

D3.5 Method integration

Marc Ehrig

Peter Haase

Steffen Staab

Christoph Tempich (University of Karlsruhe)

Executive Summary.

SWAP EU IST-2001-34103 Project Deliverable D3.5 (WP3 Methods)

This deliverable is about the integration of the different methods within the single components as well as the aspects of integration between methods of different components. In the previous deliverables of this work package, different methods were developed and their objectives defined. Furthermore, test scenarios were elaborated to compare methods with the same objectives. This deliverable emphasizes the integration of different methods independently from the actual implementation. Hence, the results found here will also be applicable for the refined methods described in the previous deliverable in this work package. In contrast to deliverable 5.2, which describes the integration of the different tools, the focus here is on the data model and data representation.

Document Id. SWAP/2003/D3.5/v0.3
Project SWAP EU IST-2001-34103
Date July 31th, 2003
Distribution Public

SWAP Consortium

This document is part of a research project partially funded by the IST Programme of the Commission of the European Communities as project number IST-2001-34103. The partners in this project are: Institute AIFB / University of Karlsruhe (coordinator, Germany), Vrije Universiteit Amsterdam VUA (Netherlands), Meta4 (Spain), empolis UK Ltd. (UK), empolis Polska (Poland), Dresdner Bank AG (Germany), and IBIT (Spain).

University of Karlsruhe

Institute AIFB
Englerstr. 28
D-76128 Karlsruhe
Germany
Tel: +49 721 608 3923, Fax: +49 721 608 6580
Contactperson: Steffen Staab
E-mail: staab@aifb.uni-karlsruhe.de

Meta4 Spain S.A.

Rozabella, 8 Centro Europa Empresarial
28230 Las Rozas (Madrid)
Spain
Tel: +34 91 634 85 00, Fax: +34 91 634 86 86
Contactperson: Adelma Stolbun
E-mail: adelmas@meta4.com

empolis Polska Sp. z o. o.

Plocka 5a
01-231 Warsaw
Poland
Tel: +48 22 535 88 06, Fax: +48 22 535 88 14
Contactperson: Mariusz Olko
E-mail: mariusz.olko@empolis.pl

Fundación IBIT

Reverend Francesc Sitjar 1
07010 Palma de Mallorca
Spain
Tel: +34 971 177 271, Fax: +34 971 177 279
Contactperson: Esteve Lladó Martí
E-mail: esteve@ibit.org

Vrije Universiteit Amsterdam (VUA)

Division of Mathematics and Informatics W&I
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
Tel: +31 20 4447786, Fax: +31 20 4447653
Contactperson: Hans Akkermans
E-mail: hansakkermans@cs.vu.nl

empolis UK Ltd.

Unit B, The Dorcan Complex, Faraday Road
SN3 5HQ Swindon
United Kingdom
Tel: +44 77 99694621, Fax: +44 17 93485451
Contactperson: Graham Moore
E-mail: gdm@empolis.co.uk

Dresdner Bank AG

Jürgen-Ponto-Platz 1
60301 Frankfurt am Main
Germany
Tel: +49 69 263 58265, Fax: +49 69 263 81385
Contactperson: Martin Lorenz
E-mail: martin.lorenz@dresdner-bank.com

Changes

Version	Date	Author	Changes
0.1	13.5.03	cte	first outline of the document
0.2	09.7.03	cte	addition of PMML description
0.25	21.7.03	meh	inclusion merger
0.28	22.7.03	pha	inclusion informer
0.3	21.7.03	cte	revised version
0.4	22.7.03	cte	correction version
1.0	31.07.03	cte	final version

Contents

1	Introduction	3
2	Local Node Repository	5
2.1	Storage	5
2.1.1	Summary of features	5
2.2	Access to the Local Node Repository	8
3	Knowledge Source Integrator	9
3.1	Extractor	9
3.1.1	Syntactic extractor	9
3.1.2	Semantic extractor	10
3.2	Ontology Integrator	11
3.2.1	Summary of methods	12
3.2.2	Integration of different integration methods	12
3.2.3	Integration with other methods	12
3.3	Metadata Integrator	13
3.3.1	Annotation	13
3.3.2	Rating	13
3.4	Merger	14
3.4.1	Integration with other methods	14
3.4.2	Integration with other components	14
4	Informer	16
4.1	Advertiser	16
4.1.1	Summary of Methods	16
4.1.2	Integration with other methods	17
4.2	Discoverer	18
4.2.1	Summary of Methods	18
4.2.2	Integration with other Methods	18
5	User Interface	19
5.1	Integration of OntoEdit	19
5.2	Integration with other components	20

<i>CONTENTS</i>	2
6 Replier	21
6.1 Peer selector	21
6.2 Query rewriting	22
7 Controller	23
7.1 Integration of the different methods	23
7.1.1 Query	23
7.1.2 Advertisement	24
7.1.3 Discovery	25
8 PMML - Predictive markup modelling language	26
8.1 Overview	26
8.2 Language Primitives	27
8.2.1 Data Flow	27
8.2.2 Header	27
8.2.3 Data dictionary	28
8.2.4 Mining Schema	28
8.2.5 Transformations	28
8.2.6 Statistics	29
8.2.7 Taxonomy	29
8.2.8 Models	29
8.3 Tools	32
8.4 Integration	32
8.4.1 Knowledge Integrator	33
8.4.2 Informer	33
9 Conclusion	34

Chapter 1

Introduction

Integration is the process of combining different methods in a way that they enhance each other and the user can take full advantage of them.

Deliverable 3.1 surveys different existing methods which were promising for an application within SWAP. The following deliverable 3.2 examines the method design. In deliverable 3.3 the implementation of the different methods is described, while deliverable 3.4 focuses on the testing of the implemented methods. This deliverable will emphasise on the integration of the developed methods.

In each chapter a component is examined by firstly summarizing the particular functionalities, *viz.* considering the methods, secondly investigating the integration of methods within the component and finally evaluating the dependencies between methods of different components.

The first part of the deliverable examines the role of the local node repository and the two meta models defined by SWAP. The different ways the components use the SWAP metadata model and access the repository are specified.

The knowledge source integrator is considered in the following chapter. The knowledge source integrator comprises the extractor, selector, annotator and merger. Each of the methods provides functionalities which are of importance for other components.

The methods of the informer depend on the results of the knowledge source integrator in the same way as vice versa. The knowledge source integrator externalizes the knowledge a peer has, while the informer publishes that knowledge and seeks for other peers of interest.

