INTEGRATING CONTEXT KNOWLEDGE IN USER INTERACTION USING ANSWER SET PROGRAMMING FOR ENHANCING WEB ACCESSIBILITY

Jesia Zakraoui and Wolfgang Zagler
Institute "integrated study", Vienna University of Technology, Vienna, Austria

Keywords: Ontology, Answer set programming, Web accessibility, Context, User interaction.

Abstract: Today, users of web-based platforms are generally heterogeneous and interact via mobile devices for computing, and this is likely to increase in the future. The user interfaces to these platforms can considerably benefit from the semantic of context. They can adapt the utility of interaction styles and display modes depending largely on the surrounding environment, the user's needs and the characteristics of on-line resources. In order to provide means for that, we trace in this work a perspective approach for integrating relevant parts of context in user interaction using semantic web technologies and Answer Set Programming that enhances web accessibility. Furthermore, by specifying constraints on the contextual information, we can determine which types of context are more important than others regarding the user interaction process. This allow us in a semi-automatic manner to consider only the most relevant answer sets in order to automatically adapting the user interface characteristics to users' contextual information.

1 INTRODUCTION

In recent years mobile computing has become very popular as it provides a great opportunity for all people to access and to manipulate on-line resources, especially for people with mobility problems. Some mobile devices offer an efficient way to manipulate the user interface and to personalize the way the interface and the content are presented to the user. One possibility is to offer domain-specific user profiles, which can be selected by the user or an assistant, to personalize and adjust the interface characteristics to the user's needs (Bogacki and Schneider-Hufschmidt, 2002) (e.g. respective age, disabilities) and his/her context information (e.g. noise, temperature, light). However, it requires effort from the user who may not be able to maintain the profile coherently due to disability constraints and/or due to the dynamic change of the context.

At the same time, many user interface improvements based on the concept of *Context-Awareness* (Schilit et al., 1994) and *User Context* (Dey, 2001) were achieved to enable people to use their device in special occasions, e.g. the voice interface in the mobile phone such as SenSay (Siewiorek et al., 2003) or Light-Sensitive Display which allowed users to have display backlight automatically adapted as light conditions changed. In the next phase the notion of context has been extended in the ubiquitous computing (Buttussi, 2008). We now believe that it may enhance the user interaction for e.g. blind people using their mobile devices for E-Learning as well as presenting textual descriptions of images to users who aren't able to download images due to the bandwidth costs, for instance, when they are on mobile devices or when they are in a developing country.

For our purpose, we extend the concept of context with more awareness for environmental context, physical context, computing context and user context in the user interaction process which is collected manually in a Context Ontology (Zakraoui and Zagler, 2010) and we build a rule layer using *Answer set programming (ASP)* over the ontology described with some Semantic Web technologies. The rule layer provides some background knowledge to be able to perform various reasoning tasks. That most of the information consists actually of defaults and that the context ontology contains incomplete knowledge motivated us to use a *nonmonotonic* formalism to build a rule layer over the ontology. We might want to express preferences (e.g. aggregate functions) as well as constraints (e.g. integrity constraints) while querying the knowledge stored in ontologies to be able to discover new knowledge. ASP provides an expres-

Zakraoui J. and Zagler W..
INTEGRATING CONTEXT KNOWLEDGE IN USER INTERACTION USING ANSWER SET PROGRAMMING FOR ENHANCING WEB ACCESSIBILITY.
DOI: 10.5220/0003108304590466
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development* (KEOD-2010), pages 459-466
ISBN: 978-989-8425-29-4

sive language to express these knowledge and efficient solvers, like DLV-HEX (DLV-HEX, 2007) built over DLV (DLV, 2007) to reason about it, this motivated us to use ASP as such a nonmonotonic formalism.
Furthermore, such concept allow us to choose the most relevant answer sets in order to allow the user interface characteristics to change dynamically according to these selected knowledge, so that interfaces are not chosen in advance but rather could be generated in real-time.
The rest of this document is organized as follows: in the next Section we introduce web accessibility together with user interaction. Section 2 gives an overview of Answer Set Programming and explains how it can be used to reason on the context information. In Section 3 we present the way by which we access the content of our Context Ontology and reason on it. Then we discuss in Section 4 shortcomings of some approaches that make use of the semantic web technologies and reasoning. Future work and conclusion are given in Section 5.

