inside a Microsoft SQL Server 2012 R2 Express. Express means that this server only supports a subset of all the functions the full version supports. Also smaller amounts of data can be stored and processed. These limitations do not affect the Context Search Helper application because only stores strings of texts and uses the SQL tables as queues. So there is not much data that is stored and therefore an MSSQL Server Express Edition was chosen.

The table below shows all the SQL tables used by the application as well as a description how they are used and what data is stored.

NAME	DATA	DESCRIPTION
SEARCHRESULTQUEUE	Queue for currently most relevant documents, emails,	For information about this SQL table please see section: 2.5.2 SearchResultqueue
SETTINGS	Application wide settings	This table is used to store application wide settings like the database connection string or the directories in which the Datasource and sensor modules look for the plugins. The structure of the table is exactly like a dictionary. The entries are stored with a key and a value as pair.
LOGENTRIES	All data mostly used for debug information.	This database table stores everything that goes on while the Context Search Helper application is running. All search results alongside with all the context phrases are stored in this SQL table together with a timestamp. With the information stored in this table it is easy to analyze the system sensors and Datasources.
GENERICCONTEXTSOURCE	Queue for external data	For information about this SQL table please see section: 2.5.1 GenericContextSource

2.5.1 GenericContextSource

This SQL table is used as a place where external applications can store data so that this data will get analyzed by the Context Search Helper System. As already mentioned in the section

2.1.3 The Sensor Module this table is used by the Generic Context Sensor.

The sensor is a very easy way to get information into the Context Search Helper system without the need to implement a new sensor or extend the application itself. An external application could be Yammer, Skype4Business or other applications which are used in a company and handle lots of valuable information.

Another external application which handles very important information for the Context Search Helper System is Microsoft Word. At first thought it might not appear as an external application for the system, as it is used to present the data gathered to the user. In fact, Word also uses the GenericContextSource SQL table to store data.

This design decision was made, because it gives the system a very modular architecture. In the first version of the Context Search Helper the Word AddIn and the core application were very tightly bound together so that a change in the Word AddIn required the whole application to be recompiled and restarted. With the modular design and the very loose binding between all the modules and parts, like the Word AddIn with the core component, there is no need to touch the core component when the Word plugin needs to be changed or updated. The Context Search Helper system does not even need to be restarted.

The table below shows the columns of the GenericContextSource.

COLUMN NAME	DESCRIPTION
PHRASES	In this column all the relevant phrases are stored. Each row represents the phrases at a concrete time.
DATETIMESENSED	The timestamp of the phrases

Each row in this SQL table is one snapshot of an external application. An improvement which is not implemented in the current version of the system is that the data table gets an additional column which stores the name or ID of the external application. Another column for weighting could be introduced. These two columns would make it possible to prioritize the external application. A very important application which handles more critical and important data than another could just get a higher weighting so the information would be more likely to influence the search results.

The GenericContextSource SQL table is used like a queue. This means that various external applications store their data in this table and every few seconds the Context Search Helper system collects all the information in the table and uses it alongside all the other context sensors' data for the calculation of the relevant key phrases. Every time the application reads the data stored in this table, the table gets cleared. This is the behavior of a queue.

The interval in which the data is read is every time the application’s main loop is in a new iteration. The flow of the application can be found in figure 10.

2.5.2 SearchResultqueue

The SearchResultqueue SQL table is also used as a queue. That means that the data that is written into it gets read periodically and then immediately gets deleted. This way the size of the table and the SQL server will not grow as the data in this table is deleted very often.

This data table is one of the most important ones of the Context Search Helper application as it stores the results of what the application is collection, analyzing and calculating in the background the whole time. This means the at every moment the data in this table represents the currently most interesting and important result sets which will help the user the most.

During implementation it became clear that it would result in a better overall architecture if the background system, the so called core, is not to closely coupled with the visualization module of the application. The reason for this is that modular structure of the application not only makes it much easier to maintain, but also there can be various and multiple clients.

So the visualization module is more like a standalone application which only reads one of the Context Search Helper’s database tables. And this is the real strength of the architecture. With the current version of the application a Microsoft Word 2013 AddIn is shipped which displays the context aware search results to the user. The Word AddIn is just reading the SearchResultqueue data table and shows the user the data in a user-friendly way.