The user interface is integrated with all other components, since many automatic decisions of the system, might be subject to user intervention.

The replier depends also on the information the other components provide, because it selects peers and rewrites queries on the basis of information the other components have generated.

Finally, the controller as a central component which controls the data flow is examined. The task of the controller is to organize the different components in a way that the above described interactions can take place.

In addition to the components we will introduce the predictive markup modelling language (PMML). PMML defines a model for many aspects of input and output data for machine learning applications in XML. Thus, it provides a general semantic to exchange information about learning models. Using PMML, peers can exchange information about the characteristics they have learned from applying machine learning techniques.

This deliverable will be summarized in the conclusion.

Chapter 2

Local Node Repository

The Local Node Repository provides storing and querying facilities to all other components. Thus, it plays a central role within the SWAP system. RDF(S) was chosen as representation language, hence the repository allows for storing RDF statements. The stored information can either be retrieved via the API or the Query language SeRQL. This chapter will briefly summarize the functionalities of the local node repository, viz. storing and querying, and then emphasize on the integration aspects it provides for the other components. Furthermore the role of the SWAP metadata model for integrating the different methods is examined.

2.1 Storage

The Local Node Repository is used to store the knowledge the peer can provide to the network as well as its metadata. Integration will be based on the use of the common knowledge representation language RDF(S) as well as the use of the SWAP metadata model.

2.1.1 Summary of features

The local node repository provides means to add and delete statements from the repository. The evolving RDF(S) Graph can be queried with SeRQL. Inferencing is done according to the RDF(S) model theory.

Two methods are implemented for the local node repository. One is based on a database and allows for persistent storing. The other one stores information in the main memory.

Knowledge representation language RDF(S)

The fundament of all integration between the different components is our knowledge representation language. We use RDF(S)¹ to represent the external knowledge each peer has, to describe ontologies and to store information about peers. Furthermore, RDF(S) is used to describe the confidence and trust a peer has in assertions and other peers.

RDF(S) offers different concepts to describe a semantic model. We will briefly introduce the main concepts and describe their use in the SWAP system.

rdf:Statement Statements are the way to express anything in RDF. A statement is build from a subject, predicate and object. Due to the heterogeneity of statements which occur in a peer-to-peer environment many statements have to be reified and assigned a confidence rating. This stored in the swabbi-object.

rdf:Property With properties the characteristics of a class are being modelled.

rdf:type In RDF(S) no explicit construct for instances exists. To assert that something is of a certain type, one uses the “rdf:type” construct.

rdfs:Class A class is the most abstract construct in RDFS. A concept within an ontology has “rdf:type” class. We have defined several classes for the SWAP - system. The swabbi-object is a class as well as all the concepts within the application ontology. In case we can transform e.g. a folder type into a concept we add a type class statement to the repository.

rdfs:Resource Resources are identified by a unique URI. All resources which are added to the local node repository are annotated with a swabbi-object to identify the origin and other meta data.

Other There are many more constructs within RDFS which are used for SWAP. However, we use them with the same semantics as described by the W3C.

OWL Some of the methods within SWAP have the need for richer semantics than are offered by RDFS. For example the Merger analyses the extracted structures and tries to find overlaps between the existing ontology and a new ontology. In RDFS there is no possibility to express the equality of two objects. Furthermore, RDFS does not offer an explicit distinction between concepts and instances. Therefore, we are currently evaluating OWL² to represent higher semantical models. In particular we need the

¹<http://www.w3.org/RDF/>

²<http://www.w3.org/TR/owl-ref/>

`owl:equivalentClass`; `owl:sameAs` properties to express equivalence between concepts and instances.

Application ontologies

One of the many requirements which derive from the case studies was, that the SWAP peer-to-peer environment should be integrated seamlessly into the users current work process. From a technical perspective it means, that the information which is stored in many different applications should be integrated into the system. This implies, that the essential information from the different applications must be extracted and transformed into our representation language. It is then possible to integrate it into the repository. Therefore, we have built application ontologies for some applications. These ontologies contain the main concepts of the different applications. Since many applications and their data structures are optimized for one particular task, the implicit semantics of the different concepts is not evident. E.g. a “folder” in the filesystem has no semantics attached to it, except containing files and other folders. However, most users group documents of similar content into the same “folder” and label it accordingly. Hence, we can use the implicit semantics within each application and externalize it.

The application ontologies build the bridge between the different applications and the semantic layer we want to provide to the user. On the one hand they are used to retrieve information from the original sources. On the other hand the implicit semantics are used to construct the ontologies.

In the future it might not be necessary anymore to handcraft ontologies for different applications, since the introduction of web services for many applications will standardize the way to access applications from outside.

Currently we have to build application ontologies for the “filesystem”, “email system” and “favorites”.

SWAP Metadata Model

As described in Deliverable 3.2. all information which is added to the local node repository is annotated according to the SWAP metadata model³.

As described above we extract information from external sources and try to build richer structures from the extracted knowledge. The metadata model can be seen as the link between these two structures. The metadata model provides information about the addition date, confidence in statements, caching information, the peer from which the information came etc. Hence, the model is the basis for the collaboration between the different components.

³<http://swap.semanticweb.org/2003/01/swap-peer>

It is used to retrieve the external sources from its origins (Resource sharing component), to store the confidence ratings used by the peer selector and to enable the network to work more efficiently. Furthermore, usage patterns can be analyzed based on the `additionDate` property. During the rewriting process it will be used to get the original labels on other peers.

2.2 Access to the Local Node Repository

The local node repository stores all information which represent the peers knowledge. Thus, all components need to access the local node repository. There are different possibilities for other components depending on their needs. Straightforward is the use of the `SWAPrepository` API to get statements, or to check whether a statement is in the repository. Furthermore, the API allows for deleting statements and adding statements. Complex queries can be posted to the local node repository using `SERQL`.

The metadata model is of interest for many components, thus an API exists which enables easier access to retrieve, change and add new swabbi-objects to the local node repository. In this way the integration with other components is easy, and overhead is minimized.

Chapter 3

Knowledge Source Integrator

As described in deliverable 3.2 the knowledge source integrator comprises methods for (1) extraction of knowledge from external sources, (2) selection of relevant knowledge, (3) annotation and (4) merging of equivalent knowledge. These methods were included, following the requirements to extract knowledge from different sources which could either be applications on the local peer or knowledge from other peers in the network.