2 WEB ACCESSIBILITY

A web application is accessible, if web users with disabilities can perform all the navigational tasks with ease (Baguma and Lubega, 2008). Blind users also rely on audio to perform navigational tasks. From a technical point of view, web accessibility corresponds to making it possible to any user by using some user agent (software or hardware to view web content), to understand and interact with a web site, despite of disabilities, languages or technological constraints. The W3C/WAI (WAI, 2008) Working Group offers standards which are internationally accepted. They offer quantifiable rules, but, web developers often fail to implement them effectively. One of the reasons is that most of the available accessibility guidelines appear to be too costly (Baguma and Lubega, 2008). However, there are many positive arguments for applying these rules: Major arguments are cited in (WAI/BusinessCase, 2009); social responsibility, sustainable technologies, financial benefits and legal liability. Indeed, by dealing with accessibility issues a larger number of people will use the web based platforms and hence one can realize substantial return on investment (ROI).
In fact, web accessibility benefits also older users, whose percentage is increasing significantly, but also mobile device user, and other individuals, since the needs and preferences that are essential to a user are a consequence of having a disability and/or it may be that the circumstances, devices, or other factors have led to a mismatch between them and the resources they wish to use. In this context, the *Adaptive Technology Resource Centre (ATRC)*[1] at the University of Toronto sees disability "as a mismatch between the needs of the individual and the service, education, tools or environment provided and accessibility as the adaptability of the system to the needs of each individual" (Treviranus, 2009).

3 ANSWER SET PROGRAMMING

Answer Set programming (ASP) (Lifschitz, 2008) has emerged as an important tool for declarative knowledge representation and reasoning. This approach is rooted in semantic notions and is based on methods to compute stable models (Gelfond and Lifschitz, 1988). ASP is one of the most prominent and successful semantics for *non-monotonic* logic programs. The specific treatment of default negation under ASP allows for the generation of multiple models for a single program, which in this respect can be seen as the encoding of a problem specification. With ASP, one can encode a problem as a set of rules and the solutions are found by the stable models (Answer sets) of these rules.
An Answer Set Program consists of rules of the form *head :- body* that can contain variables. The head can be a disjunction or empty and the body is a conjunction or empty. A *term* is either a constant or a variable. An *atom* is defined as $p(t_1,...t_k)$ where k is called the arity of p and $t_1,...t_k$ are terms. A *literal* is an atom p or a negated atom $\neg p$, also strong negation (also often referred to as classical negation). A rule without head literal is an *integrity (strong) constraint*. A rule with exactly one head literal is a *normal rule*. If the body of the rule variable-free is empty then this rule is a *fact*. A *negation as failure literal* (or NAF-literal) is a literal l or a default-negated literal *not* l. Negation as failure is an extension to classical negation, it represents default negation in a program, and infers negation of a fact if it is not provable in a program. Thus, *not* l evaluate to true if it cannot be proven that l is true. This is relevant in our work since we don't have complete information about the user interaction process and we must assume some defaults reasoning until we confirm the reasoner with a new knowledge.
In order to solve a problem in ASP, a logic program should be constructed so that its answer sets correspond to the solutions of the problem. By adding new knowledge, not only new answer sets become possi-

[1] http://atrc.utoronto.ca/index.php

ble, but old answer sets are defeated, so that the sets do not grow monotonically. ASP provides us this *non-monotonicity*. An important feature of ASP is that the body of a rule can also contain negation, which is handled as negation as failure, thus allowing methods from non-monotonic reasoning, since additional information might lead to retraction of a previously made inference (Eiter, 2007).
For example, lets say that the frequent user interaction constraint for which we cannot approve that they are meaningful are low priority constraints. If "C" is a frequent constraint and nothing is known about its meaning regarding user interaction, then all the answer sets show that "C" is a low priority constraint.

```
lowpriorityConstraint(C) :-
frequentConstraint(C), not meaningful(C).
```

However, assume that additional information is now available indicating that all the frequent constraints are considered as meaningful in user interaction. Thus, in such a case we are no longer able to expose that "C" is a low priority constraint.

```
meaningful(C) :- frequentConstraint(C).
```

In order to compute these answer sets, there exist ASP solvers or engines such as DLV (DLV, 2007), a highly efficient reasoner for ASP which extends the core language with various sophisticated features such as aggregates or weak constraints (Leone et al., 2006).

