Abstract: The issue of conflicting statements in distributed partial user models was a key task that had to be solved in this work package. A semantic conflict resolution approach has been taken, based on derivation rules to solve semantic conflicts, as well as a meta-data conflict resolution approach. The design of modules is presented to resolve conflicting user models, consisting of conflict detection strategies and conflict resolution strategies. The design has been based on related research areas like classical conflict resolution systems, semantic web reasoning, as well as a discussion on issues about integrating conflict resolution strategies into GUMF. The implementation is realised and executed via a specialised inference engine that builds on so called GRAPPLE Derivation Rules, GDRs.

Keyword list: User Modelling, Conflict Resolution, Integration of partial models, Derivation Rules
Summary

This GRAPPLE design and implementation deliverable b) contains the document D6.3a) completely. Slight changes have been made to the design part, especially in the clear definition of the GDR.

The problem of conflicting GrappleStatements, given by different LMS in distributed partial models is addressed. The quality of the user model data can vary and make decisions based on user models especially difficult. The task of this deliverable is to design and implement the use of the user models as well as semantic relations between different GrappleStatements to support the conflict resolution and improve the final learner models.

This deliverable is divided into several chapters: an introduction and a related work chapter, a chapter about the design of the conflict resolution service, a chapter about the implementation with the new GRAPPLE derivation rule format and finally one about the integration of conflict resolution strategies into GUMF.

Authors

<table>
<thead>
<tr>
<th>Person</th>
<th>Email</th>
<th>Partner code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dominikus Heckmann</td>
<td><a href="mailto:heckmann@dfki.de">heckmann@dfki.de</a></td>
<td>DFKI</td>
</tr>
<tr>
<td>Rafael Math</td>
<td><a href="mailto:Rafael.Math@dfki.de">Rafael.Math@dfki.de</a></td>
<td>DFKI</td>
</tr>
<tr>
<td>Erwin Leonardi</td>
<td><a href="mailto:e.leonardi@tudelft.nl">e.leonardi@tudelft.nl</a></td>
<td>TUD</td>
</tr>
<tr>
<td>Geert-Jan Houben</td>
<td><a href="mailto:g.j.p.m.houben@tudelft.nl">g.j.p.m.houben@tudelft.nl</a></td>
<td>TUD</td>
</tr>
<tr>
<td>Fabian Abel</td>
<td><a href="mailto:abel@l3s.de">abel@l3s.de</a> &amp; <a href="mailto:f.abel@tudelft.nl">f.abel@tudelft.nl</a></td>
<td>L3S &amp; TUD</td>
</tr>
<tr>
<td>Eelco Herder</td>
<td><a href="mailto:herder@l3s.de">herder@l3s.de</a></td>
<td>L3S</td>
</tr>
</tbody>
</table>

Table of Contents

SUMMARY ........................................................................................................................................................ 2
AUTHORS ......................................................................................................................................................... 2
TABLE OF CONTENTS.................................................................................................................................... 2
TABLES AND FIGURES................................................................................................................................... 4
LIST OF ACRONYMS AND ABBREVIATIONS ...................................................................................................... 4
1 INTRODUCTION ........................................................................................................................................ 5
2 RELATED WORK ...................................................................................................................................... 5
2.1 Conflict Resolution in OPS 5 ............................................................................................................... 5
2.2 User Model Integration & Ubiquitous User Modelling ...................................................................... 6
2.3 Avoiding conflicts by enhancing the Semantics ............................................................................... 6
2.4 Rules on the Semantic Web ................................................................................................................. 8
   2.4.1 Rule Markup Language ................................................................................................................... 8
   2.4.2 Semantic Web Rule Language ....................................................................................................... 9
   2.4.3 Rule Interchange Format .............................................................................................................. 10
   2.4.4 REWERSE Rule Markup Language .............................................................................................. 10
3 DESIGN OF THE CONFLICT RESOLUTION SERVICE ................................................................. 11

3.1 Possible Conflict (Resolution) Categories ........................................................................ 11

3.2 Example Scenarios with Conflicting Statements ............................................................ 12
  3.2.1 Scenario A) .................................................................................................................. 12
  3.2.2 Scenario B) .................................................................................................................. 13

3.3 Conflict Resolvers and Conflict Resolution Strategies ..................................................... 13

3.4 Architecture of Smart Grapple Statement Retrieval ......................................................... 15

4 THE DESIGN OF GRAPPLE DERIVATION RULES FOR CONFLICT RESOLUTION ............. 16

4.1 Format .................................................................................................................................. 16

4.2 Example ................................................................................................................................ 17

4.3 GDR and Conflict Resolution Service ................................................................................ 21

5 DOCUMENTATION ON IMPLEMENTED CONFLICT RESOLVERS ...................................... 22

  5.1.1 Most recent .................................................................................................................. 22
  5.1.2 Most named ................................................................................................................ 23
  5.1.3 Most specific .............................................................................................................. 23
  5.1.4 Most personal ............................................................................................................ 24
  5.1.5 Highest Level ............................................................................................................. 27
  5.1.6 Return All, None, Random ......................................................................................... 28
  5.1.7 Return private, expired, replaced ................................................................................ 29
  5.1.8 Final Remarks ............................................................................................................ 29

6 INTEGRATION OF CONFLICT RESOLUTION INTO GUMF ................................................... 29

7 CONCLUSION AND OUTLOOK ............................................................................................... 30

REFERENCES ............................................................................................................................ 31
Tables and Figures

List of Figures

**Figure 1** - Precision of mapping keywords (tags and categories) to DBpedia URIs. ........................................ 8
**Figure 2**  Example visualization of three conflicting GrappleStatements ................................................................. 12
**Figure 3**  Example statements that form a semantic conflict .................................................................................. 13
**Figure 4**  Example of Conflict Resolution Strategies ............................................................................................. 15
**Figure 5**  Smart GrappleStatement Conflict Resolution Architecture ................................................................ 16
**Figure 6**  Applications A1 and A2 Dataspaces ........................................................................................................ 18
**Figure 7**  A GDR Rule (Rule 1) ................................................................................................................................ 19
**Figure 8**  A GDR Rule (Rule 2) ................................................................................................................................ 20
**Figure 9**  Architecture of the GRAPPLE User Modeling Framework (GUMF) with Conflict Resolution ......................... 30

List of Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLD</td>
<td>Basic Logic Dialect (RIF)</td>
</tr>
<tr>
<td>GRAPPLE</td>
<td>Generic Responsive Adaptive Personalised Learning Environment</td>
</tr>
<tr>
<td>GALE</td>
<td>GRAPPLE Adaptive Learning Environment</td>
</tr>
<tr>
<td>GCC</td>
<td>GRAPPLE Conversion Component</td>
</tr>
<tr>
<td>GDR</td>
<td>GRAPPLE Derivation Rule</td>
</tr>
<tr>
<td>GrappleBroker</td>
<td>A service to store and retrieve partial user models represented as GrappleStatements</td>
</tr>
<tr>
<td>GrappleStatement</td>
<td>A Unit of information about a user together with contextual meta data.</td>
</tr>
<tr>
<td>GUMF</td>
<td>GRAPPLE User Model Framework</td>
</tr>
<tr>
<td>IMS-LIP</td>
<td>IMS Learner Information Package Specification</td>
</tr>
<tr>
<td>LMS</td>
<td>Learning Management System</td>
</tr>
<tr>
<td>OPS5</td>
<td>Official Production System 5</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>PRD</td>
<td>Production Rules Dialect</td>
</tr>
<tr>
<td>R2ML</td>
<td>REWERSE Rule Markup Language</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RIF</td>
<td>Rule Interchange Format</td>
</tr>
<tr>
<td>SWRL</td>
<td>Semantic Web Rule Language</td>
</tr>
<tr>
<td>TEL</td>
<td>Technology-Enhanced Learning</td>
</tr>
<tr>
<td>UM</td>
<td>User Model (or sometimes “User Modelling”) a set of GrappleStatements</td>
</tr>
<tr>
<td>UMI</td>
<td>User Model Integration</td>
</tr>
<tr>
<td>WP</td>
<td>Work Package</td>
</tr>
</tbody>
</table>
1 Introduction

In a distributed user modelling scenario the problem of conflicting statements in distributed partial user models has to be addressed. The quality of the user model data can vary and make decisions based on user models especially difficult. The aim of this deliverable is to use the user models quality ratings from deliverable D6.2) as well as semantic relations between different statements to support the conflict resolution and to improve the final user models.