In the remainder of this chapter the different parts of the knowledge source integrator are examined with a focus on aspects of integration with other methods.

3.1 Extractor

The extractor fulfills two tasks, namely the syntactic extraction and the semantic extraction. During the syntactic extraction a connection to a particular application is established and some of the information stored within the application is extracted and transcribed according to the corresponding application ontology. The semantic extraction takes into account the inherent semantics of the notions within a particular application and externalizes those semantics.

3.1.1 Syntactic extractor

As described in deliverable 3.2. the syntactic extractor excerpts information from applications on the peer's computer and annotates them according to the SWAP common ontology¹. Therefore the syntactic extractor provides the means to abstract from certain implementations of a given functionality, e.g. email-client, and concentrates on the information which the SWAP user needs to do her work. Hence, the syntactic extractor transforms the external representations into our internal format, in a way that all other

¹<http://swap.semanticweb.org/2003/01/swap-common>

components can work with the external information. The internal representation language is RDF(S) as described in chapter 2.

Integration of different methods The methods for syntactic extraction of information from different application work separately. If information from a particular application should be included into the SWAP system, a special wrapper/connector has to be written, which annotates the information according to the common model. However, in case of the extractor for MS-Outlook, more than one concept from the common model could be instantiated. Within Outlook emails as well as contacts are organized. Hence, both types of information can be extracted by the same wrapper. The same holds for the Windows filesystem and favorites of IExplorer, since the favorites are stored basically as a special kind of file.

Integration with other components The correct annotation is of particular interest for the resource sharing component. For each application an access point is modelled, e.g. the path for filesystem. In this way the resource sharing component can retrieve the represented resource from the original application, by firstly determine from which application the resource came and secondly using the access point to actually get the resource.

The semantic extraction refers to the concepts within the application ontology. Hence, the syntactic extractor prepares the external information in a way, that the semantic extraction can work with inputs from different applications with the same functionality.

3.1.2 Semantic extractor

In the previous section the integration of the syntactic extractor with other methods has been described. The semantic extractor tries to create semantical information from the extracted information as described in deliverable 3.2. The user of a particular application associates certain semantics with the concepts offered by the application. For example file system folders are used by most users to group documents of the same topic. This implicit semantic can be used and externalized. The main problem is, to decide on the real semantic of the application concept, e.g. the folder label. The considered semantic relations include the “rdfs:Class”, “rdf:type” and “rdfs:subClassOf” properties. The externalization of other properties is extremely complicated and those are the properties we focus on, since they are the most common.

Currently a very easy approach is implemented, which assumes that instances of the SWAP-common concept “swapcommon:folder” are instances of the RDFS concept “rdfs:class”. Furthermore, instances of the SWAP-common concept “swapcommon:person” are assumed to work in a group which is called like a folder object in Outlook.

Integration of different methods In a first step the semantics of the application concepts can be defined by the user manually. This needs an integration of the semantic extractor with the user interface. Furthermore, assuming that static rules can be applied to convert a certain application concept into a specific semantic concept, the user needs an interface to define those rules.

Some methods for the automatic extraction of the explicit semantics of a certain application concept are under development. The inclusion of WordNet and other pre-existing ontologies is one of them. The use of RDFS makes it particularly easy to exchange already existing ontologies and allows for a feasible use of them. Additionally, the extracted information from different applications can be combined to reach a common understanding. The mapping methods have some impact in such a scenario. Ontologies of other peers are a further resource, which can be analyzed to gain deeper insights. In this case the integration with the advertisement and discovery methods is of specific interest. Discovery messages, which ask for ontologies of other peers can be used to solve this problem.

However, the combination with the manual selection will certainly be of importance since automatic methods will probably not have a precision of 100 percent.

3.2 Ontology Integrator

The task of the Ontology Integrator (selector) is to select relevant concepts and properties which shall be integrated into the local node repository. The selector will decide upon the relevance of a particular object considering different objectives.

1. Knowledge from the local peer should be accessible anyway
2. Not everything can be included
3. Knowledge from other peers should be included to augment network efficiency
4. Concepts and properties are more important than instances, since the very objective of a concept is to summarize instances
5. The other components are supported

This means the selection is basically an optimization task with more than one objective. From an integration point of view the selection should take into account the information that the other components need to fulfill their task.

In the remainder we will summarize the different methods to tackle the optimization task and describe the integration of different integration methods and the integration with other components.

3.2.1 Summary of methods

Currently we have implemented the following methods to select from incoming statements.

User interaction The user selects which statements should be integrated.

Monte carlo Statements are selected randomly. A rate of 0 means no statements at all are included. 1 means all statements are included

Types Just concepts and properties are included

Labels Just Statements are included which have the same labels as there are already in the repository

Peer trust Statements which come from a peer that has a trust above a certain threshold are included

Semantic distance Concepts which have a certain semantic distance to known concepts are included.

3.2.2 Integration of different integration methods

User Interaction can be combined with all other methods. The automatic selection process suggests the user the statements to be integrated, while the user then decides if she really likes the decision. However, this procedure is only reasonable for selections on answer statements from user queries. For discovery messages it would overload the user.

Monte Carlo method can be useful in case a decision must be taken on to many statements, and processing time is scarce.

Types, Selection from labels, Peer trust method These methods can be combined with all other selection methods.

3.2.3 Integration with other methods

Local Node Repository The “Selection from labels” methods depends on the information which is stored in the local node repository. Not only the labels of the resources can be used, but also the related labels, which are stored in the swabbi-objects. The “peer trust method” depends on the calculations applied in the rating component. Again, the swabbi-object is the integrating construct, since it stores the information about other peers.

Semantic Integration will have some effect on the selection based on “semantic distance”. During the semantic integration phase labels from external sources are assigned properties which make them to concepts, properties or instances. Hence, the decisions will influence the selection.

3.3 Metadata Integrator

The metadata integrator (annotator) annotates the statements which were selected for integration into the local node repository according to the SWAP metadata model. As described before in 2.1.1, the metadata model defines a clear semantic to describe the location of external resources and other metadata to handle the ambiguity and decentralization, which is inherent in peer-to-peer systems.

When it comes to integration with other methods two aspects of the annotator must be considered.

1. Annotation of resources and statements
2. Rating of the annotated statements

3.3.1 Annotation

All resources which are included into the local node repository are annotated with “swabbi”-objects. There is just one method to annotate the resources, thus different annotation methods need not to be integrated with each other. Other components rely on the information which is stored according to the metadata model, hence it is important that the annotator adheres to the defined semantics.