3.1 DLV Datalog with Disjunction

DLV is a system for "disjunctive datalog" with constraints, true negation and queries. Disjunctive datalog (Eiter et al., 1997) extends datalog by allowing disjunction, the logical OR expression to appear in the rules. Recall that Datalog programs are function-free logic programs. Disjunctive Datalog is a declarative programming language, which is a finite set of rules over a function-free first-order language (Eiter et al., 1997). For knowledge representation, tries to find the way to solve the problem and the solution itself which is achieved by rules and facts and designed for working with logic and databases, DLV also provides several ways to define indefinite knowledge, by the use of disjunctive rules.
The language of DLV extends disjunctive datalog by another construct, which is called weak constraint (Leone et al., 2006). The DLV system also includes support for negation as failure and classical negation. An important feature is that it supports strong (integrity) and weak constraints, where a constraint is a rule with an empty head, if its body is true, a model is made inconsistent, thus removed. A Disjunctive Datalog program consists of facts, disjunctive rules, and constraints.

3.2 DLV Example

Example 1: Let's say that, in the current user interaction process, the noise constraint is a relevant context value since the user is blind and interacts via a text-to-speech Screen Reader with a web based platform. The used device has a sound volume V less than *maxVolume*. It has the capability to change the interaction mode such as braille display expressed with the fact *alternative(braille)*. The noise has a *deciBel* value and lasts Y time units. For the above scenario, we can use the following predicates:

- adjustWith(S), S the new calculated volume.
- adjustVolume(X,S), the context X has the prior requirements to deal with and S the suggested new volume.
- noiseConstraint(X), the noise constraint X.
- deciBel(X, Y), where Y is the deciBel of the noise constraint X.
- duration(X,Z), denotes the duration Z of the noise constraint X.
- volume(V), denotes the volume V of the device.
- reachLimit(V), denotes the maximum of the tuned volume V when reached.
- suggestAlternative(A, V), where A is an alternative mode due to the maximum volume V.

Let's assume that, the context is relevant when dealing with noise constraints which are based on *deciBel* value, lasts for some time and the *volume* of the used device has a value less than the maximum volume value.
A noise constraint expressed with the head predicate *selectedConstraints(X,S)* is selected if:
$deciBel(X,Y), Y >= 50$ where X is a constraint and Y is the decibel and $Y >= 50$ value and
$duration(X,Z), Z >= 30$ where X is a constraint and Z is the duration value and $Z >= 30$ and
$volume(V), maxVolume(max\_v), V < max\_v$ where V is the volume value and max_v is the maximum volume value and $V < max\_v$ and
$adjustWith(S)$ where S is the new calculated volume $S = V + 10$ and
$not-selectedConstraints(X,S)$ where $-selectedConstraints(X,S)$ fires if:
A noise constaint X and a selected constraint Y are not the same.
Then it is important to take action to overcome this constraint (e.g. adjusting a higher volume if possible otherwise change the interaction mode to an alternative mode if possible). We can represent these facts as follows:

```
% "-" stands for classical negation.
% "not" stands for negation as failure.
% Rules for selecting a part of the context.
% user context in interaction process is given
% by the below requirements:
#const max_v = 100.
#const v = 100.
#maxint=200.

noiseConstraint(publicPlace).
noiseConstraint(speech).
noiseConstraint(streetTraffic).
deciBel(voiceAmplifer,80).
deciBel(streetTraffic,60).
deciBel(publicPlace,50).
deciBel(speech,70).

duration(voiceAmplifer,50).
duration(streetTraffic,30).
duration(publicPlace,60).
duration(speech,100).

maxVolume(max_v). volume(v).

alternative(braille).
-alternative(voiceAmplifier).

adjustWith(S) :- +(v,10,S), S <= max_v.

selectedConstraints(X,S) :- noiseConstraint(X),
  deciBel(X,Y), Y >= 50, duration(X,Z), Z >=30,
  volume(V), maxVolume(max_v),V < max_v,
  adjustWith(S), not -selectedConstraints(X,S).

-selectedConstraints(X,S):- noiseConstraint(X),
               selectedConstraints(Y,S), X !=Y.

reachLimit(V,X) :-  noiseConstraint(X),
 deciBel(X,Y), Y >= 50, duration(X, Z), Z >=30,
 volume(V), maxVolume(max_v),V=max_v.

suggestAlternative(A,V) :- alternative(A),
                           reachLimit(V,X).
```