The deliverable D6.3 consists of a design (D6.3a) and implementation (D6.3b) of extensions to the D6.1 and D6.2 services, to handle conflicting statements in partial user models. The implementation will follow in deliverable (D6.3b).

User Model Integration (UMI) is a reasonably new research area and in the first workshop about that topic Peter Brusilovsky stated “The integration of adaptive educational systems is changing from an interesting research problem into an important practical task. One of the major challenges that need to be accepted on the way is the development of mechanisms for student model integration.”, see [15].

In GRAPPLE it is believed that user/student model integration can be successfully realised with the notion of “conflict resolution” that played an important role in artificial intelligence some decades ago. Interestingly in recent years this issue has been revitalised within the “Semantic Web Reasoning” approach. Unfortunately there is still no “silver bullet” approach to integrate conflict resolution reasoning into the semantic web. The statement “Peter is interested in Java” could be understood as an example of a semantic conflict: it is per se not exactly clear whether the term Java refers to the Indonesian island or to the programming language.

This deliverable is a design and implementation deliverable. It has a chapter about related work, a chapter about the conflict resolution service in mind, a chapter about Grapple Derivation Rules for conflict resolution and a chapter about the integration of conflict resolution into GUMF.

2 Related Work

Conflict resolution is a range of methods for alleviating or eliminating sources of conflict (according to [http://en.wikipedia.org/wiki/Conflict_resolution], retrieved 11.01.2010). Processes of conflict resolution in real life generally include negotiation, mediation and diplomacy.

In computer science and specifically the branches of knowledge engineering and artificial intelligence, a conflict resolution inference engine is a computer program that tries to derive answers from a knowledge base to solve the conflict. It is the intelligence used to reason about the information in the knowledge base for the ultimate purpose of formulating new conclusions. Inference engines are considered to be a special case of reasoning engines, which can use more general methods of reasoning.

The discussion starts with a history of conflict resolution in computer science.

2.1 Conflict Resolution in OPS 5

In the historical OPS5 system (see [http://en.wikipedia.org/wiki/OPS5] or [13] for instance) a choice of two conflict resolution strategies is presented to the programmer. The LEX strategy orders instantiations on the basis of recency of the time tags attached to their data items. Instantiations with data items having recently matched rules in previous cycles are considered with higher priority. Within this ordering, instantiations are sorted additionally on the complexity of the conditions in the rule. The other strategy, MEA, puts special emphasis on the recency of working memory elements that match the first condition of the rule. (The latter heuristic is heavily used in means-ends analysis.)

OPS5 is a rule-based or production system computer language, regarded as the first language of its kind to be used in a successful expert system.

The OPS (which stands for Official Production System) family was developed in the late 1970s. Since it is based on the efficient Rete algorithm it was possible to scale up to larger problems involving hundreds or thousands of rules. OPS5 uses a forward chaining inference engine; programs execute by scanning "working memory elements" with classes and attributes looking for matches with the rules in "production memory".

Rules have actions that may modify or remove the matched element, create new ones, perform side effects such as output, and so forth. Execution continues until no more matches can be found. (see [http://en.wikipedia.org/wiki/OPS5])

For GRAPPLE it is interesting to analyse the conflict resolution strategies. For example the recency-filter could be used in analogy. This means the more recent the information is, the higher the probability it is more beneficial than older information.

## 2.2 User Model Integration & Ubiquitous User Modelling

A key challenge for GRAPPLE’s distributed approach to handle user- and learner-models is user model integration. Different systems can represent the same information in different ways, using different syntactic and conceptual structures; a way of negotiation and clarification of data among systems is therefore required. This is fundamental not only to access information, but also to reuse information. In the GRAPPLE project a harmonised syntactical structure of GrappleStatements (see WP6.1 and WP2.1) has been introduced. However, this is only the first step towards solving the issue of user model integration.

In a recent workshop about Ubiquitous User Modelling\(^2\), participants have shown an increased awareness of the need for standards for representing and exchanging user profile data. At the same time it appeared that dealing with syntactic and semantic heterogeneity of user models is complicated, especially in an open environment like adaptive hypermedia, adaptive web-based systems and adaptive learning environments. The issue is: how can semantic heterogeneity be handled for ubiquitous user modelling? How can the Semantic Web technologies be employed to cope with such heterogeneity?

These issues are also important to resolve conflicting user models in GRAPPLE, while “Ubiquitous User Modelling” describes ongoing modelling and exploitation of user behaviour with a variety of systems that share their user models. These shared user models can either be used for mutual or for individual adaptation goals. Ubiquitous user modelling differs from generic user modelling by the three additional concepts: ongoing modelling, ongoing sharing and ongoing exploitation. Systems that share their user models will improve the coverage, the level of detail and the reliability of the integrated user models and therefore allow better functions of adaptation. Ubiquitous user modelling implies new challenges of scalability, scrutability and privacy that strongly relate to the situation in GRAPPLE.

## 2.3 Avoiding conflicts by enhancing the Semantics

In GRAPPLE, users are modelled using Grapple statements (see Deliverable 2.1), which consist of subject-predicate-object structures enriched with metadata such as date of creation, level denoting to which degree the statements seems to be true or the creator of the statement. Some conflicts may arise because the semantics of a given statement are not clearly defined. While the subject usually refers to a user and the predicate to some domain vocabulary that is shared among the involved systems and therewith semantically well defined, the object can sometimes be unclear. For example, the meaning of the object value might be ambiguous for the following Grapple statement (formatted in RDF/XML).

```xml
<?xml version="1.0"?><rdf:RDF
   xmlns="http://www.grapple-project.org/grapple-core/#"
   xmlns:gc="http://www.grapple-project.org/grapple-core/#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xml:base="http://www.grapple-project.org/grapple-core/">
   <gc:Statement rdf:ID="example-app-mary-2010-01-10-165252631562">
     <!-- main part -->
     <gc:user rdf:resource="http://grapple-project.org/user/mary"/>
     <gc:predicate rdf:resource="http://xmlns.com/foaf/0.1/interest"/>
     <gc:object rdf:datatype="&xsd;anyURI">Java</gc:object>
     <gc:level rdf:datatype="&xsd;double">0.9</gc:level>
   </gc:Statement>
</rdf:RDF>
```

\(^2\) Workshop about Ubiquitous User Modelling, supported by GRAPPLE: http://www.u2m.org/ubiqum2009/
The Grapple statement above states that Mary (http://grapple-project.org/user/mary) is interested (http://xmlns.com/foaf/0.1/interest) in “Java”. However, it is not exactly clear whether the term Java refers to the Indonesian island or to the programming language. Another problem that can cause conflicts is given by different words (synonyms) that actually refer to the same concepts. For example, there could be the following two Grapple statements.

1. (http://grapple-project.org/user/mary, http://xmlns.com/foaf/0.1/interest, "J2EE")

Both statements basically say that Mary is interested in the enterprise edition of the Java programming language. However, the first statement uses the abbreviation while in the second statement the full name of the edition is applied.