3.3.2 Rating

The rating is a part of the annotation process. One part of the SWAP metadata model defines the confidence values for statements and the trust values for peers. If new statements are added to the repository those ratings might alter.

The actual ratings influence the peer behavior in several ways. The peer selector depends partly on the trust one has in particular peers. The inferencing will also take into account the confidence ratings in particular statements. The inferencing is involved in all query answer procedures, thus the rating influences almost all other components. The rating of statements is important if the constructed ontology contains contradicting statements. In those cases a decision must be taken, which statement to evaluate.

Due to the impact of the rating, the different components can give feedback to the rating module and thus influence the rating. The automatic rating of statements is certainly the only feasible. However, the user might want to interact in the rating process, hence an integration with user interface is provided.

3.4 Merger

In the SWAP environment ontologies from many different peers are integrated locally. To efficiently use and query them it is necessary to identify entities which are the same. They are then internally merged and treated as one afterwards. Methods for solving this task use the similarity of labels, instances, or structures in the ontologies. Merging can therefore be regarded as a service scanning the incoming statements along with the local node repository and giving back new statements indicating equality between entities.

3.4.1 Integration with other methods

Merging is basically a component of its own. From the variety of algorithms special methods are required, which normally only have a meaning within the merger component (clustering, mapping, etc.). These methods will be reused by different algorithms within the merger component. One example of this can be similarity measures. Many mapping and merging algorithms will require some similarity considerations e.g. based on similar labels. This will only be implemented once and reused thereafter.

3.4.2 Integration with other components

The merger has to interact with different components. First of all it needs access to the local node repository. This is done via the defined API. Within the knowledge source integration process the merger is called with a single method. During this call it is necessary to pass on the new statements and the current local node repository. The result will then be a small repository containing the statements describing entities which are the same. These will finally be integrated into the local node repository. Other components interested in the information can also access the information through this easy standard SWAP format, which is based in RDF(S). This is especially the case for the rating module, which does some recalculations when the local node repository changes. Many more components are affected indirectly by the outcome of the merger. The most important one would be the query engine. Once the merger decides two entities are the same, new information can be inferred and be returned on incoming queries. Besides complete interaction via the component APIs some internal methods of the merger will be of interest for other components. One example is the above described notion of identifying similar entities. This basic method (or group of methods, when based on more than one similarity measure) is also necessary for the peer selector component. If the peer selector can't find a peer matching exactly the query, it will look for peers with similar entities. This is where the similarity measures can be used. A common similarity method is used by both components.

Even though there is a merging method within the user interface edit component this does not directly affect the interactions of the merger. The edit component has the differ-

ent goal of doing manual changes (including merging) and then adding the new statements into the local node repository. It has its own integration applications.

Chapter 4

Informer

The Informer consists of two sub-components: the Advertiser and the Discoverer. These two sub-components work towards the same goal - to promote information about the knowledge of the peers.

The functions of the Informer are tightly integrated with those of the Peer Selector. Based on the knowledge about the expertise of other peers, the Peer Selector will select the peers that are most likely capable of answering a specific query. The means to achieve this integration between Informer and Peer Selector is an Expertise model, which will be described below.

4.1 Advertiser

The Advertiser pro-actively sends information about his knowledge to other peers in the network. In order to do that, it has to decide which information is used for the advertisements and to which peers the advertisements should be sent.

4.1.1 Summary of Methods

Looking at the advertisements themselves, we can distinguish different methods how to represent and promote advertisements.

One naive way to model advertisements is to simply consider a subset of the statements in the Local Node Repository. With this approach, subgraphs of a peers model would be promoted as advertisements. These subgraphs implicitly contain information about the expertise of a peer.

Another approach is to model expertises explicitly. This requires an abstraction of the knowledge contained in the Local Node Repository.

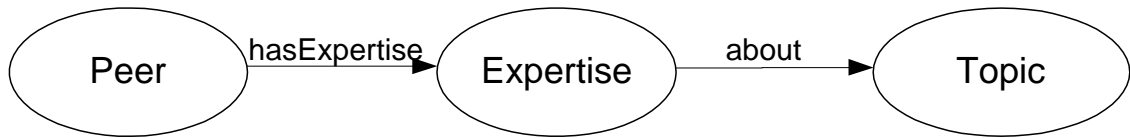


Figure 4.1: Peer Expertise Model

Figure 4.2 shows a general representation of an expertise model. Peers are associated with their expertises. A good expertise model will always contain domain specific information, such as references to topics, e.g. the ACM Topic hierarchy for the domain of computer science literature.

4.1.2 Integration with other methods

Peer Selector As already mentioned, the Informer operates tightly integrated with the Peer Selector. The Peer Selection method is based on a method called Expertise Based Matching, which is similar to capability based matching in resource discovery. The different expertise is extracted from the user knowledge contained in the Local Node Repository and then pro-actively sent to other peers by the Advertiser. The Peer Selector on the other hand extracts the subject from a user query and is then able to apply a matching algorithm to find and select a peer that has the expertise to answer the user query.

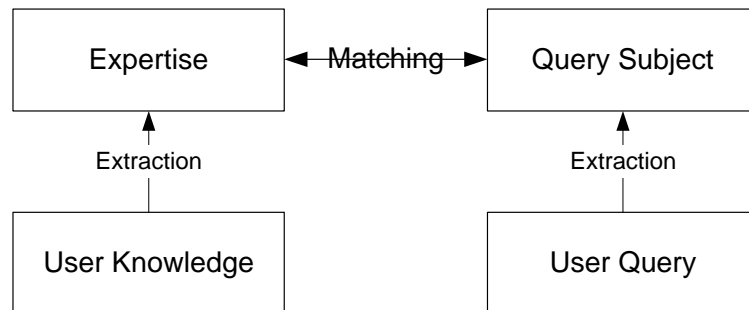


Figure 4.2: Expertise Based Matching

Local Node Repository The Informer stores the knowledge about the expertises of other peers directly in the Local Node Repository. This allows for an easy integration with all methods that may need access to expertise knowledge.

Communication Adapter A new message type – AdvertisementMessage – has been created to allow integration with the Communication Adapter to send advertisements as regular messages in the SWAP P2P communication. The AdvertisementMessage contains statements according to the Expertise Model.