When the above program is given as input to the DLV system, by using the *-nofacts* (in order to exclude facts as part of the output) and *-pfilter* (in order to output only positive instances of the specified predicates) switch together, the resulting model is:

```
{selectedConstraints(streetTraffic,90)}
{selectedConstraints(speech),90}
{selectedConstraints(publicPlace),90}
```

According to the rules given, a volume is to adjust if there is a noise constraint, the deciBel of the noise is bigger than or equal to 50, and if the noise duration is higher than 30 time units and the maxVolume not yet reached. Therefore, the program has three answer sets. If the maxVolume is reached, the *suggestAlternative(A,X)* predicate fires. Therefore the program has one answer:

```
{suggestAlternative(braille,100)}
```

3.2.1 Using Constraints

Rules can generate different models. The constraints are used to select only the desired ones among possible models. There are two types of constraints namely strong (integrity) constraints (introduced below) and weak constraints. According to (Leone et al., 2006), DLV extends the logic programs by the weak constraints.

Example 2: Considering the same knowledge base as in Example 1, we can also add integrity constraints. If we add the below integrity constraint to our example program, we can discard the noise constraints that take a short duration since we plan to follow the constraint if it restricts the user interaction for a time unit longer than 40 time units.

```
% Integrity constraint
:- selectedConstraints(X,S),
duration(X, Z), Z <= 40.
```

Therefore we have only to adjust the volume due to *publicPlace* and *speech*.

```
{selectedConstraints(speech,90)}
{selectedConstraints(publicPlace,90)}
```

If we add weak constraints to our program in order to obtain the most promising answer set, which gives in fact a ranking by the use of [Weight:Level] couple, where (weight) and (level) are integer constants or variables occurring in the positive body of the rule and all body literal are classical literals. If is not specified, it defaults to 1, and we can just write [w:]. Weak constraints are stated with a " :" and the (optional) specification of level and weight:

```
% Weak Constraints
:~ selectedConstraints(X,S), duration(X,Z). [Z:1]
:~ selectedConstraints(X,S), deciBel(X,Y). [Y:2]
```

When the above program is given as input to the system, by using the *-nofacts*, *-pfilter* and *-wctrace* (to print all models during computation of weak constraints) switch together, the resulting model is:

```
Current model [maybe not optimal]:
{selectedConstraints(speech,90)}
Cost ([Weight:Level]): <[100:1],[70:2]>
Current model [maybe not optimal]:
{selectedConstraints(publicPlace,90)}
Cost ([Weight:Level]): <[60:1],[50:2]>
Best model: {selectedConstraints(publicPlace,90)}
Cost ([Weight:Level]): <[60:1],[50:2]>
```

Thus, using weak constraints, the best model turns out to be "publicPlace" noise constraint which most restrict the user interaction. Consequently, we prioritize the user interaction constraint "publicPlace" to which one has to adjust the volume of the used device to 90 units.

4 INTEGRATING KNOWLEDGE FROM ONTOLOGY

DLV-HEX is a prototype application for computing the models of *hex-programs* (Eiter et al., 2006). Hex-programs originate from dl-programs (Eiter et al., 2006) and they are an extension of answer set programs for the integration of external computation sources that may be in different formats. The answer-set semantics has been extended to Hex-programs, which are *higher-order logic programs* (which accommodate meta-reasoning through higher-order atoms) with external atoms for software interoperability (Eiter et al., 2006).

DLV-HEX involves several plugins such as Description Logic Plugin (DLV-HEX, 2007) which interfaces OWL ontologies by using a reasoner. For instance, we consider in this section as an external knowledge base, our context ontology described in OWL and we can, therefore refer to the ontology and access to the classes and the properties represented in the ontology by the use of dl-atoms. We use DLV for reasoning and RACER (RacerPro, 2005), which is one of the fastest OWL reasoning systems available to query OWL knowledge bases.