Solutions to such problems that are inspired by Semantic Web paradigms try to map such ambiguous or synonymic keywords to URIs that clearly describe the meaning of the words. There already are systems such as Faviki that foster the usage of URIs instead of keywords. There are also approaches that automatically map keywords to meaningful URIs. For example, Marchetti et al. present an approach, which exploits WordNet and Wikipedia URLs to detect and define the meaning of words in collaborative tagging systems [27]. In [28] Passant et al. present a framework that can be applied to attach URIs to tag assignments that clearly describe the meaning of the corresponding tag.

The feasibility of mapping keywords to DBpedia URIs has been investigated. DBpedia[5] [26] can be described as the Semantic Web version of Wikipedia. DBpedia makes a huge portion of Wikipedia articles available in RDF format and enriches each article with additional metadata, e.g. articles are classified using the Yago ontology [29]. In an evaluation two different lightweight approaches for mapping words to DBpedia URIs have been compared.

- **DBpedia Lookup**: The naive lookup strategy invokes the DBpedia lookup service with the keyword that should be mapped to a URI as a search query. DBpedia ranks the returned URIs similarly to PageRank [30] and our naive mapping strategy simply assigns the top-ranked URI to the keyword in order to define its meaning.

- **DBpedia Lookup + Feedback**: The advanced mapping strategy is able to consider feedback while selecting an appropriate DBpedia URI. Whenever a keyword occurs, for which a correctly validated DBpedia URI already exists in the repository, that URI is selected. Otherwise the strategy reverts back to the naive DBpedia Lookup.

Both strategies on the data corpus of the TagMe! system[6] that enables users to annotate resources with freely chosen keywords have been investigated. It also enables users to categorise resources into categories that are suggested by the system but can also be extended by the end-users.

---

3 http://faviki.com
4 http://wordnet.princeton.edu/
5 http://dbpedia.org/
6 http://tagme.groupme.org/
Figure 1 shows the results of the evaluation [31] with respect to precision (= number of correctly mapped words / number of mapped words). The mappings of the naive approach result in a precision of 79.92% for mapping tags to DBpedia URIs and 84.94% for mapping categories considering only those keywords where a DBpedia URI that describes the meaning properly exists (e.g., keywords such as "me" or "to-read" were not fed into the algorithms). The consideration of feedback improves the precisions of the naive DBpedia Lookup considerably to 86.85% and 93.77% respectively, which corresponds to an improvement of 8.7% and 10.4%. As the mapping accuracy for categories is higher than the one for tags, it seems that the identification of meaningful URIs for categories, which are suggested by the system, is easier than for totally freely chosen tags. In summary, the results of the DBpedia mapping are very encouraging. The mapping strategies themselves can be enhanced by also considering the context of the tag/category that should be mapped. For example, for mapping a tag assignment one could select the DBpedia URI, which best fits to the DBpedia URIs of the categories that are related to the context of the tag assignment (e.g., that are already assigned to the same resource). The DBpedia mapping reduces the number of distinct tags and categories within dataset by 14.1% and 20.9% respectively, which promises a positive impact on the recall when performing keyword search. The strategies therewith successfully identify synonyms that are a potential source of conflict.

2.4 Rules on the Semantic Web

This section briefly reviews the rule language and rule interchange language on the Semantic Web since rules will play a major role in the conflict resolution process.

2.4.1 Rule Markup Language

The Rule Markup Language (RuleML) [19] defined by the Rule Markup Initiative is a markup language developed to express both forward (bottom-up) and backward (top-down) rules in XML for deduction, rewriting and additional inferential-transformational tasks. The Rule Markup Initiative develops a modular RuleML specification and transformations from and to other rule standards/systems. Moreover, it coordinates the development of tools to elicit, maintain and execute RuleML rules. RuleML itself covers the entire rule spectrum, including derivation rules, transformation rules and reaction rules. It can therefore specify queries and inferences in Web ontologies, mappings between Web ontologies and dynamic Web behaviours of workflows, services and agents [16].
2.4.2 Semantic Web Rule Language

Semantic Web Rule Language (SWRL) [16] is a proposal for a Semantic Web rules-language that is based on a combination of the OWL DL and OWL Lite sublanguages of the OWL Web Ontology Language [17,18] with the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language [19].

Rules are like an implication between an antecedent (body) and consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, the conditions specified in the consequent must also hold. In the human readable syntax, a rule looks like:

\[ \text{antecedent} \implies \text{consequent} \]

where both antecedent and consequent are conjunctions of atoms written \( a_1 \land \ldots \land a_n \). Variables are indicated using the standard convention of prefixing them with a question mark (e.g., \( ?x \)). Using this syntax, a rule asserting that the composition of \( \text{hasParent} \) and \( \text{hasBrother} \) properties implies the \( \text{hasUncle} \) property would be written:

\[ \text{hasParent}(x,y) \land \text{hasBrother}(y,z) \implies \text{hasUncle}(x,z) \]

This rule says if \( x \) has parent \( y \) and \( y \) has brother \( z \), then \( x \) has uncle \( z \).

The XML Concrete Syntax of SWRL is a combination of the OWL Web Ontology Language XML Presentation Syntax [20] with the RuleML XML syntax [19]. The above rule in XML concrete syntax is shown below.

```xml
<ruleml:imp>
  <ruleml:_rlab ruleml:href="#example1"/>
  <ruleml:_body>
    <swrlx:individualPropertyAtom swrlx:property="hasParent">
      <ruleml:var>x</ruleml:var>
      <ruleml:var>y</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom swrlx:property="hasBrother">
      <ruleml:var>y</ruleml:var>
      <ruleml:var>z</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:individualPropertyAtom swrlx:property="hasUncle">
      <ruleml:var>x</ruleml:var>
      <ruleml:var>z</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_head>
</ruleml:imp>
```

A SWRL rule can also be in RDF. The RDF concrete syntax for the above rule is as follows.

```rdf
<swrl:Variable rdf:ID="x"/>
```
2.4.3 Rule Interchange Format

The Rule Interchange Format (RIF) [24] is a standard in development within the W3C Semantic Web Activity. When completed, it will be a component of the semantic web, along with (principally) RDF [25] and OWL [17,18]. Although originally envisioned by many as a "rules layer" for the semantic web, in reality the design of RIF is based on the observation that there exist many "rules languages" and what is needed is to exchange rules between them [24]. There are two categories of dialects in RIF: logic-based dialects and dialects for rules with actions. Generally, logic-based dialects include languages that employ some kind of logic, such as first-order logic (often restricted to Horn logic) or non-first-order logics underlying the various logic programming languages. The rules-with-actions dialects include production rule systems as well as reactive (or event-condition-action) rules. The standard RIF dialects are Core, BLD and PRD. These dialects depend on an extensive list of data types with built-in functions and predicates on those data types. The Core dialect comprises a common subset of most rule engines. The Basic Logic Dialect (BLD) adds features to the Core dialect that are not directly available such as: logic functions, equality in the then-part and named arguments. The Production Rules Dialect (PRD) adds the notion of forward-chaining rules.

2.4.4 REWERSE Rule Markup Language

REWERSE Rule Markup Language (R2ML) [21] is developed by the REWERSE Working Group I1 [22] for the purpose of rules interchange between different systems and tools. It is a comprehensive and user-friendly XML rule format that allows: interchanging rules between different systems and tools, enriching ontologies by rules, connecting your rule system with R2ML-based tools for visualisation, verbalisation, verification and validation. It integrates the Object Constraint Language (OCL) [23], the Semantic Web Rule
Language (SWRL) [16] and the Rule Markup Language (RuleML) [19]. Integrity rules, derivation rules, production rules and reaction rules are supported by R2ML. Integrity rules, also known as (integrity) constraints, consist of a constraint assertion, which is a sentence in a logical language such as first-order predicate logic or OCL. R2ML derivation rules have “conditions” and “conclusions”. Production rules have “conditions”, “post-conditions” and a “produced action”.