4.2 Discoverer

The Discoverer works similarly to the Advertiser, but instead of proactively sending advertisements with various expertise (Push approach), the Discoverer sends requests to find peers with a specific expertise (Pull approach).

4.2.1 Summary of Methods

In order to find peers that are able to answer specific queries, the advertiser sends requests that specify the required expertise according to the expertise model. If a peer receives such a request and has the specified (or a semantically related) expertise, it will send back a specification of his expertise which can then be used by the original peer as the basis for peer selection.

4.2.2 Integration with other Methods

The discovery requests are based on the same expertise model, in this sense the Discoverer is integrated with the other components, such as the Peer Selector, in the same manner.

Integration of Advertiser and Discoverer

The Advertiser and the Discoverer both function to promote the peers knowledge of peers in the P2P network. The two components do this with different approaches and operate independently of each other. Typically it will be sufficient in a P2P scenario to use either the Advertiser or the Discoverer. To improve flexibility and possibly effectivity and performance of query routing, it might be useful to use both components simultaneously. Deliverable D3.4 (Test Scenarios) shows under which circumstances the use of the single components is sufficient or whether a combination of the two is advisable.

Chapter 5

User Interface

The objectives of the user interface are trifold. Firstly it gives the user the possibility to post queries to the system, secondly it visualizes the local node repository and answers from the network and finally it provides means to edit the local node repository and to ask the user if automatic decisions are correct.

The visualization of ontologies which are formed within a peer-to-peer network imposes some amendments to existing approaches of ontology visualization. However, the research of visualization techniques is not one of the main objectives of SWAP. Thus, we will just briefly describe the integration of the local node repository with the chosen editor “OntoEdit”.

5.1 Integration of OntoEdit

OntoEdit [SEA⁺02] offers many different visualization metaphors for ontologies. The most common among them are hierarchies and the Touchgraph¹.

The SWAP functionalities are included into OntoEdit by means of its plug-in architecture. In this way it is possible to capture the events in OntoEdit and impose the changes on the local node repository. Even though OntoEdit is an Ontology engineering platform it uses a different data model to store the ontology. Therefore, a direct link to the local node repository was not possible. The import of the local node repository is implemented as a view query to the local node repository. This means the user can decide which parts of the ontology she wants to edit, or just select one of the predefined views. The events of change are captured by the SWAP plug-in and transformed into the SWAP data model meanwhile adding information according to the SWAP metadata model.

We have further amended the hierarchy view to distinguish between ontologies from different peers. The concepts and properties from other peers are highlighted in different

¹<http://www.touchgraph.com/>

colors. In addition, the merging of concepts and properties was introduced.

5.2 Integration with other components

As described in the previous chapters the user interface can be used to visualize the changes and suggestions which the different methods make to change the local node repository. The integration of the user interface with the different components/methods were described in the sections of the methods.

Summarizing the integration of the components with the user interface some points are of importance.

- The methods propose changes to the local node repository as well as changes to single statements. Hence, the user can decide for each change, if she likes to accept the change to statement or not.
- User interaction can not be required all the time, the system works. However, it establishes confidence in the system, if it suggests the right changes and the user can approve them.

Chapter 6

Replier

The replier comprises the components for peer selection and query rewriting. Both methods are used on sending a query to other peers. The peer selector decides on the basis of the peer knowledge about the network where to send a query and the rewriter reformulates the query in a sense, that the receiving peers can understand the query. Of course, the rewriting can just take place if the peer knows about the schema of the receiving peer.

In the following the peer selector and the query rewriter are described in the context of integration with other methods.

6.1 Peer selector

The peer selector analyzes the queries a peer posts to the network and decides to which peers a particular query should be send. There are two main reasons for a peer selection process. The first one is to limit the number of messages sent around, thus reducing the network traffic. The second reason aims at selecting peers to reduce the number of irrelevant answers.

The peer selection process can be based on different methods. In deliverable 3.2. we have enumerated the ones used within SWAP, e.g. based on semantic distance, on ratings or on explicitly modelled expertise.

For the rating the SWAP metadata model provides a clear semantic to store the ratings for different statements. The rater itself uses semantic distance measures to come up with a rating. The semantic distance can only be calculated properly, if the merger has correctly identified equivalent concepts on different peers. Additionally to the inclusion of semantic measures to rate statements, peers can be rated by the reliance, uptime and network connection. To evaluate those technical measures the communication adapter is crucial.

The peer selector can only recommend peers, it knows and information about their

knowledge exists. Thus, the information the informer gathers has significant effect on the selection process.

Another important factor are the user queries. The more the user interacts with the system, the more possibilities exist to learn about the user interests. The peer selector can then choose peers, which have been useful in the past.

6.2 Query rewriting

In a heterogenous environment like a peer-to-peer system, peers can model their knowledge in different ways. The query rewriter uses the peers knowledge about a queried peer to formulate a user query in a way, so that the other peer is more likely to understand the query. Two main differences exist which the Query rewriter tries to overcome. The lexical differences, *viz.* the words used to label a certain concept and the schema differences which refer to different modelling approaches. The lexical differences are stored in the SWAP metadata model, hence it is easy to retrieve them. However, schema differences are more complicated to deal with. They are not yet tackled.

Chapter 7

Controller

The controller is the central integration unit. The controller organizes the data flow between the different components of the system. The data flow is independent of the finally used methods within one component. The data flow is defined as the order of the method calls which are applied to a particular message. Currently three different message types are considered within SWAP. (1) The query message is sent to other peers by the user, (2) the advertisement message is sent to other peers to publicize the knowledge of the peer on behalf of the peer and finally (3) the discovery message looks automatically for information on other peers. Currently we intend to have just one data flow, however a different implementation of the controller might be necessary in the future. In the remainder we will describe the data flow for each message.

7.1 Integration of the different methods

For each type of message we consider three different cases.

1. *Send* The message is sent to another peer.
2. *Receive* The message arrives at the receiving peer.
3. *Forward* The message is forwarded to another peer.

In this section we will not emphasise on the particular method which is used by the different components, but rather abstract from it, and concentrate on the data flow.

7.1.1 Query

The user creates a query and wants to get an answer from the peer network. The query transferred to other peers via the query message.