The context class has some properties like hasConstraint, hasStartTime, hasDuration, and so on. A representation of a part of this context is given by these instances below.

```
<Context rdf:ID="c1">
 <hasConstraint rdf:resource="#Speech"/>
 <hasConstraint rdf:resource="#StreetTraffic"/>
 <hasDevice rdf:resource="#PC"/>
 </Context>

 <Device rdf:ID="PC">
  <hasStatus>On</hasStatus>
  <hasVolume>80</hasVolume>
  <hasAlternative rdf:resource="#Braille"/>
 </Device>

 <Software rdf:ID="Braille">
 </Software>

 <Noise rdf:ID="Speech">
  <hasDecibel>60</hasDecibel>
  <hasDuration>90</hasDuration>
  <hasStartTime>10:00:00</hasStartTime>
 </Noise>

 <Noise rdf:ID="PublicPlace">
  <hasDecibel>80</hasDecibel>
  <hasDuration>100</hasDuration>
  <hasStartTime>00:00:00</hasStartTime>
 </Noise>

 <Noise rdf:ID="StreetTraffic">
  <hasDecibel>50</hasDecibel>
  <hasDuration>110</hasDuration>
  <hasStartTime>11:00:00</hasStartTime>
 </Noise>
```

Example 3: when a blind user interacts via a Screen Reader with a mobile device. The user is located in a place in which many user interaction constraints originating from different sources like noises. If one constraint needs to be favored over another due to its meaningfulness(e.g. light intensity for a blind user) or to a definite measure (e.g. duration, level) to user interaction, these can be ranked accordingly.

For example, DL[Context] refers to the Context class and DL[hasDuration] refers to the hasDuration datatype property of the Context class in our ontology, according to the above scenario.

```
% "-" stands for classical negation.
% "not" stands for negation as failure.
% Rules for selecting a part of the context.
% user situation in interaction is given by
% the below requirements:
% Using dl-atoms, we refer to our ontology

constraints(X) :- DL[Context](X),
             not -constraints(X).
-constraints(X) :- DL[Context](X),
   constraints(Y), X != Y.

duration(D) :- constraints(X),
  DL[hasDuration](X,D).
deciBel(B) :- constraints(X),
  DL[hasDecibel](X,B).
volume(V) :- constraints(X),
  DL[hasVolume](X,V).
alternative(A) :- constraints(X),
  DL[hasAlternative](X,A).

selectedConstraints(X) :- constraints(X),
 deciBel(Y), Y >= 20, duration(D), D >= 30,
             not -selectedConstraints(X,Y).

-selectedConstraints(X,S) :- constraints(X),
          selectedConstraints(Y,S), X != Y.

reachLimit(V) :- constraints(X),
volume(V), V=100.

suggestAlternative(A,V)  :- alternative(A),
                            reachLimit(V).

% Using a weak constraint for the ordering
:~ duration(D). [D:1]
% Adding another weak constraint in level 2
:~ deciBel(Y). [Y:2]
```

When the above program is given as input to the dlvhex system using weak constraints we allow our program to represent a quantitative cost specification of the constraints and using this feature we can limit the answer sets to the important in user interaction, it returns the following output:

```
{selectedConstraints("context.owl#StreetTraffic>"),
deciBel(50), duration(110)}
Cost ([Weight:Level]): <[110:1],[50:2]>

{selectedConstraints("context.owl#Speech>"),
deciBel(60), duration(50)}
Cost ([Weight:Level]): <[50:1],[60:2]>

{selectedConstraints("context.owl#PublicPlace>"),
deciBel(80), duration(100)}
Cost ([Weight:Level]): <[100:1],[80:2]>
```

Next, with the use of integrity constraint in the program to discard the noise constraints that have a duration below 60 time units:

```
% Checking part:
:- constraints(X), duration(D), D <= 60.
```

Then, the answer sets will be:

```
{selectedConstraints("context.owl#StreetTraffic>"),
deciBel(50), duration(110)}
Cost ([Weight:Level]): <[110:1],[50:2]>

{selectedConstraints("context.owl#PublicPlace>"),
deciBel(80), duration(100)}
Cost ([Weight:Level]): <[100:1],[80:2]>
```

As seen, DLV-HEX returns answer sets in an ordered way with respect to the [Weight:Level] pairs used in weak constraints. In the above output, the least costly model turned to be the one having a duration of 110 time units and 50 deciBel ”StreetTraffic” and the most costly turned out to be the second answer set ”PublicPlace”. Consequently, we prioritize the user interaction constraint ”publicPlace” to which one has to adjust the volume of the used device in order to overcome it.