Finally, please refer to the documents [1], [2], [3], and related work [5], [7], [11] that are not described in this deliverable but also indirectly influenced the subsequent development.

3 Design of the Conflict Resolution Service

This conflict resolution service is based heavily on work done in [4] and [9] and is applied to the new situation in GRAPPLE of distributed, generic, personalised learning environments.

In widely spread distributed user modelling and adaptive system scenarios as given in GRAPPLE it cannot be expected that all systems use the same standards for representing user models and context information. However, as a common basis and to narrow down the research area, it is assumed that all systems involved in the information exchange are able to use the framework of GrappleStatements: either as their main representation language or at least as an additional transforming feature. To support this assumption for example, the IMS-LIP Learner Information Package (as used by the LMSs to exchange user data) has recently been translated into GrappleStatements. Even though it has been assumed that all involved user-adaptive and context aware systems are supposed to communicate with the same grammatical framework of GrappleStatements and GrappleQueries, there is still a number of interesting points of possible ambiguity left to be solved.

3.1 Possible Conflict (Resolution) Categories

As every LMS is allowed to enter statements into repositories, some information might be contradictory. Conflicts among GrappleStatements, such as a contradiction caused by different opinions of different creators or changed values over time, are categorised as follows (adapted from [4]):

1. REPRESENTATIONAL & SYNTACTICAL LEVEL:
   Each system can choose between a variety of possible representations to express the same information which leads to the so-called variation mapping. The statements can for instance differ in the use of the statement attributes like subject, predicate, object, level, creator, created etc.. Clear modelling guidelines are necessary.

2. SEMANTICAL LEVEL:
   The systems are not forced to use the same vocabulary, that is to say the same ontology, to represent the meaning of the concepts, which leads to user model integration problem number one: ontology merging and semantic web integration. (The outcomes from deliverable D5.2 have to be analysed and integrated, the GCC = Conversion components between GRAPPLE and LMSs)

3. OBSERVATION AND INFERENCE LEVEL:
   Several sensors can see the same things differently and claim to be right, measurement errors can occur, systems may have preferred information sources (The rating and reputation system of deliverable D6.2 can be utilised here)

4. TEMPORAL AND SPATIAL LEVEL:
   Information can be out of date or out of spatial range, a degree of expiry can hold, thus reasoning on temporal and spatial meta data becomes necessary (Temporal reasoning can be built into GUMF directly since the creation time is stored through the Dublin Core attributed “created”)

5. PRIVACY AND TRUST LEVEL:
   Information can be hidden, incomplete, secret or falsified on purpose. A system of trustworthiness could be applied (Rhe rating and reputation system of deliverable D6.2 can also improve the conflict resolution)
All these aspects are content- and domain independent and can be relevant for user related information as well as context related information. In the next section several conflicts are analysed and categorised and possible solutions are presented. In relation to the GrappleQuery retrieval task, the two concepts of precision (How many retrieved statements are really relevant?) and recall (Is all relevant information retrieved?) are also taken into consideration. This section is partly based on the research described in [12] and on ideas about meta rules and conflict resolution in OPS5, see for example [13].

3.2 Example Scenarios with Conflicting Statements

3.2.1 Scenario A)
Imagine GALE has to recommend the next learning object Mary should choose in her mathematics course and GALE applies the following rule:

\[
\text{if the learner has high knowledge level in mathematics, GALE start with X}
\]
\[
\text{but if the learner’s knowledge level is low, GALE recommends Y.}
\]

GALE’S knowledge consists of the three following, conflicting statements about Mary’s knowledge in mathematics: LMS C claims that Mary is a good, LMS B claims that Mary is medium while Mary herself claims that she is bad in mathematics.

![Figure 2](image-url)  
**Figure 2** Example visualisation of three conflicting GrappleStatements

Figure 1 visualises the three statements as rectangles that also indicate additional meta data dimensions: the time-axis shows the created values which reveals that LMS B’s statement is the oldest one, while LMS C’s statement is the most recent one. The height of the rectangle relates to the confidence value: the confidence rating of LMS B’s statement is the highest while the user’s statement has the lowest confidence. So, how should GALE decide?

How should such conflicts be resolved? Naive approaches could ask: should we return the latest entry? Should we return a random element? Should we return the one with the highest confidence value or should...
we prefer the learner’s own entry? Conflicting statements turned out to be a complex and abstract problem but with no generic solution that is valid for all situations. However, in this approach the means is supported to express the intended conflict resolution methods as described in section 3.3. Conflicts on the semantic level are presented in the following example.

### 3.2.2 Scenario B)

Assume that a GRAPPLE mobile learning environment, used with a mobile device, uses the following energy saving and contrast optimising adaptation rule if its device battery is low:

* if the surrounding brightness is low then set the display brightness to its minimum,
* if the surrounding brightness is medium or high then set the display brightness to medium.

Since the mobile device has no light sensor itself, it needs to retrieve and infer information about the *surrounding brightness* from the user model and context service. The current position of the mobile device should be *Room124* and the light situation that is given by the semantically conflicting statements:

1) subject=*Room124* predicate=brightness range=lowHigh object=low
2) subject=Saarbrücken predicate=weather object=sunny with no clouds
3) subject=Light124.1 predicate=switched range=onOff object=on

**Figure 3** Example statements that form a semantic conflict

The most direct statement 1) about the brightness in *Room124* claims that it is dark. However, statement 2) claims that the sun is shining. Since the device is inside a building which could hinder the sun to lighten the surroundings, the situation is not clear. Additionally, statement 3) claims that the lights in this room are switched on. This conflict is difficult to detect since ontological reasoning, spatial reasoning and qualitative reasoning might be necessary to handle this problem satisfactorily. Only a complete knowledge base and general semantic web reasoning can help.

### 3.3 Conflict Resolvers and Conflict Resolution Strategies

*Conflict Resolvers* are a special kind of filters that control the *conflict resolution process*. An ordered list of these resolvers defines the *conflict resolution strategy*. They are modelled in the GrappleQuery.strategy attribute. These resolvers are needed if the *match process* and *filter process* leave several conflicting statements as possible answers. Three kinds of conflict resolvers can be identified:

- The most(n)-resolvers that use meta data for their decision,
- The add-resolvers that add expired or replaced statements to the conflict sets, and
- The return-resolvers that don’t use any data for their selection.

**mostRecent(n)**

Especially where sensors send new statements on a frequent basis, values tend to change quicker than they expire. This leads to conflicting non-expired statements. The *mostRecent(n)* resolver returns the *n* newest non-expired statements, where *n* is a natural number between 1 and the number of remaining statements.

**mostNamed(n)**

If there are many statements that claim A and only a few claim B or something else, then *n* of the “most named” statements are returned. Of course it is not certain that the majority necessarily tells the truth but it could be a reasonable rule of thumb for some cases.