Send query message We assume in the following, that the user interface has transformed the query into SeRQL. Initially the context is added to the query. The context is a set of statements which are related to the query. The context information is extracted from the local node repository. The context includes super- and sub-concepts of the concepts used in the query. This gives the receiving peer the chance to evaluate the query, although it might not exactly understand all concepts used in the query itself. In the second step the peers are selected by the peer selector. Afterwards the query is rewritten according to the knowledge about the particular peer. A query message is generated from the rewritten query and sent to the selected peer.

Receive query message A query which is received by a remote peer, if it was not answered before¹, is evaluated against the local node repository and the answer is returned to the sender. Additionally the context of the query is extracted and passed to the selector. Subsequently the selected statements are annotated, rated, merged and finally included into the repository. In this way the peer learns from queries about the other peers knowledge. Depending on the implementation, user interaction can be involved in some of the methods. The procedure of selecting statements up to the integration is in the following referred to as knowledge integration process.

Receive answer message The received answer is presented to the user. The user can browse the received ontology and in case the answer includes reference to external resources e.g. files download them. The answer is then integrated into the local node repository following the knowledge integration process.

Forward query message Query messages can be forwarded by a peer to other peers in the network. The query message is passed to the “peerselector” and it decides to which peer the query message should be forwarded. Before a query is forwarded, the meta information of the message is evaluated and checked, whether the maximum amount of hops is reached, or the query has been too long in the network. As for receiving query messages, the incoming messages are stored, and a forward is performed, if the query has not been processed before!

7.1.2 Advertisement

Advertisements are used to inform other peers about the peers knowledge.

¹Queries are forwarded. This means a query can reach a peer by different roots. To avoid double processing the queries are stored and new queries are compared to the before received queries.

Send advertisement message The advertisement are instances of the SWAP expertise model². The advertisement is analyzed and some peers are selected accordingly. Finally the advertisement message is sent to the selected peers.

Receive advertisement message Received advertisements are processed separately from queries. Hence, other statement selection strategies are applied for advertisements. Finally the advertisements are included into the local node repository.

Forward advertisement message It is not yet clear, wether advertisements should be forwarded or not. This is currently evaluated. However, the forwarding process would be the same as before. Some peers are selected and the message is sent to them.

7.1.3 Discovery

Discovery messages are means to learn about other peers actively.

Send discovery message The peer deduces from usage of its local node repository, randomly or on user request which expertise to look for. The discovery queries are again augmented with a context and peers are selected.

Receive discovery message A discovery query is processed differently to a user query message. The context of the query is included into the local node repository according to the knowledge integration process. The query itself is evaluated against the local node repository and the answer is returned as instance of the SWAP expertise model.

Receive answer message The answers on discovery requests are treated in the same way as advertisements.

Forward discovery message Discovery message are analyzed by the receiving peer and forwarded to other peers.

²<http://swap.semanticweb.org/2003/07/swap-expertise>

Chapter 8

PMML - Predictive markup modelling language

As the project proceeds intelligent methods will be included into the different components. More intelligent methods will partly use machine learning techniques to capture the inherent knowledge in the user interaction with the system. For bootstrapping as well as improvement purposes it might be useful to exchange learned information between the peers. PMML¹ is an XML based language which provides a way for applications to define statistical and data mining models and to share models between PMML compliant applications. PMML was not considered in the first design of the methods. However, the primitives offered by PMML seem to be the right way to exchange learned statistical knowledge and will be used in the revised methods.

In this chapter we will describe PMML and provide some first ideas how different components can use PMML to exchange knowledge with other peers.

8.1 Overview

Predictive Model Markup Language (PMML) is an XML-based language which provides a quick and easy way for companies to define predictive models and share models between compliant vendors' applications.

PMML provides applications a method of defining models so that proprietary issues and incompatibilities are no longer a barrier to the exchange of models between applications. Hence, different applications can be used to analyze data and/or visualize it, without the consideration of different data models.

It is supported by the Data Mining Group². Participants include IBM, National Center

¹<http://www.dmg.org/index.htm>

²www.dmg.org

for Data Mining, Oracle Corporation, SAS and others.

8.2 Language Primitives

Since PMML is an XML based standard, the specification comes in the form of an XML Document Type Definition (DTD)³. In the following the data-flow within the implementing application and the elements of the DTD are briefly introduced.

8.2.1 Data Flow

PMML defines a variety of specific mining models such as tree classification, neural networks, regression, etc. Equally important there are also definitions which are common to all models, in order to describe the input data itself and the generic transformations, which can be applied to the input data before the model itself is evaluated.

- The `DataDictionary`⁴ describes the actual data which shall be analyzed.
- The `MiningSchema` defines the input values and their weights from which the mining model is derived.
- Transformations can be defined which transfer actual data into standardized values. These are called derived values and represented as `DerivedFields`. The transformations are modelled in the `TransformationDictionary`.
- Derived values can be the origin of further transformations.
- After the model has been generated on the basis of the normalized values, the results need to be retransferred into the original domain. This transformation is provided by the PMML consumer.

8.2.2 Header

In the `Header` element meta-Information about the XML document is summarized. This includes the `Application`, which has generated the model, plain text `Annotation` and a `Timestamp`.

³For the detailed description of the specification see http://www.dmg.org/pmmlspecs_v2/pmml_v2_0.html

⁴XML elements of PMML are highlighted.

8.2.3 Data dictionary

The `DataDictionary` contains definitions for fields as used in mining models. It specifies the types and value ranges. These definitions are assumed to be independent of specific data sets as used for training or building a specific model.

A `DataDictionary` can be shared by multiple models, statistics and other information related to the training set.

The fields are defined as `DataField`. `DataFields` can be categorical⁵, ordinal and continuous. Corresponding operations can be defined on them. Additionally `Values` or `Intervals` can be defined for each `Data` field.

8.2.4 Mining Schema

Each model contains one `MiningSchema` which lists fields as used in that model. This is a subset of the fields as defined in the `DataDictionary`. While the `MiningSchema` contains information that is specific to a certain model, the `DataDictionary` contains data definitions which do not vary per model.

The element to define the `MiningSchema` is the `MiningField`. In the definition of the `MiningField` one can define whether a field is *predicted* or *active*. Furthermore value ranges, replacement values etc. can be set. It is important that the `MiningField` values are the same as defined in the `DataDictionary`.

8.2.5 Transformations

At various places the mining models use simple functions in order to map user data to values that are easier to use in the specific model. For example, neural networks internally work with numbers, usually in the range from 0 to 1. Numeric input data are mapped to the range [0..1], and categorical fields are mapped to series of 0/1 indicators. Similarly, Naive Bayes models internally map all input data to categorical values.