It is very easy to add more clients to the application. Reusing the Word AddIn and displaying it in all Office Products is possible without any major changes to the code. It also does not require the rest of the application to be touched.

But there are even more possibilities than just an Office Plugin. If a company uses an own application, it could also act as client and read the SearchResultqueue SQL table. The content could be displayed on a website and other users could check on what a user is working on currently. Then they could contact the user and tell him that they once worked on the same topic and can help him for example.

The following table shows the columns that are used in this SQL table:

COLUMN NAME	DESCRIPTION
TYPE	This column holds information about the type of the context information. As this column is a single line of text type, there is no validation of the type. It is left to the client application displaying the data if it wants to use the information or not.
ADDITIONALINFORMATION	This column was introduced for more complex results like emails. As there is no possibility to open emails as easy as word documents for example in a client, this field was introduced to hold addition information about the result. In the current implementation of the application the email ID is stored in this column
ICON	The column is an optional column. It is meant to add a specific icon to a search results. As already mentioned, the client application which visualizes the data can also ignore this column.
ENTRYID	The ID of a search entry is stored in the column.
SCORE	The relevance or score of the stored result is written into this column. The database or the column do not validate the data in this column in any kind.

2.6 Demo and development hardware

The following sections will discuss the hardware resources that are needed to run the Context Search Helper System and the hardware that was used to implement and test our system. As our system runs in the background and tries not to disturb the user that is currently working on the computer in any way, it is clear that a slow hardware could become a problem. The system that runs the Context Search Helper System, needs to have enough calculating power to analyze all actions and tasks of the user as well as search various data sources without slowing the system down and decreasing the click performance of the OS (Operation System) and the used applications. The following section will show how the system performs on various hardware configurations and what hardware is the lower limit to even run the system properly.

The development machine that was used for the implementation of the Context Search Helper System was a virtualized one. The virtualization software that was used was Hyper-V [jj], which is part of Microsoft Windows. As the host machine was running Microsoft Windows 10 Professional the Hyper-V software could be added.

As mentioned above the system that was used was on a virtualized machine so this section will focus on the host as well as the client machine as both parts are important for a fast and optimized system.

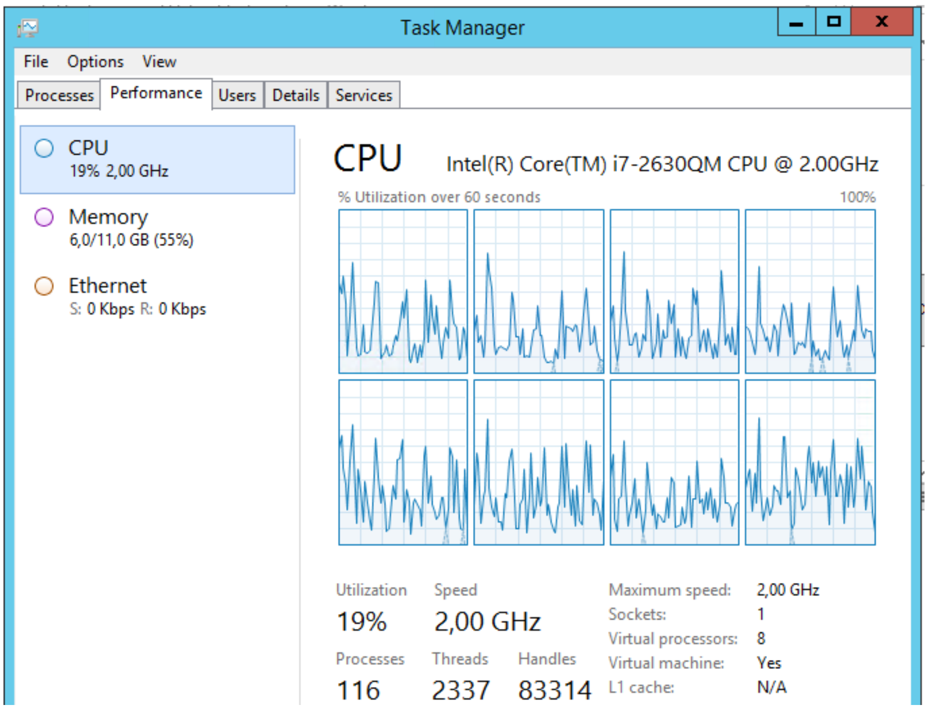
The host machine was a Packard Bell [w] Easynote TS11-HR notebook which has been upgraded for more performance. The system provides an Intel i7 2630qm [x] with 4 cores and 8 threads. As for virtualizing machines a lot of RAM is needed, the host system provides 16GB of RAM. The RAM is a DDR3 Notebook RAM with 1033 MHz frequency from Kingston [y]. Another important part for running virtualized systems is the hard drive. That part of a host computer can easily become a serious problem and be a bottleneck as a slow hard drive will decrease a system's performance drastically. The hard drive which was used for the development of the Context Search Helper System is of the SSD type. The brand and model of the used hard drive is a Crucial m550 [aa] with 250GB of Space. The hard drive was only used for the virtualized machine so the performance was better.