**mostSpecific(n)**

If the range or the object of a statement is more specific than in others, the *n*—“most specific” statements are returned by this resolver. For example if: predicate=hasKnowledge, object=SolarSystem and first range=yesNo while the second range=Novice-Occasional-Professional-Expert-Grandmaster, the statement with the second range contains a more specific
information. Another specificity range ordering is for example: yesNo < lowMediumHigh < 0%-100%

mostPersonal(n)
If the creator of the statement is the same as the statement’s subject (a self-reflecting statement), this statement is preferred by the mostPersonal(n) resolver. If an is-friend-of relation is defined, statements by friends could be preferred to statements by others. However, this resolver bears the problem that users might not be their best judge. However, due to privacy arguments, in some situations the user’s own statements that are given (on purpose) should be preferred.

addExpired
Per default the already expired statements are filtered out. However, if one wants to take them into consideration, the addExpired-resolver adds these statements to the conflict sets.

addReplaced
Statements that are marked with the replaced-flag by other statements are also per default filtered out and not considered in the situation retrieval process. The addReplaced-resolver brings these statements back into the process.

addPrivate
Statements that do not pass the privacy settings are always filtered out. However, for development, testing and administrative reasons experimental private statements may also be recognised with the addPrivate-resolver.

returnAll
If the remaining conflict set should not be resolved any further by the integrated mechanism, the resolver returnAll returns all remaining statements that can then be resolved by an external conflict resolution method, resolved by introspection or left unresolved since this approach also allows conflicting extensions in parallel.

returnNone
If a conflict that could not be resolved still occurs until the returnNone resolver is applied, no statement is returned. This is a very safe way not to say something wrong.\footnote{This rule could be compared with skeptical inheritance in non-monotonic reasoning: I don’t know!}

returnRandom(n)
If after applying several filters still no unique value is found but a unique answer is expected, a random pick will be offered by this resolver. This credulous behaviour is selected by the requestor and therefore acceptable.

returnDialog
If no unique value is found, an alternative conflict resolution strategy could be clarification by dialog\footnote{The idea of clarification by dialog was recommended by Vania Dimitrova.}. In some cases an appropriate human-computer dialog will be initiated in this case, (triggered by GUMF via the LMS)

These conflict resolver rules are based on common sense heuristics. An important issue to keep in mind is the problem that resolvers and strategies imply uncertainty. To contribute to this, the confidence value of the resulting statement is appropriately changed; the conflict situation is also added to the evidence attribute. Additional ideas like calculating the average value or the maximum value of the statement’s object can be covered by the function attribute that allows the application of mathematical functions to the value of either one statement or a set of statements. The function evaluation forms the last part of the control procedure.

Figure 4 shows several examples of conflict resolution strategies. In general, a strategy can be defined by every combination of resolvers, but not all make sense.

1.) strategy=returnAll
2.) strategy=mostRecent(1)
3.) strategy=mostRecent(4) / returnRandom(2)
The first conflict resolution strategy is the empty one, which means that no conflict resolution will be applied and all statements that pass the match and filter process will be returned. Note that if the query.strategy attribute is left empty, a default strategy will be applied but not necessarily the returnAll one. The second conflict resolution strategy is adequate for statements that are frequently renewed by only one sensor, thus simply the last entry is returned. The third strategy is slightly more complex: the four most recent statements are handed over. However, only two statements of these four are returned by random selection. The fourth one selects the five most-named ones and returns the most specific one among them. The fifth and last conflict resolution strategy first selects the most personal ones and finally applies the average function to calculate the average value. This strategy could be interesting for user model dimensions that do not change over time like personality traits.

When revisiting the prefacing example of figure 2, (Mary's knowledge in mathematics) and applying the strategies of figure 4, the second and the third strategy would return high, the fourth strategy would return medium while the fifth strategy would return low. However, the first strategy returns all three values and therefore is no help in this situation. These completely different results show the power and the challenge of the conflict resolution strategies. However, they also indicate that not all conflicting statements can be satisfactorily solved by resolution strategies. As different strategies lead to different statements and resulting values, the choice of the “right” conflict resolution strategy isn’t that obvious.

Different classes of problems need different conflict resolution strategies. The open question is if there is any correlation of the best strategy in certain circumstances. It is expected that the user model dimension equivalence classes already correlate with the best strategies. The conflict resolution topics discussed so far left the semantics of the statement’s main part out of consideration. However, both need to be integrated: conflict resolution with semantic analysis.

A more complex problem is how to detect conflicts even if the auxiliary and predicate in the conflicting statements (or in the query and the statements) differ, even though the same ontology is used? For example, system B could use predicate=hasKnowledge and object=chess while LMS C uses predicate=hasAbility and object=boardGames. In such a case, a semantic property mapping is already needed. See [14] for a discussion about the semantic conflict resolution ontology. A closely related problem arises if statements refer to different ontologies. In such a case additional ontology mapping is needed in order to be able to detect and solve these conflicts. For the challenge of detecting possible conflicts a technique is used that is also used in production systems: classifying statements into so-called conflict sets. Mathematically speaking these statements are classified into equivalence classes.

In summary, it has been demonstrated that the complex task of detecting conflicting statements in a repository can be analysed with a fine-grained definition of conflict sets that represent syntactical as well as semantic conflicts. Ontological reasoning is mostly integrated by using semantic mapping functionality. However, additional research (like the experimental analysis in GRAPPLE of the conflict detection classes) needs to be carried out at this point.

### 3.4 Architecture of Smart GrappleStatement Retrieval

The architectural diagram in Figure 5 shows the semantic conflict resolution part of the retrieval process.

The numbered ovals indicate the reading direction and represents:

1. (no icon) the request in GrappleQuery that has to be parsed first.
2. (no icon) the distributed retrieval of GrappleStatements.
3. (no icon) the input to the conflict resolution process.
4. the three syntactical procedures VariationMapping, RemoveExpired and RemoveReplaced.
5. the semantic procedure SemanticMapping that is based on knowledge in ontologies (like GUMO, see [10], SUMO/MILO, see [4]) and the knowledge base WordNet.
(6) the detection of conflicts and the construction of conflict sets as defined in [4].
(7) the post-processing of ranking, format, naming and function that control the output format.
(8) (no icon) the resulting Grapple statements that are reported to the requestor.

![Smart GrappleStatement Conflict Resolution Architecture](image)

**Figure 5** Smart GrappleStatement Conflict Resolution Architecture

### 4 The Design of Grapple Derivation Rules for Conflict Resolution

Grapple Derivation Rule (GDR) [32] is an XML-based rule language used by Grapple User Modelling Framework (GUMF) to reason over and enrich existing Grapple statements about users. It supports reasoning over multiple GUMF dataspaces and uses external knowledge bases (e.g. DBpedia\(^9\), GeoNames\(^{10}\), DBLP\(^{11}\), WordNet\(^{12}\), etc.). The external knowledge base plays an important role as it allows GUMF to reason over Grapple statements that have different levels of semantics, and to translate low-level concepts in the user model into high-level ones.

Note that Grapple Derivation Rules are discussed in detail in deliverable D2.3. This chapter briefly introduces GDR and points out how they can be used for conflict resolution in GRAPPLE.

#### 4.1 Format

In human readable syntax, a GDR rule looks like:

```
antecedent => consequent
```

Antecedent are conjunctions of premises/conditions written \( p_1 \land \ldots \land p_n \) and consequent is specified in term of Grapple statements. Premises are classified into two different types:

1. **Dataspace premises**, which specify conditions in terms of a Grapple statement for user data in GUMF dataspace,
2. **External premises**, which specify conditions in triple patterns for external knowledge bases accessible through SPARQL endpoints.

---

\(^9\) [http://dbpedia.org/](http://dbpedia.org/)

\(^{10}\) [http://geonames.org/](http://geonames.org/)

\(^{11}\) [http://www.informatik.uni-trier.de/~ley/db/](http://www.informatik.uni-trier.de/~ley/db/) and [http://dblp.l3s.de/d2r/](http://dblp.l3s.de/d2r/)

\(^{12}\) [http://wordnet.princeton.edu/](http://wordnet.princeton.edu/)
Variables are indicated using the standard convention of prefixing them with a question mark (e.g., \(?x\)). In XML format, a GDR rule has the form:

\[
<gdr:rule xmlns:gdr="http://www.grapple-project.org/grapple-derivation-rule/"
    xmlns:gc="http://www.grapple-project.org/grapple-core/
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    name="rule name"
    creator="uri of the client application which creates the rule"
    description="a short description about the rule">
    <!—-  dataspace premise definition -->
    <gdr:premise dataspace="dataspace id">
        <gc:subject>subject</gc:subject>
        <gc:predicate rdf:resource="predicate" />
        <gc:object>object</gc:object>
        <gc:level>level</gc:level>
    </gdr:premise>
    
    <!—-  external source premise definition -->
    <gdr:premise source="external source uri">
        <gdr:pattern>a triple pattern</gdr:pattern>
        <gdr:pattern>a triple pattern</gdr:pattern>
        ... ...
    </gdr:premise>
    
    <!—-  consequent definition -->
    <gdr:consequent dataspace="dataspace id in which the rule is defined">
        <gc:subject>another subject</gc:subject>
        <gc:predicate rdf:resource="another predicate" />
        <gc:object>another object</gc:object>
        [<gc:level>another level</gc:level>]
    </gdr:consequent>
</gdr:rule>

4.2 Example

Suppose a Moodle-based client application A1 and a Sakai-based client application A2 is using GUMF. Application A2 subscribes to application A1 dataspace, thus allowing application A2 to query application A1 dataspace. Figure 5 depicts the dataspace of application A1 and application A2 (partial view only).