Although no prototype is available for complete validation yet, preliminary scenarios on a few instances showed that answer sets computed by the DLV-HEX solver provide a significant amount of contextual information. This information can also be easily analysed by interface designer, thus supporting them in deriving hints for improving the interface final design.

5 DISCUSSION

There are generally two kinds of reasoning mechanisms used by the reasoners like *Pellet* and *Racer* and by the *Jena Rule Engine* namely Forward Chaining Rules Execution and Backward Chaining Rules Execution. In the former only the conditional statements in the body are specified in contrary to the latter which is goal oriented.

Shuaib et al. introduced the concept *Connecting ontologies* (Shuaib et al., 2007) for providing universal Accessibility, which links two heterogeneous ontologies or in other words, which describes the linking of heterogeneous entities (concepts, relations and properties) across two ontologies. The *Connecting Rules* are made via Jena Inference API (Reynolds, 2009) which are used to formally describe the connection between the two ontologies. However, due to lack of support for rules or for some concepts (e.g., transitive closure, negation as failure, cardinality constraints), some queries can not be represented concisely and some queries can not be represented at all. In this sense, the ASP-approach provides a more expressive formalism to represent rules, concepts, constraints, and queries than the rule mechanism used in this approach.

Data integration can be formally described using *Horn Clause* rules and *F-Logic* rules (Angele and Gesmann, 2006) with some benefits of expressiveness using *F-logic*. *F-Logic* rules are used to define mappings between ontologies. Since the *Semantic Web Open World Reasoning* does not fit very well with *F-Logic* which is *frame-based*, therefore this involes risks of inconsistence and undecidabiliy.

Sahoo et al. discussed in (Sahoo et al., 2007) the use of the Semantic Web technology *RDF* for integrating two heterogeneous data sources frequently used in genomic studies: phenotype (Entrez Gene) and the genotype (Gene Ontology). The inference rules are implemented based upon *isA* and *partOf* in the RDF store to make explicit the semantics of some predicates.

Another very relevant work is introduced by (Obitko, 2007) about the translation of ontologies in Multi-Agent Systems in the manufacturing domain. The rules are transported via messages and are interpreted in respective agents. In our opinion, when the rules are executed in sequence then the inferred triples are added to the model which are not necessarily being transported or referred. This may not be a requirement in specific manufacturing application but certainly it is an issue if one has to benefit from the open world reasoning provided by ontologies in DL.

Bodenreider et. al (Bodenreider et al., 2008) studied the integration of relevant parts of knowledge extracted from biomedical ontologies, and answering complex queries related to drug safety and discovery, using Semantic Web technologies and Answer Set Programming (ASP). They have illustrated the applicability of this method on some ontologies extracted from existing biomedical ontologies, and its effectiveness by computing an answer to a real-world biomedical query.

Zirtiloğlu et. al have developed a complaints ontology (Zirtiloğlu and Yolum, 2008) with which the

complaints of the citizens can be expressed. Further, they assumed that by using answer set semantics and DLV-HEX as a solver, they can generate more expresive rules than using Semantic Web Rule Language (SWRL) (SWRL, 2004). They applied constraints on the complaints to rank them based on their importance. For our case, this approach can also be investigated and applied.

6 CONCLUSIONS AND FUTURE WORK

We have presented an approach for integrating relevant parts of context in user interaction process using semantic web technologies, Answer Set Programming and DLV-HEX as a solver. This approach allowed us by the use of constraints to limit and to prioritize the set of fired facts. We have achieved thereby an efficient problem reduction, since this approach scales the size of the answer sets and the run time.
We plan to enable the user interface characteristics to change dynamically, according to dynamic change of user characteristics and situations that are detected at run-time from the Context Ontology. The combination of concepts from other ontologies such as User Profile Ontology (UPO) and the Context Ontology are possible. Furthermore, we plan to automatically detect the user's intention.