The client system, which was the system that was actually used for development was running a Microsoft Windows Server 2012 R2. As the Context Search Helper is doing a lot in the background and using multiple threads to run smoothly is was important that as many cores as possible are usable for the virtual machine. Therefore, all 8 threads of the host's CPU were also passed to the virtualized development system. To have enough RAM for an IDE, the system got 11GB of RAM. As mentioned above the Crucial m550 SSD was used as hard drive. The systems overall performance was very fast which was perfect for developing the first version of the Context Search Helper.

As integrated development environment [bb] Microsoft's Visual Studio 2015 Community was used. As the Context Search Helper System needs a kind of data storage in the background, a Microsoft SQL Server 2012 R2 Express was used.

2.7 Performance

As mentioned in the previous section the development system was fast and the Context Search Helper System was running with a very good performance.



CPU Intel(R) Core(TM) i7-2630QM CPU @ 2.00GHz

Utilization 19%
Speed 2,00 GHz
Processes 116
Threads 2337
Handles 83314

Maximum speed:	2,00 GHz
Sockets:	1
Virtual processors:	8
Virtual machine:	Yes
L1 cache:	N/A

Figure 16. This figure shows how much resources are used when the Context Search Helper System is running.

The figure above shows the average workload the Context Search Helper System needed the system to handle. It can be seen that the Context Search Helper makes use of various threads, as all the cores are used and there are no noticeable peaks at one specific core. The system is running with not optimized settings. This means that the system runs continuously and there is no logic used to optimize the performance. About 20% of the CPUs resources are used if the system runs all the time. Additionally, it must be said, that also the client application, which is a Microsoft Word 2013 Sidebar is running. On top of that both parts, the backend and the client application where running in a debugging mode and were started from Visual Studio. So there is some overhead for all the debugging code that runs in the background as well. On average the debugging functionalities added 5% of CPU activity. Also the RAM that is used is higher than running the Context Search Helper on a normal client computer as Visual Studio alone takes up to about 1GB when in debugging mode.

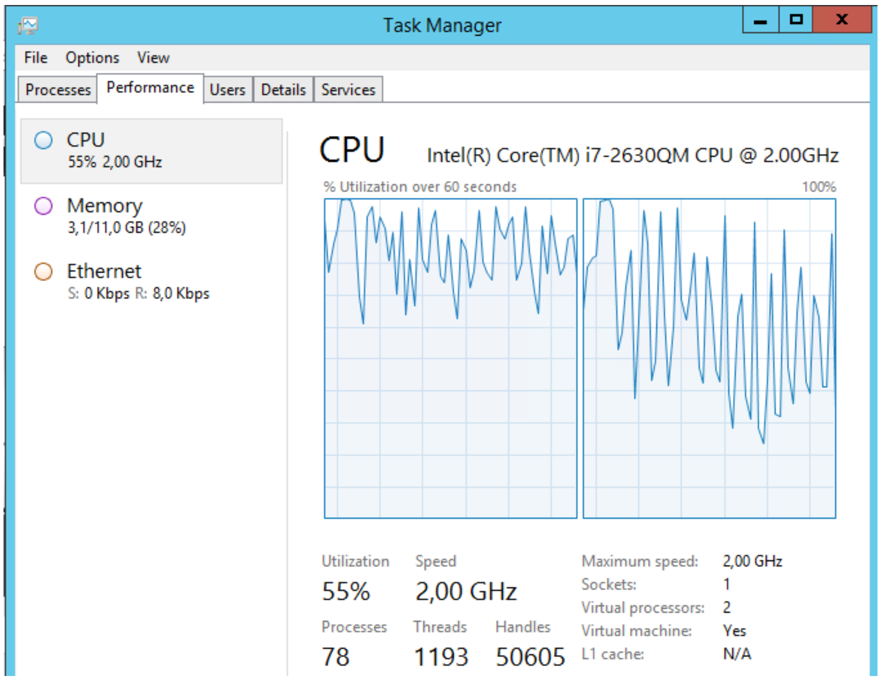


Figure 17. The hardware resources of the virtual machine running the application were limited.