PMML defines various kinds of simple data transformations:

Normalization maps values to numbers. Input can be continuous or discrete.

Discretization map continuous values to discrete values.

Value mapping map discrete values to discrete values. Mapping missing values as a special case of value mapping.

Aggregation summarize or collect groups of values, e.g. compute average.

⁵Categorical fields express differences in an element by labels rather than numbers. A taxonomy 8.2.7 can be defined on these fields.

For each kind of transformation exists a corresponding XML element. The XML element for derived values provides a common element for the various mappings. Derived values can appear in the `DataDictionary` within `DataFields`. They can also appear at several places in the definition of specific models such as neural network or Naive Bayes models. Transformed fields have a name so that statistics and the model can refer to these fields.

The transformations in PMML do not cover the full set of preprocessing functions which may be needed to collect and prepare the data for mining. There are too many variations of preprocessing expressions. Instead, the PMML transformations represent expressions that are created automatically by a mining system. The corresponding expressions are often generated. Similarly, a discretization might be constructed by a mining system that computes quantile ranges in order to transform skewed data.

8.2.6 Statistics

In PMML just a subset for statistics is provided as a basic framework for representing univariate statistics. The element name for all statistical measures is `ModelStats`. The defined statistics include descriptions for `counts`, `numericInfo`, `Quantile`, etc. Other statistics can be defined, but should follow the design schema of the existing ones.

8.2.7 Taxonomy

The values of a categorical field (see also 8.2.3) can be organized in a hierarchy. PMML provides semantics to express parent child relationships. Since, the fields used within SWAP are more likely to belong to an ontology it might be necessary not only to convert the underlying ontology into an reduced parent child relationship but to include the richer semantics into the mining model.

PMML further allows to integrate external data from external tables into the mining model. The links can be provided within the description.

8.2.8 Models

The above described elements of PMML provide a framework to describe input data, the statistics which can be derived, transformations and so on. These elements are used in different mining models. In the following the mining models which are supported by PMML are introduced and their elements described.

Trees The `TreeModel` uses the `MiningSchema`, `ModelStats` and a `Node` element for its description. The tree modeling framework allows for defining either a classification or prediction structure. Each node holds a rule, called `PREDICATE`, that determines

the reason for choosing the node or any of the branching nodes. The Predicates are composed with mostly boolean operations or `SimplePredicates` which are combinations of field value pairs⁶.

Regression The regression functions are used to determine the relationship between the dependent variable (target field) and one or more independent variables. The dependent variable is the one whose values you want to predict, whereas the independent variables are the variables that you base your prediction on.

PMML 2.0 defines three types of regression models: linear, polynomial, and logistic regression.

Linear and stepwise-polynomial regression are designed for numeric dependent variables having a continuous spectrum of values. These models should contain exactly one regression table. The attributes `normalizationMethod` and `targetCategory` are not used in that case.

Logistic regression is designed for categorical dependent variables. These models should contain exactly one regression table for each `targetCategory`. The `normalizationMethod` describes whether/how the prediction is converted into a probability.

Other regression Models can be defined using the element types which are defined for the `General Regression` element.

Cluster Models PMML models for cluster models are defined in two different classes. These are center-based and distribution-based cluster models. Both models have the DTD element `ClusteringModel` as the toplevel type and they share many other element types.

A cluster model basically consists of a set of clusters. For each cluster a center vector can be given. In center-based models a cluster is defined by a vector of center coordinates. Some distance measure is used to determine the nearest center, that is the nearest cluster for a given input record. For distribution-based models (e.g. in demographic clustering) the clusters are defined by their statistics. Some similarity measure is used to determine the best matching cluster for a given record. The center vectors then only approximate the clusters.

The model must contain information on the distance or similarity measure used for clustering. It may also contain information on overall data distribution, such as covariance matrix, or other statistics. Names of coordinates in `CenterFields`, `ClusteringFields` and statistics must be consistent with the names of the fields in the `DataDictionary` and in the transformation dictionary.

⁶see <http://www.dmg.org/pmmlspecs.v2/TreeModel.html> for example

Association Rules The Association Rule model represents rules where some set of items is associated to another set of items. For example a rule can express that a product is often bought in combination with a certain set of other products.

An Association Rule model consists of four major parts the `MiningSchema`, `Item`, `Itemset` and the `AssociationRule`.

The `item` element defines the input data by value and there weight. The `AssociationRule` element relates different `Itemset` and assigns a support value and a confidence value to the relation.

Neural Network For a general introduction to neural networks refer to [RN95]. A neural network has one or more input nodes and one or more neurons. Some neuron's outputs are the output of the network. The network is defined by the neurons and their connections, with corresponding weights. All neurons are organized into layers; the sequence of layers defines the order in which the activations are computed. All output activations for neurons in some layer L are evaluated before computation proceeds to the next layer $L + 1$. Note that this allows for recurrent networks where outputs of neurons in layer $L + i$ can be used as input in layer L where $L + i > L$. The model does not define a specific evaluation order for neurons within a layer.

Each neuron receives one or more input values, each coming via a network connection, and sends only one output value. All incoming connections for a certain neuron are contained in the corresponding `Neuron` element.

To describe a `Neuralnetwork` `NeuralInputs` `NeuralLayer` and `NeuralOutputs` are used.

`NeuralInput` defines how input fields are normalized so that the values can be processed in the neural network. For example, string values must be encoded as numeric values. Each `Neuron` is defined by a `Neural Input`.

A `NeuralLayer` defines the connections weights between the different `Neurons`.

`NeuralOutput` defines how the output of the neural network must be interpreted.

Naive Bayes Naive Bayes uses Bayes' Theorem, combined with a ("naive") presumption of conditional independence, to predict, for each record (a set of values, one for each field), the value of a target (output) field, from evidence given by one or more predictor (input) fields.

Naive Bayes models require that each field (whether target or predictor) be discretized so that for each field, only a small, finite number of values are considered by the model.

The `NaiveBayesModel` uses the `MiningSchema`, `ModelStats`, `BayesInputs` and `BayesOutput` elements for its description. It uses special semantics to define weights for the different input fields and thresholds for missing values. Furthermore

`TargetValueCounts` are defined to express for each input field value the number of occurrences in the training set and `PairCounts` to store the joint occurrences of the predictor value and the target value⁷.