REFERENCES

Angele, J. and Gesmann, M. (2006). Data integration using semantic technology: A use case. In *RULEML '06: Proceedings of the Second International Conference on Rules and Rule Markup Languages for the Semantic Web*, pages 58–66, Washington, DC, USA. IEEE Computer Society.

Baguma, R. and Lubega, J. T. (2008). A web design framework for improved accessibility for people with disabilities (wdfad). In *W4A '08: Proceedings of the 2008 international cross-disciplinary conference on Web accessibility)*, pages 134–140.

Bodenreider, O., oban, Z. H., Doganay, M. C., and Erdem, E. (2008). A preliminary report on answering complex queries related to drug discovery using answer set programming. In *3rd International Workshop on Applications of Logic Programming to the (Semantic) Web and Web Services (ALPSWS2008)*, pages 85–90.

Bogacki, V. and Schneider-Hufschmidt, D. M. (2002). How to bridge the gap between mass market production and usability for people with special needs. *WWDU*.

Buttussi, F. (2008). A user-adaptive and context-aware architecture for mobile and desktop training applications. In *MobileHCI '08: Proceedings of the 10th international conference on Human computer interaction with mobile devices and services*, pages 543–543, New York, NY, USA. ACM.

Dey, A. K. (2001). Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7.

DLV (2007). http://www.dbai.tuwien.ac.at/proj/dlv/.

DLV-HEX (2007). http://www.kr.tuwien.ac.at/research/systems/dlvhex/.

Eiter, T. (2007). Answer set programming for the semantic web. In *ICLP*, pages 23–26.

Eiter, T., Gottlob, G., and Mannila, H. (1997). Disjunctive datalog. *ACM Trans. Database Syst.*, 22(3):364–418.

Eiter, T., Ianni, G., Schindlauer, R., and Tompits, H. (2006). dlvhex: A tool for semantic-web reasoning under the answer-set semantics. In *Proceedings of International Workshop on Applications of Logic Programming in the Semantic Web and Semantic Web Services*, pages 33–39.

Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., and Scarcello, F. (2006). The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562.

Lifschitz, V. (2008). What is answer set programming? In *AAAI'08: Proceedings of the 23rd national conference on Artificial intelligence*, pages 1594–1597. AAAI Press.

Obitko, M. (2007). Translations between ontologies in multi-agent systems. phd thesis, czech technical university, 2007.

RacerPro (2005). http://www.racer-systems.com.

Reynolds, D. (2009). http://jena.sourceforge.net/inference/.

Sahoo, S. A., Bodenreider, O., Zeng, K., and Sheth, A. (2007). An experiment in integrating large biomedical knowledge resources with rdf: Application to associating genotype and phenotype information. In *Proceedings of the workshop on Health Care and Life Sciences Data Integration for the Semantic Web*. World Wide Web.

Schilit, B., Adams, N., and Want, R. (1994). Context-aware computing applications. In *In Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90. IEEE Computer Society.

Shuaib, K., Khalid, L., and Tjoa, A. M. (2007). Providing universal accessibility using connecting ontologies: A holistic approach. In *HCI (7)*, pages 637–646.

Siewiorek, D., Smailagic, A., Furukawa, J., Krause, A., Moraveji, N., Reiger, K., Shaffer, J., and Wong, F. L. (2003). Sensay: A context-aware mobile phone. In *ISWC '03: Proceedings of the 7th IEEE International Symposium on Wearable Computers*, page 248, Washington, DC, USA. IEEE Computer Society.

SWRL (2004). http://www.w3.org/Submission/SWRL/.

Treviranus, J. (2009). http://atrc.utoronto.ca/index.php?option=com_content&task=view&id=1&Itemid=245.

WAI (2008). http://www.w3.org/WAI/.

WAI/BusinessCase (2009). http://www.w3.org/WAI/bcase/.

Zakraoui, J. and Zagler, W. (2010). An ontology for representing context in user interaction for enhancing web accessibility for all. In *Proceedings of 1st International Conference on eLearning for all (LEAFA2010)*, Tunisia.

Zirtiloğlu, H. and Yolum, P. (2008). Ranking semantic information for e-government: complaints management. In *OBI '08: Proceedings of the first international workshop on Ontology-supported business intelligence*, pages 1–7, New York, NY, USA. ACM.