Figure 17 shows how the Context Search Helper application performs on a much more limited environment. The CPU usage is much higher than with 8 cores, but the overall speed of the operating system was not lowered. Browsing the web and writing in Microsoft Word was easily possible without any delays. The overall click performance of the system was fast as well.

Nonetheless the two screenshots of the Windows task manager show that there is much optimization potential in the application. The fact that the CPU usage is very constant at a specific level, depending on the CPU cores available, shows that this is a good part for optimizations to reduce the static noise on the cores.

2.7.1 Optimization and Improvements

Please note that at the moment the following optimization possibilities are not implemented as the Context Search Helper system is still in a development state.

As mentioned in the paragraphs above the bottleneck of the system is the CPU. There are so many operations needed in the background that there is a constant level of workload for the CPU. This leads to the idea that idle times for the application would drastically decrease the usage of the CPU. It is not needed to the system to run permanently. For example, if the user is not working with the computer at the moment there is definitely no need for the system to run in the background.

The question is how to determine if the user is doing something constructive and is working and when is the user relaxing and does things like watching a video on YouTube [cc]. A very simple approach would be to check the mouse activity of the computer. If the mouse did not move for more than a few minutes the system could enter some kind of idle mode. Then a timer would check periodically if the mouse started moving again and then wake the system up from its idle state. This logic would work for a detection if the user is watching videos, but what about scenarios where the user is reading the newspaper or some other website. The user could simply be chatting with a friend over some messenger without any relation to his work.

A possibility to detect these scenarios would be that the Context Search Helper checks a list of known applications. For example, the list which would be configurable for an administrator or a power user would contain the following applications: Word, Excel, PowerPoint, Visio, Skype4Business, Outlook, Internet Explorer and OneNote. The system would check the running applications for the listed ones and if a defined number of the above applications are running then the system would start running in the background. The idea of it is that if the user has opened some of these business applications, the probability is very high that the user is working and not using the computer in his leisure time. If the application is running on a corporate computer and the company has its own applications which are used very often by the employees, an admin would simply roll out a different or updated version of the “running applications” file.

There are other parts of the Context Search Helper System which could be improved in the means of resource usage. A part which could be improved is the way the underlying database is used. The tables are used as a queue and therefore the content is deleted every time the data is read. This behavior makes sure that the data which gets read is always the newest and most important one. But also a lot of I/O operations need to be done. There could be mechanisms implemented which check the timestamp of the read operation with a column in the queue tables and only if the information in the table is older than X minutes the data is deleted. This would lead to less usage of the database. The same idea can be used for writing data to the database. The system could periodically check if the current changed more than a defined threshold and if not no information would be written to the database.

Another important part of the application in the means of optimization is the visualization module. At the moment this module is the most optimized one. There are already code blocks and logic implemented so that there are idle times to reduce the workload for the CPU. The client application checks if the user is working or not. Only if the user worked and then stopped for a defined timespan, the system triggers a database read action. It is then also checked if the user started working again. Only if the user stops working again for a defined timespan the system read the database tables again.

2.8 Error Handling

In every software errors occur. Even though the number of errors can be reduced by a test phase, not all error can be eliminated. But not every failure is equally severe and not every failure is a bug. Sometimes it is just needed to react to certain situations with Error Logging. Like the administrator did not configure a path to the Plugin location for the Sensor module or the Datasource module. More details about these two modules and their architecture can be found in the sections

2.1.3 The Sensor Module and

The Datasource Module. Not providing a path to Dynamic Link Libraries (dll) is not really an error of the system and it will run, but just without the functionality of the module, so it should be logged or the user should be informed in some way, but it does not stop the program from running.

The Error Logging module of the Context Search Helper system is very simple. It provides specific function for different types of logging message. There are functions to log information or errors. If an error shall be logged there are two ways to log it. Either just an error message will be handed to a function or an object of the type `System.Excpetion`.