Example 1: Suppose application A2 wants to derive new Grapple statements describing \( profile:origin \) property of its users using data in application A1 dataspace. Unlike application A1 that maintains the city of origin of the users, application A2 decides to keep the country of origin of the users. For this purpose, application A2 defines a GDR rule as depicted in Figure 5.
### Grapple Statements in Application A1 Dataspace

<table>
<thead>
<tr>
<th>gc:subject</th>
<th>gc:predicate</th>
<th>gc:object</th>
<th>gc:level</th>
</tr>
</thead>
<tbody>
<tr>
<td>user:anna</td>
<td>profile:origin</td>
<td>“Dubai”</td>
<td>1.0</td>
</tr>
<tr>
<td>user:anna</td>
<td>moodle:hasKnowledge</td>
<td>subject:Geography</td>
<td>0.8</td>
</tr>
<tr>
<td>user:bob</td>
<td>profile:origin</td>
<td>“Delft”</td>
<td>1.0</td>
</tr>
<tr>
<td>user:cindy</td>
<td>profile:origin</td>
<td>“Johannesburg”</td>
<td>1.0</td>
</tr>
<tr>
<td>user:cindy</td>
<td>moodle:hasKnowledge</td>
<td>subject:Geography</td>
<td>0.6</td>
</tr>
<tr>
<td>user:donald</td>
<td>profile:origin</td>
<td>“Beijing”</td>
<td>1.0</td>
</tr>
<tr>
<td>user:donald</td>
<td>moodle:hasKnowledge</td>
<td>subject:Geography</td>
<td>0.4</td>
</tr>
</tbody>
</table>

### Grapple Statements in Application A2 Dataspace

<table>
<thead>
<tr>
<th>gc:subject</th>
<th>gc:predicate</th>
<th>gc:object</th>
<th>gc:level</th>
</tr>
</thead>
<tbody>
<tr>
<td>user:anna</td>
<td>sakai:isLearning</td>
<td>subject:Malaysia</td>
<td>0.0</td>
</tr>
<tr>
<td>user:bob</td>
<td>sakai:isLearning</td>
<td>subject:Japan</td>
<td>0.0</td>
</tr>
<tr>
<td>user:cindy</td>
<td>sakai:isLearning</td>
<td>subject:Delft</td>
<td>0.0</td>
</tr>
<tr>
<td>user:donald</td>
<td>sakai:isLearning</td>
<td>subject:Bangkok</td>
<td>0.0</td>
</tr>
<tr>
<td>user:frans</td>
<td>sakai:isLearning</td>
<td>subject:Japan</td>
<td>0.0</td>
</tr>
</tbody>
</table>

### Additional Knowledge in Application A2 Dataspace

| subject:Malaysia | sakai:isRelatedTo | <http://sws.geonames.org/1733045/> |
| subject:Japan    | sakai:isRelatedTo | <http://sws.geonames.org/993960/>  |
| subject:Delft    | sakai:isRelatedTo | <http://sws.geonames.org/2757345/> |
| subject:Bangkok  | sakai:isRelatedTo | <http://sws.geonames.org/2757345/> |

**Figure 6** Applications A1 and A2 Dataspaces
Figure 7 A GDR Rule (Rule 1)

Lines 07-09 are used to get all the users in application A2 dataspace (id=2). The second premise (Lines 10-14) is used to get the profile:origin property of the users stored in application A1 dataspace (id=1). An external knowledge base that maintains geographical information, named GeoNames, is used to derive the country from a city. Lines 15-23 define conditions in triple patterns that are used to retrieve the name of the country from GeoNames given the name of a city. Finally, the consequent derives new Grapple statements describing profile:origin property of the users for application A2.

Note that the details about how a GDR rule is processed by the Grapple Derivation Rule Engine (GDR Engine) will also be presented in deliverable D2.3.

Example 2: Suppose application A2 wants to suggest Wikipedia pages about the subjects the users are currently taking (described using property sakai:isLearning) or the users learned subject:Geography using
application A1 and the level is more than 0.5. Rule 2 depicted in Figure 8 can be used to get the appropriate Wikipedia\textsuperscript{13} pages for the users.

Rule 2 contains two dataspace premises (Lines 07-12 and 13-17) and three external source premises (Lines 18-20, 21-23, and 24-26). Note that the external source premise defined in Lines 18-20 is used to query application A2 dataspace via dataspace SPARQL endpoint. Lines 07-12 are used to get the users who have knowledge level about Geography more than 0.5. The premise in Lines 13-17 retrieves the subjects that are currently being taken by the users. Using the additional knowledge in application A2 dataspace, the subjects are then related to the concepts in GeoNames (Lines 19-20). The external source premise in Lines 21-23 retrieves the DBpedia concepts that are related to GeoNames concepts using owl:sameAs property. The Wikipedia pages described by foaf:page property can then be found in DBpedia knowledge base (Lines 24-26). Finally, the consequent derives new Grapple statements describing sakai:suggestedWikiPage property of the users for application A2.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{image}
\caption{A GDR Rule (Rule 2)}
\end{figure}

\textsuperscript{13} http://www.wikipedia.org/
4.3 GDR and Conflict Resolution Service

One of the important tasks of the Conflict Resolution Service is to detect conflicting statements in repository. The conflicts can be syntactical conflicts as well as semantic conflicts. The Grapple Derivation Rule (GDR) can be used to perform ontological reasoning using external knowledge bases.

Example 1: Application A has a statement about user John as follows.

subject=john  predicate=city  object=Delft

Another application B also has a statement about John:

subject=john  predicate=country  object=Germany

These statements are semantically conflicting with each other. The GDR can be used to assert for example, that Delft is a city in the Netherlands, using GeoNames. The consequences if such contradictory statements occur are not straight forward, as discussed in chapter 3.

Example 2: Application A has a statement about user Hans as follows.

subject=hans  predicate=city  object=Delft

Another application B also has a statement about Hans:

subject=hans  predicate=town  object=Delft

These statements are not semantically conflicting to each other. A GDR rule can be defined to check that city and town are synonyms in WordNet.

Example 3: Application A has a statement about user Mary as follows.

subject=mary  predicate=likes  object=chess

Another application B also has a statement about John:

subject=mary  predicate=likes  object=boardGames

These statements can be semantically not conflicting. Using DBpedia and OpenCyc\(^\text{14}\), a GDR rule is able to determine that chess is a kind of board games (by exploring rdf:type property).

Note that the above examples exemplify the usage of GDR in Conflict Resolution Service. However, there could be some necessary pre- and post-processing when using GDR. At this point additional research still

\(^{14}\text{http://www.opencyc.org/cyc/opencyc}\)
has to be done to find out to what extent GDR can be used for example and experimental analysis has to be done.

5 Documentation on implemented Conflict Resolvers

As pointed out in the previous section, GUMF exploits GDR, the Java-based inference engine of GUMF, to allow for conflict resolution. The following conflict resolvers have been implemented and integrated in the GRAPPLE inference engine.