Other models Some of the most important mining models which can be represented with PMML have been described. PMML is still under development and will certainly include more mining models over time. One which is already included but was not introduced here is the schema for `Sequence` mining models. In the end it depends on the specific task which model has to be applied. Furthermore, tools are needed which can be used to interpret the described data.

8.3 Tools

All participants in Data Mining Group provide applications which can either use PMML as import format or export results in PMML.

In the research community the Weka machine learning package is widely used⁸. Weka implements many standard machine learning algorithms. Since it is implemented in Java and is open source it would be the ideal candidate to be included into the SWAP system. There are currently some efforts underway to add PMML as an import format for Weka.

Even without a free available tool PMML is useful, since it allows to externalize the mining task, and transmit the mining results in a standardized way. Hence, it would be possible to set up a separate mining peer, which uses commercial tools and provide the results to the peers within the network.

8.4 Integration

The component which is most likely to use machine learning techniques is the Knowledge Integrator. The creation of ontologies from existing structures as well as the mapping between ontologies from different peers will involve machine learning to some extent. Furthermore, the Informer is the component which will distribute learned information within the peer network. In the following we will describe the use of PMML within these two components in more detail.

⁷see <http://www.dmg.org/pmmlspecs.v2/NaiveBayes.html> for example

⁸<http://www.cs.waikato.ac.nz/ml/weka/>

8.4.1 Knowledge Integrator

The main task of the knowledge integrator is to extract knowledge from external source and assign them to the internal ontology. This task is depended on the external source. Hence different machine learning techniques might be appropriate. Some methods are listed below where machine learning has been applied successfully. The list is not comprehensive it shall rather hint at directions how learning can be applied within the peer-to-peer environment.

Document assignment to classes in the hierarchy In [MRMN98] a naive bayes classifier has been applied to assign new documents to a given hierarchy. The classifier was trained with already assigned documents. In a peer-to-peer setting one could train a classifier on large set of documents as often available in large companies and distribute it within the network. Each peer could than use the classifier to organize its documents according to a general structure.

Hierarchy learning with metadata In [Har03] a classification based on association rules is described. Metadata from text documents is used to classify new documents into a given hierarchy. In a setting in which large training sets are not available, the combination of information on the single peers could help to train a classifier more precisely.

Text clustering Many different approaches exist to group a given set of text documents into a smaller amount of clusters. Peer-to-peer systems could be a good way to collect constraints on document to cluster assignments. Providing constraints has been shown in effective way to improve clustering accuracy [KKM02].

Merging of concepts In [SM01] concepts are identified which could be merged given two different ontologies with documents assigned to concepts. The exchange of the mapping information could improve the collaboration within the peer network.

8.4.2 Informer

The informer basically has to decide which information to share with the other peers in the network. PMML is a way to exchange that information. The test scenarios (see D3.4) will provide the basis for the decision which information should be shared.

Chapter 9

Conclusion

Deliverable 3.5 has looked at the the diverse methods used in the SWAP system and examined the interaction and integration of the methods with each other.

The local node repository plays a central role as storage medium for the information each method processes. With RDF(S) as representation language the information can be stored with clear semantics attached to them. SeRQL as query language enables other components to retrieve information seamlessly. The SWAP metadata model and the SWAP common application ontologies define a model to store information from external sources.

The knowledge integrator selects, annotates and merges the incoming information. Hence, the chosen strategies for each method influence the other methods in several ways. The selection process decides which information is integrated, thus on which information other components can base their decisions. The merger looks for equivalent information in different external sources which is important to tackle the heterogeneity within the network.

The replier selects peers and rewrites queries. Hence, it depends on the strategies for knowledge integration as well as the strategies of the informer. It cannot select a peer without knowing about it, or rewrite a query without knowing about the peers schema. The knowledge integrator and informer provide the knowledge to increase the possibility that peers with relevant knowledge are found. On the other hand do the other components rely on it, because peers which are not selected cannot provide any knowledge.

The informer collects and distributes advertisements of different peers. In this way it facilitates the acquisition of knowledge about the network. The informer depends on the knowledge integrator, to know about the knowledge the user is interested in and provides information for the replier.

The user interface is integrated with all other components, since all components might use it, to ask the user whether automatic decisions were correct or not.

Finally the controller is responsible for the coordination between the different compo-

nents and the data flow.

The integration of methods within one peer is based on RDF(S) and the meta models defined by SWAP. The exchange of knowledge between different peers is also represented in RDF(S). However, when it comes to the integration of learned statistical knowledge between different peers PMML is the right answer. PMML is based on XML and defines clear semantics to exchange statistical models in a uniform way. Different statistical models can be evaluated on the basis of the peers knowledge. The resulting models can be shared with other peers using PMML.

Deliverable 3.5 has emphasized the many aspects of collaboration and integration of the different components and methods. The requirements of a knowledge enabled peer-to-peer syteme are manifold, thus different components were developed to tackle the diverse aspects. However, only through a tight integration of all components the methods can achieve the best results.

Bibliography

- [Har03] Jens Hartmann. Ilp-basierte klassifikation von web-dokumenten. In Bergmann et. al., editor, *Maschinelles Lernen, Wissenentdeckung, Data Mining - FGML 2003, GI-Workshopwoche LLWA*, Karlsruhe, Okt. 6-8 2003. to appear.
- [KKM02] Dan Klein, Sepandar Kamvar, and Christopher Manning. From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. In *The Nineteenth International Conference on Machine Learning*, 2002.
- [MRMN98] Andrew K. McCallum, Ronald Rosenfeld, Tom M. Mitchell, and Andrew Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In Jude W. Shavlik, editor, *Proceedings of ICML-98, 15th International Conference on Machine Learning*, pages 359–367, Madison, US, 1998. Morgan Kaufmann Publishers, San Francisco, US.
- [RN95] Stuart J. Russell and Peter Norvig. *Artificial Intelligence. A Modern Approach*. Prentice-Hall, Englewood Cliffs, 1995.
- [SEA⁺02] Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke. On-toEdit: Collaborative ontology development for the semantic web. In I. Horrocks and J.A. Hendler, editors, *Proc. of the 1st Int. Semantic Web Conf. (ISWC 2002)*, volume 2342 of *LNCS*, pages 221–235, Sardinia, IT, 2002. Springer.
- [SM01] Gerd Stumme and Alexander Maedche. FCA-MERGE: Bottom-up merging of ontologies. In *IJCAI*, pages 225–234, 2001.