The method which takes an exception object as input also reads the stack trace and inner exception if given of this object. In case of a severe error this information can help the most when trying to figure out what went wrong exactly. The functions are implemented as follows:

```
        public static void LogError(Exception ex)
        {
            using (EventLog eventLog = new EventLog("Application"))
            {
                string message = String.Empty;

                message += ex.Message;
              message += Environment.NewLine + "StackTrace: " + ex.StackTrace +
              Environment.NewLine;

                if (ex.InnerException != null)
                {
                    message += Environment.NewLine + "InnerException: " +
Environment.NewLine +
                                ex.InnerException.Message;
                    message += Environment.NewLine + "StackTrace InnerException: " +
                                ex.InnerException.StackTrace + Environment.NewLine;
                }


                eventLog.Source = "ContextSearchHelper";
                eventLog.WriteEntry(message, EventLogEntryType.Error, 234);


            }
        }
```

Listing 10. The listing shows the error handling functions of the Context Search Helper System.

The messages and information handed to the Error Logging module are then written to the Microsoft Windows Event Log and marked as an error. The windows event log is an application which can be opened by the user and shows all the error of all applications of the system including the Context Search Helper.

2.9 Hardware specifications for real world usage

As shown in the previous chapters, there is a lower limit for the hardware of the computer that runs the Context Search Helper. It was explained that two hardware components are the most important for a system to run our application. The CPU as well as the RAM are these two components. The computer that runs the Context Search Helper needs enough RAM to store all the data that is being process during execution of our system. Not enough RAM would lead to a highly reduced overall system performance as the OS (operating system) starts to use the hard disc to store data if the system runs out of RAM. As the performance of a HDD or SSD (solid state disk) is nowhere near the speed of RAM memory, the overall click performance of the computer would decrease a lot.

We suggest 2-3 gigabyte of additional RAM on a system that is meant to run the Context Search Helper smoothly.

The other important hardware component is the CPU. The previous sections showed that the performance of our system is directly proportional to the number of cores of the CPU. As the Context Search Helper is implemented using multiple threads, the .NET framework, which is the core of our system, can easily use multiple cores of a CPU. We showed that running the Context Search Helper system produces some kind of background noise on all cores of the CPU because the system is running all the time and analyzes the user and his tasks in real time. The more cores the CPU offers, the less noise per core is produced.

Our suggestion for a CPU that is strong enough to run our system is that it should have at least 4 cores. It would be sufficient if there are only 2 real cores and 4 threads like in a lot of Intel’s i3 processors.

To summarize the hardware specifications, the following table shows the needed components to run the Context Search Helper smoothly.

COMPONENT	MINMAL REQUIREMENT
CPU	Min. 4 CPU threads @ ~ 2 – 2.5 GHz
RAM	~8GB of Ram
OS	Windows 7, 8, 8.1, 10
HDD-SPACE	< 200 MB of free space

3. Future releases and improvements

The current version of the Context Search Helper may be a good piece of software that does help the user to get his work done easier and faster. Nevertheless, the project which was written for this thesis is more like a POC, which is a short term for Prove of Concept. That means that the Context Search Helper showed that it is possible to access all the sensors and collect all the data as well as search the various search data sources. Another important part was the client application which is implemented as a Microsoft Word 2013 AddIn. The AddIn showed that it is easily possible to visualize all the collected data in a useful way as well as the possibility to open all the result types as new applications.

Another idea for the Context Search Helper system and the thesis was to build a modular system that can be integrated in other, bigger projects. The loosely coupled and modular architecture of the Context Search Helper system makes it possible to take parts of it and use them in other projects where these parts are missing. For example, other projects that need to analyze the user's system can only use the sensor module of the context Search Helper. As the interfaces are weakly typed and simple it is easily possible to integrate whole modules in other software projects with nearly no effort.

Another important information regarding the Context Search Helper System is, the system is not finished or done. The project is still in development and will be enhanced and improved in future releases. Furthermore, the system is available at GitHub [dd] for users to download or improve. The system is open source so extending the system and improving it is wanted.

3.1 Improvements

There are a lot of parts of the software that can and should be improved. This section will discuss the open issues and parts which should be improved in future releases. Not all of the mentioned issues are bugs in the application, many of them are kind of UX or performance improvements. In the following paragraphs ISSUE stands for a possible improvement or missing functionality. FIX describes the actions that need to be done to resolve the ISSUE. LOCATION is the file or module where the FIX has to be implemented.