The usage of such conflict resolution rules have been demonstrated during the international conference on User Modelling, Adaptation, and Personalisation (UMAP) in a Grapple setting where profile information is gathered from different sources [33].

5.1.1 Most recent

For situations where only the n most recent statements should be returned one could use the MAX operator and apply it to the gc:created field of Grapple statements. The gc:created field of a Grapple statement states when a Grapple statement was created.

<table>
<thead>
<tr>
<th>Name</th>
<th>Most Recent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Select the 5 latest statements about a user’s interests (<a href="http://xmlns.com/foaf/0.1/interest">http://xmlns.com/foaf/0.1/interest</a>) where gc:created has the most recent value. Applications (LMS) that would query GUMF about interests then would just receive the 5 most recent interests specified for the user.</td>
</tr>
</tbody>
</table>
5.1.2 Most named

For situations where there might be several statements that characterise a certain property of a user, a strategy to decide on the right statement(s) to be returned is to identify the most popular statements. For example, if there are many statements about a user’s interest into a certain concept A, but only a few statements about her interest in concept B, then – with the most named strategy – one decides to return statements about the user’s interest in concept A.

<table>
<thead>
<tr>
<th>Name</th>
<th>Most Named</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Select the top 3 interests (<a href="http://xmlns.com/foaf/0.1/interest">http://xmlns.com/foaf/0.1/interest</a>) of a user.</td>
</tr>
</tbody>
</table>

5.1.3 Most specific

Another strategy to resolve conflicts is to select the most specific statements. In this case “most specific” may refer to different parts/fields of a Grapple statement. When it refers to the gc:object, i.e. to the value of a predicate, then one approach is to exploit the hierarchy in which the objects are embedded. RDF Schema, for example, allows for the specification of hierarchies via rdfs:subClasOf. With GDR these hierarchies can be exploited to identify the most specific concepts. For example, when specifying the knowledge of a student of certain concepts such as gale:Java, gale:Java_Basics, gale:Java_Threads then statements like (peter, knowledge, gale:Java, 0) and (peter, knowledge, gale:Java_Thread, 100), for which the last item indicates the level, would contradict each other so that it may become useful to just return the most specific knowledge statement.

<table>
<thead>
<tr>
<th>Name</th>
<th>Most Specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Select the most specific knowledge statement (<a href="http://www.grapple-project.org/interest_mostSpecific">http://www.grapple-project.org/interest_mostSpecific</a>) of a user.</td>
</tr>
</tbody>
</table>
project.org/knowledge) about a user by exploiting skos:broader relationships specified in DBpedia. Idea: return only those statements for which there isn’t a more specific concept (specificConcept) than the knowledge concept.

GDR rule

```xml
<gdr:rule
  xmlns:gdr="http://www.grapple-project.org/grapple-derivation-rule/
  xmlns:gc="http://www.grapple-project.org/grapple-core/
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  name="Most specific knowledge"
  description="Select the most specific knowledge statements about the user."
  creator="http://pcwin530.win.tue.nl:8080/grapple-umf/client/1">
  <gdr:premise dataspace="1">
    <gc:subject>?user</gc:subject>
    <gc:predicate rdf:resource="http://www.grapple-project.org/knowledge"/>
    <gc:object>?knowledge</gc:object>
  </gdr:premise>
  <gdr:premise sourceURI="http://dbpedia.org/
  endpoint="http://dbpedia.org/sparql">
    <gdr:pattern>OPTIONAL {?specificConcept
      <http://www.w3.org/2004/02/skos/core#broader>
      ?knowledge}</gdr:pattern>
    <gdr:pattern>
      FILTER(!bound(?specificConcept))
    </gdr:pattern>
  </gdr:premise>
  <gdr:consequent dataspace="1">
    <gc:subject>?user</gc:subject>
    <gc:predicate rdf:resource=" http://www.grapple-project.org/knowledge_mostSpecific"/>
    <gc:object>?knowledge</gc:object>
  </gdr:consequent>
</gdr:rule>
```

5.1.4 Most personal

Sometimes Grapple statements that have been created by the user or by people who know the user very well, are the best solution to query. GDR allows for rules that return the “most personal” statements. The gc:creator predicate of the Grapple statements can be exploited for this.

<table>
<thead>
<tr>
<th>Name</th>
<th>Most Personal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Select the most personal statement about the user’s interest (<a href="http://xmlns.com/foaf/0.1/interest">http://xmlns.com/foaf/0.1/interest</a>). If there is a statement A that was created by the user U herself then A is returned, otherwise if there is not such a statement, but a statement B, which was created by a friend of the U (A foaf:knows U) then statement B is returned. If none of the previous conditions hold then any statement that matches the query is returned. GDR rules model such if-then-else conditions by connecting multiple</td>
</tr>
</tbody>
</table>
rules. The requesting LMS can then send a pattern-based query that specifies “interest_mostPersonal” as a property (pattern).

<table>
<thead>
<tr>
<th>GDR rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 1 (if there exists a statement created by the user):</td>
</tr>
<tr>
<td><a href="">gc:subject</a>?user&lt;/gc:subject&gt;</td>
</tr>
<tr>
<td>&lt;gc:predicate rdf:resource=&quot;<a href="http://xmlns.com/foaf/0.1/interest%22/">http://xmlns.com/foaf/0.1/interest&quot;/</a> &gt;</td>
</tr>
<tr>
<td><a href="">gc:object</a>?interest&lt;/gc:object&gt;</td>
</tr>
<tr>
<td><a href="">gc:creator</a>?user&lt;/gc:creator&gt;</td>
</tr>
<tr>
<td>&lt;/gdr:premise&gt;</td>
</tr>
</tbody>
</table>
|<gdr:consequent dataspace="1">
|<gc:subject>?user</gc:subject> |
|<gc:predicate rdf:resource="http://www.grapple-project.org/interest_mostPersonalUser"/> |
|<gc:object>?interest</gc:object> |
|</gdr:consequent> |
|</gdr:rule> |

Rule 2 (else if there exists a statement created by a friend of the user and none that was created by the user): |
|<gc:subject>?user</gc:subject> |
|<gc:predicate rdf:resource="http://xmlns.com/foaf/0.1/"/> |
|<gc:object>?interest</gc:object> |
|<gc:creator>?friend</gc:creator> |
|</gdr:premise> |
Another approach for returning the most personal statements exploits the trust ratings performed by the user.

<table>
<thead>
<tr>
<th>Name</th>
<th>Most Personal (trust-based)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Select these statements that were created by services or people the user trusts most.</td>
</tr>
</tbody>
</table>

**GDR rule**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<gdr:rule

  xmlns:gdr="http://www.grapple-project.org/grapple-derivation-rule/"
  xmlns:gc="http://www.grapple-project.org/grapple-core/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  name="Most personal knowledge (based on trust ratings)"
  description="Select the most interest statements about the user created by instances the user trusts in."
  creator="http://pcwin530.win.tue.nl:8080/grapple-umf/client/1">

  <gdr:premise
dataspace="1">
    <gc:subject>?user</gc:subject>
    <gc:predicate rdf:resource="http://xmlns.com/foaf/0.1/interest"/>
    <gc:object>?interest</gc:object>
    <gc:creator>?creator</gc:creator>
  </gdr:premise>

  <gdr:premise
dataspace="1">
    <gc:subject>?user</gc:subject>
    <gc:predicate rdf:resource="http://www.grapple-project.org/trusts"/>
    <gc:object>?creator</gc:object>
    <gc:level>?trustLevel</gc:level>
    <gdr:filter>
      MAX(?trustLevel) FOR ?interest LIMIT 1
    </gdr:filter>
  </gdr:premise>