ISSUE:

Sensor types are hardcoded in the sensor classes.

FIX:

The sensor type is loaded from a configuration file or from the database. This would lead to a much cleaner and more robust code and application.

LOCATION:

Every Sensor Project as well as the Proxy that searches for the Plugins and uses them.

ISSUE:

The generic datasource has no application link. This means that there no possibility to know where the information in the table COMES from. For a weighting logic this would be important.

FIX:

The generic Datasource SQL table gets an additional column to store the application that inserted the data into the table.

LOCATION:

All client applications that make use of the generic Datasource. The DBWriter would make the column required and would return an error in case there is no information given.

ISSUE:

Sensors do not implement a weighting logic.

FIX:

The possibility to weight a sensor is implemented. This way one sensor could be weighted because it is more important for the user or the company than others.

LOCATION:

ContextSourceProxy can handle a weighting for all the plugins that are used.

ISSUE:

Graphical User Interface is not very optimized and not looking good. It breaks the overall look of Word.

FIX:

The Microsoft Word Sidebar is looking better and optically integrates better with Word 2013 and the Windows 8 / 10 look and feel.

LOCATION:

ContextSearchBar in the AddIn project.

ISSUE:

Some Datasources have an improvable ranking system. The Outlook source for example does not provide a ranking information at the moment.

FIX:

All Datasources use a good ranking logic for the results like the Windows Search Data source.

LOCATION:

All Datasource projects.

ISSUE:

The system makes heavy use of the CPU.

FIX:

All the possible improvements from section

Optimization and Improvements are implemented.

LOCATION:

A lot of files and classes need to be optimized.

ISSUE:

Support for multiple users.

FIX:

The system can handle multiple users in a windows network and gives recommendation for the users about what other users worked on and how they got the work done. Privacy is a possible issue here and needs to be handled with care.

LOCATION:

The architecture of the application is already ready for multiuser. The database would need to be improved to store the information of more than one user.

ISSUE:

The Word Sidebar does not show the reason why a result is displayed. The user does not have any information about the reason of the result.

FIX:

The system shows a snipped of text from the result so that the user knows why the result is displayed. This would lead to a much better User Experience.

LOCATION:

The Datasources need to support the feature in order to return the information why the result was found. Also the database tables need to be extended to store the additional information. The Word Sidebar needs to display the snippet.

4. Conclusion and Lookout

The context search helper system is a first implementation and prototype at the moment. There are plenty of improvements possible. The system does a good job in detecting the context of the user who is currently working on the machine running the Context Search Helper in the background.

The implementation and tests have shown that is a very hard and complex task to get the best matching data for a given context. Even after the current context extraction of the user the system has to deal with a huge amount search result. At the moment there is no weighting mechanism implemented in the data sources module. An implementation of such a logic could improve the search for even better matching documents.

In the section about the Aposdle project the differences to the Context Search Helper application have been shown. One important difference is the absence of a user model in the Context Search Helper approach. A user model is an approach which is very often used in work-integrated learning applications. The context information in our system is always up to date but has no possibility to save any details. That means, every few seconds all the information about the user's context is deleted and the new context is calculated.

A focus in future releases of our system should be an implementation of an adaptive user model and profile. It is likely that the user's context is often the same as he is working on the same kind of tasks. For example, a knowledge worker like a software engineer for .NET and C# development will very likely search for topics related to his domain, in this case .NET and C# programming. It is unlikely that the user searches for CAD applications as this is not his field of expertise. Therefore, the implementation of a user module will improve the context detection as well as the search for relevant documents and data. This feature will make use of a weighting logic as mentioned.

5. Literature and Links

This section shows all the references used in the thesis as well as links to MSDN articles for further reading or websites of products or companies that are mentioned in the thesis. The first subsection of this section shows all the literature that was used for writing this thesis. This includes mostly Papers or scientific documents but also demo- and PR documents from the Aposdle system. The second subsection lists links and websites for all the mentioned technical companies, products or technical blogs and MSDN articles.

5.1 Literature

1. APOSDLE-D02.7-UT-Conceptual-Architecture2_v02, Aposdle Project, 2009
2. APOSDLE-D04.11-KC-FinalSystemArchitectureP3_v1.15, Aposdle Project, 2009
3. Lindstaedt, S., Ley, T., Mayer, H.: Integrating Working and Learning in APOSDLE, Poster, 2005
4. Lindstaedt, S., Ley, T., Mayer, H.: Integrating Working and Learning in APOSDLE, 2005 Aposdle Website
5. Lindstaedt, S., Mayer, H.: A Storyboard of the APOSDLE Vision, Aposdle Website
6. Final-Activity-Report_v04, Aposdle Project, 2010
7. Brakeley, H., Cheese, P., & Clinton, D.: The High Performance Workforce Study 2004. Research Report. High Performance. Delivered, Accenture (2004)
8. Guarino, N. (1998). Formal Ontology and Information Systems. In: Proceedings of FOIS'98, Trento, Italy, IOS Press.
9. Karagiannis, D. & Telesko, R. (2000). The EU-Project PROMOTE: A Process-Oriented Approach for Knowledge Management. In: Proceedings of PAKM 2000 - Conference on Practical Aspects of Knowledge Management.
10. Ley, T. & Albert, D. (2003). Identifying Employee Competencies in Dynamic Work Domains: Methodological Considerations and a Case Study. Journal of Universal Computer Science, 9(12), 1500-151
11. Garrick, J. (1997). Informal learning in the workplace. London: Routledge
12. Berry, D. C. (1997). How implicit is implicit learning? Oxford: Oxford University Press
13. O'Donnell, A.M. (1999). Structuring dyadic interaction through scripted cooperation. In A.M. O'Donnell, A. King (Eds.), Cognitive perspectives on peer learning. Mahwah, NJ: Erlbaum.
14. Dansereau, D.F., & Johnson, D.W.(1994). Cooperative learning. (Chapter 5). In D. Druckman, R. A. Bjork (Eds.), Learning, remembering, believing: Enhancing human performance. Washington, DC: National Academy Press.
15. Lindstaedt, S.N., Ley, T., Mayer, H.: Integrating Working and Learning in APOSDLE. In:
16. Proceedings of the 11th Business Meeting of the Forum Neue Medien, November 10-11, University of Vienna, Austria (2005)

17. Getting to Know Your User – Unobtrusive User Model Maintenance within Work-Integrated Learning Environments
18. Eraut, M., Hirsh, W.: The Significance of Workplace Learning for Individuals, Groups and Organisations. SKOPE, Oxford & Cardiff Universities (2007)
19. Benyon, D.R., Murray, D.M.: Adaptive systems; from intelligent tutoring to autonomous agents. Knowledge-Based Systems 6(4), 197–219 (1993)
20. Goecks, J., Shavlik, J.: Learning users' interests by unobtrusively observing their normal behavior. In: IUI 2000: International Conference on Intelligent User Interfaces, pp. 129–132 (2000)
21. Jameson, A.: Adaptive interfaces and agents. In: Jacko, J.A., Sears, A. (eds.) Humancomputer
interaction handbook, pp. 305–330. Erlbaum, Mahwah (2003)
22. Lindstaedt, S.N., Ley, T., Scheir, P., Ulbrich, A.: Applying Scruffy Methods to Enable Work-integrated Learning. Upgrade: The European Journal of the Informatics Professional
9(3), 44–50 (2008)
23. Exploiting Context Information for Identification of Relevant Experts in Collaborative Workplace-Embedded E-Learning Environments
24. Lave, J., Wenger, E.: Situated Learning: Legitimate Peripheral Participation. Cambridge University Press, Cambridge (1991)
25. van Welie, M., van der Veer, G.C., Eli¨ens, A.: An Ontology for Task World Models. In: Markopoulos, P., Johnson, P. (eds.) Design, Specification and Verification of Interactive Systems '98, Wien, pp. 57–70. Springer, Heidelberg (1998)
26. Object Management Group. OMG BPMN Final Adopted Specification (2006), http://www.omg.org/docs/dtc/06-02-01.pdf
27. Bauer, T., Leake, D.: A Research Agent Architecture for Real Time Data Collection and Analysis. In: Proceedings of the Workshop on Infrastructure for Agents, MAS, and Scalable MAS (2001)
28. Birnbaum, L., Hopp, W.J., Iravani, S., Livingston, K., Shou, B., Tirpak, T.: Task aware information access for diagnosis of manufacturing problems. In: IUI, pp. 308–310 (2005)
29. Mcdonald D.W.: Supporting nuance in groupware design: moving from naturalistic expertise location to expertise recommendation. PhD thesis, Chair-Mark S. Ackerman (2000)
30. Han, J., Kamber, M.: Data Mining. Concepts and Techniques. Morgan Kaufmann Publishers, San Francisco (2001)
31. Joachims, T.: Text Categorization with Support Vector Machines: Learning with Many Relevant Features. Proceedings of the 10th European Conference on Machine Learning, pp. 137–142 (1998)
32. Platt, J.: Fast Training of Support Vector Machines using Sequential Minimal Optimization. MIT Press, Cambridge (1998)
33. Quinlan, J.R.: Learning decision tree classifiers. ACM Computing Surveys (CSUR) 28(1), 71–72 (1996)
34. Clark, P., Boswell, R.: Rule Induction with CN2: Some Recent Improvements. Proceedings of the Fifth European Working Session on Learning 482, 151–163