</gdr:rule>
```
5.1.5 Highest Level

The gc:level can be used to quantify the validity of a Grapple statement. If there are many statements that describe a certain property of the user (e.g. name) when there should actually just be one valid statement then a GDR rule can be activated that just selects that statement, which has the highest value for gc:level.

<table>
<thead>
<tr>
<th>Name</th>
<th>Highest Level</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Select this statement about a user’s name (<a href="http://xmlns.com/foaf/0.1/name">http://xmlns.com/foaf/0.1/name</a>) where gc:level has the highest value.</td>
</tr>
</tbody>
</table>

**GDR rule**

```
<gdr:rule
   xmlns:gdr="http://www.grapple-project.org/grapple-derivation-rule/
   xmlns:gc="http://www.grapple-project.org/grapple-core/
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   name="Name property with highest level"
   description="Select this name which has the highest score (level)."
   creator="http://pcwin530.win.tue.nl:8080/grapple-umf/client/1">
  <gdr:premise dataspace="1">
    <gc:subject>?user</gc:subject>
    <gc:predicate rdf:resource="http://xmlns.com/foaf/0.1/name"/>
    <gc:object>?name</gc:object>
    <gc:level>?level</gc:level>
    <gdr:filter>MAX(?level) FOR ?name LIMIT 1</gdr:filter>
  </gdr:premise>
  <gdr:consequent dataspace="1">
    <gc:subject>?user</gc:subject>
    <gc:predicate rdf:resource="http://xmlns.com/foaf/0.1/name"/>
    <gc:object>?name</gc:object>
    <gc:level>?level</gc:level>
  </gdr:consequent>
</gdr:rule>
```
5.1.6 Return All, None, Random

By default, GUMF returns all (temporal and spatial) valid Grapple statements (strategy: all), which allows for external conflict resolution strategies. GDR-based conflict resolution also allows for randomly selecting one of the conflicting Grapple statements by applying the “LIMIT 1” operator (strategy: random). It is possible to return no statement if there are two conflicting Grapple statements.

<table>
<thead>
<tr>
<th>Name</th>
<th>Return None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Return no knowledge statement if there are two knowledge statements (<a href="http://www.grapple-project.org/knowledge">http://www.grapple-project.org/knowledge</a>), which conflict regarding the gc:level attribute. The rule states that for each knowledge statement A and B, which refer to the same user and same knowledge concept, the levels have to be equal.</td>
</tr>
</tbody>
</table>

GDR rule

```xml
<gdr:rule
    xmlns:gdr="http://www.grapple-project.org/grapple-derivation-rule/"
    xmlns:gc="http://www.grapple-project.org/grapple-core/
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    name="None strategy"
    description="Select the no statements if two knowledge statements conflict with each other."
    creator="http://pcwin530.win.tue.nl:8080/grapple-umf/client/1">
    <gdr:premise dataspace="1">
        <gc:subject>?user</gc:subject>
        <gc:predicate rdf:resource="http://www.grapple-project.org/knowledge"/>
        <gc:object>?knowledge</gc:object>
        <gc:level>?levelA</gc:level>
    </gdr:premise>
        <gdr:pattern>
            ?stmt &lt;http://www.grapple-project.org/grapple-core/subject&gt; ?user
        </gdr:pattern>
        <gdr:pattern>
            ?stmt &lt;http://www.grapple-project.org/grapple-core/predicate&gt;
            &lt;http://www.grapple-project.org/knowledge&gt;
        </gdr:pattern>
        <gdr:pattern>
            ?stmt &lt;http://www.grapple-project.org/grapple-core/object&gt; ?knowledge
        </gdr:pattern>
        <gdr:pattern>
            ?stmt &lt;http://www.grapple-project.org/grapple-core/level&gt; ?levelB
        </gdr:pattern>
    </gdr:premise>
</gdr:rule>
```
5.1.7 Return private, expired, replaced
For administration and testing purposes additional strategies exist that allow returning statements that would usually be filtered out and not be returned.

5.1.8 Final Remarks
Due to the service-oriented modular implementation, with the newest semantic web technologies and standards, this list of implemented and documented conflict resolvers can be extended in the future if new scenarios and application domains make new conflict resolvers necessary. However, this initial set already corresponds to the state of the art in the new and promising area of semantic conflict resolution.

The GDR engine and the above conflict resolution rules moreover can be applied to detect conflicts. The premise of a GDR conflict resolution rule, which specifies a conflict pattern, can in combination with the count operator, be used to determine the number of Grapple statements that are conflicting with each other. Given the RSS-based notification mechanism of GUMF, one can listen for the number of conflicts that match the pattern specified in the premise of the GDR rule.

6 Integration of Conflict Resolution into GUMF
The proposed strategies for conflict resolution have been integrated into the Grapple User Modelling Framework (GUMF) that is introduced in Deliverable 6.1a/b and depicted in Figure 9. The different conflict resolution strategies are attached to different components of the framework. The strategies mentioned in Section 3.3 have to be integrated into the “Query Engine” module so that client applications are enabled to specify conflict resolution strategies for each individual query. By contrast, Grapple Derivation Rules (GDR), introduced in Section 4, are part of the GUMF Reasoning Logic.
Derivation rules can be defined for individual dataspaces. Given a query that is sent to a particular dataspace, the dataspace logic ensures that the activated plug-ins and reasoners will be triggered. Conflict resolutions formulated in GDR will be interpreted by the corresponding GDR reasoner, which can also query external knowledge sources to deduce whether some conclusion can be drawn or not. The Grapple statements that match the query (including statements that are deduced by the reasoning logic) are finally ordered and filtered by the Query Engine according to conflict resolution strategies that are directly specified as part of the query (cf. Grapple Query, Deliverable 6.1b).

The GDR reasoning engine is implemented in Java and is hooked into GUMF as a JAR component. The actual conflict resolution rules are stored in a rule repository, which is connected to a database back-end. GDR uses a database back-end as well to store intermediate results inferred during the reasoning process. By default, conflict resolution rules are triggered on querying time. However, given the REST API and the RSS-based notification mechanism of GUMF (see Deliverable 6.1b), it is possible to also listen for conflicts continuously, enabling conflict detection as mentioned above.

The service orientation of this approach can be seen on the left hand side in Figure 9, the Web Service interfaces REST and SOAP integrate into the GUMF controller. It handles the Query Engine module that runs as a service. New is the conflict resolution service that stands in between this query engine, the dataspace logic and the reasoning logic. All components are developed as individual modules.

## 7 Conclusion and Outlook

A design and implementation of modules to resolve conflicting user models has been presented, consisting of conflict detection strategies and conflict resolution strategies. This design is based on a chapter on related research areas like classical conflict resolution systems, semantic web reasoning, as well as a discussion on issues about integrating conflict resolution strategies into GUMF.

The implementation of the designed conflict resolver concepts has fully been realised and the results are applied to the overall GRAPPLE architecture into the GUMF framework. Almost all conflict resolution strategies could be implemented within the specialised GDR Rule Engine. The implicit strategy assumes that the statements are enriched with meta data like the creation time, the creator or the level of detail of the used data structure (of the distributed information units).
Conflicting statements turned out to be a complex and abstract problem with no generic solution that is valid for all situations. In the b) deliverable, the approach as a service into GRAPPLE’s User Model Framework GUMF was implemented and integrated, which also allowed for experimenting with the application of different conflict resolution strategies in different situations.

The conflict resolving strategies have been tested and documented [32,33]. In 2010, already three workshops [34,35,36] have been organised, during which the Grapple principles to user modelling and conflict resolution has been discussed with the user modelling research community. Now and in the future, the application of GRAPPLE derivation rules as the main part of the conflict resolution intelligence has to prove their scalability by the vast usage of applied GRAPPLE instantiations.

References

12. Blass, C.: Conflict resolution strategies for ubiquitous user modeling. Student report, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2004


34. Fabian Abel and Samur Araujo and Nicola Henze and Eelco Herder and Geert-Jan Houben and Erwin Leonardi: Proceedings of the International Workshop on User Data Interoperability in the