(1991)

36. Fürnkranz, J., Widmer, G.: Incremental Reduced Error Pruning. In: Proceedings

the Eleventh International Conference on Machine Learning, New Brunswick, NJ, pp. 70–77 (1994)

5.2 Links

a. https://www.microsoft.com/en-us/windows/10
b. https://www.office.com
c. https://yammer.com
d. https://www.skype.com/en/business/skype-for-business/
e. https://www.youtube.com/watch?v=4ToXuOTKfAU – Video of Aposdle's flow
f. http://www.aposdle.tugraz.at/
g. http://europa.eu
h. http://www.know-center.tugraz.at
i. https://www.sensium.io
j. https://support.microsoft.com/en-us/kb/176792 - How to connect to a running instance of Internet Explorer
k. https://msdn.microsoft.com/en-us/library/bb508515(v=vs.85).aspx - MSHTML
l. https://msdn.microsoft.com/sv-se/library/ff521735(v=vs.85).aspx – A list of all SearchIndex columns in the System namespace
m. https://msdn.microsoft.com/en-us/library/windows/desktop/bb231278(v=vs.85).aspx – The Rank By clause
n. https://msdn.microsoft.com/en-us/library/ms526936(v=exchg.10).aspx - urn:schemas:httpmail: Namespace
o. http://gs.statcounter.com/#desktop-browser-ww-monthly-201512-201601-bar
p. https://www.google.at/chrome/browser/desktop/index.html
q. http://sqlite.org/
r. https://saas.know-center.tugraz.at/news-demo/#/knowminerSearch
s. https://msdn.microsoft.com/en-us/library/system.net.webclient(v=vs.110).aspx – The WebClient class
t. http://www.newtonsoft.com/json
u. https://msdn.microsoft.com/en-us/library/system.diagnostics.process.start(v=vs.110).aspx – The Process.Start Method
v. https://msdn.microsoft.com/en-us/library/office/ff462097.aspx - How to: Get and Log On to an Instance of Outlook
w. http://www.packardbell.at/pb/de/AT/content/home
x. http://ark.intel.com/de/products/52219/Intel-Core-i7-2630QM-Processor-6M-Cache-up-to-2_90-GHz
y. http://www.kingston.com/de/
z. https://de.wikipedia.org/wiki/Solid-State-Drive
aa. http://eu.crucial.com/eur/en/
bb. https://de.wikipedia.org/wiki/Integrierte_Entwicklungsumgebung
cc. http://www.youtube.com
dd. https://github.com/
ee. https://en.wikipedia.org/wiki/Service-oriented_architecture
ff. http://www.opengroup.org/

gg. https://msdn.microsoft.com/en-us/library/windows/desktop/bb231256(v=vs.85).aspx – Querying the Index with Windows Search SQL Syntax
hh. https://msdn.microsoft.com/en-us/library/ms731082(v=vs.110).aspx – WCF Services
ii. https://en.wikipedia.org/wiki/Petri_net
jj. https://en.wikipedia.org/wiki/Hyper-V
kk. https://www.google.com
ll. https://www.wikipedia.org/
mm. http://www.bing.com/
nn. https://en.wikipedia.org/wiki/Windows_Forms
oo. http://wpf.codeplex.com/
pp. https://msdn.microsoft.com/en-us/data/ef.aspx - The Entity Framework
qq. https://en.wikipedia.org/wiki/Base64
rr. https://employid.eu/
ss. http://www.prolixproject.org/