Reasoning Methods for Personalization on the Semantic Web

Grigoris Antoniou
Institute of Computer Science – FORTH
Heraklion Crete, Greece
antoniou@ics.forth.gr

Matteo Baldoni, Cristina Baroglio, Viviana Patti
Dipartimento di Informatica
Università degli Studi di Torino, Torino, Italy
{baldoni,baroglio,patti }@di.unito.it

Robert Baumgartner, Thomas Eiter, Marcus Herzog, Roman Schindlauer, Hans Tompits
Institut für Informationssysteme
Technische Universität Wien, Austria
{baumgart,herzog}@dbai.tuwien.ac.at, {eiter,roman,tompits}@kr.tuwien.ac.at

François Bry, Sebastian Schaffert
Institut für Informatik
Ludiwg-Maximilians-Universität München, Germany
{francois.bry, sebastian.schaffert}@ifi.lmu.de

Nicola Henze
Information Systems Institute – Semantic Web Group,
University of Hannover, Germany
henze@kbs.uni-hannover.de

Wolfgang May
Institut für Informatik
Universität Göttingen, Germany
may@informatik.uni-goettingen.de

***Abstract*— The Semantic Web vision of a next generation Web, in which machines are enabled to understand the meaning of information in order to better interoperate and better support humans in carrying out their tasks, is very appealing and fosters the imagination of smarter applications that can retrieve, process and present information in enhanced ways. In this vision, a particular attention should be devoted to *personalization*: By bringing the user's needs into the center of interaction processes, personalized Web systems overcome the one-size-fits-all paradigm and provide individually optimized access to Web data and information. In this paper, we provide an overview of recent trends for establishing personalization on the Semantic Web: Based on a discussion on reasoning with rule- and query languages for the Semantic Web, we outline an architecture for service-based personalization, and show results in personalizing Web applications.**

***Index Terms*— semantic web, personalization, reasoning for the semantic web, rule languages, query languages, web data extraction**

I. INTRODUCTION

THe aim of the Semantic Web initiative [27] is to advance the state of the current Web through the use of semantics. More specifically, it proposes to use *semantic annotations* to describe the meaning of certain parts of Web information. For example, the Web site of a hotel could be suitably annotated to distinguish between hotel name, location, category, number of rooms, available services etc. Such meta-data could facilitate the automated processing of the information on the Web site, thus making it accessible to machines and not primarily to human users, as it is the case today.

However, the question arises as to how the semantic annotations of different Web sites can be combined, if everyone uses terminologies of their own. The solution lies in the organization of vocabularies in so-called *ontologies*. References to such shared vocabularies allow

ISSN 1109-9305

interoperability between different Web resources and applications. For example, a geographic ontology could be used to determine that Crete is a Greek island and Heraklion a city on Crete. Such information would be crucial to establish a connection between a requester looking for accommodation on a Greek island, and a hotel advertisement specifying Heraklion as the hotel location.

At the writing time of this paper, there are recommendations by the World Wide Web Consortium for the lower layers of the Semantic Web tower, including the ontology layer of the Semantic Web. The logic layer, residing on top of the semantic languages and ontology languages, is still to shape.

In this paper, we take a certain perspective on reasoning, rule- and query languages for the Semantic Web: We investigate the required expressiveness of reasoning languages for the Semantic Web which foster personalized Web applications. After a brief introduction to the Semantic Web (Section II, we introduce rule languages for the Semantic Web, with particular notion to nonmonotonic rules (Section III). Aspects of evolution, updates and events are discussed in the subsequent section, exemplified by an event-condition-action approach. Reasoning about actions for implementing personalization is described in Section V.

We then turn attention to mechanisms and applications for maintaining effective reasoning and rule-based approaches: For querying and transforming semantic descriptions, we discuss the *Xcerpt* language (Section VI). An approach to automatically generate semantic descriptions by Web data extraction is provided by the *Lixto Suite* (Section VII). The last section finally goes practical and describes the *Personal Reader Framework* for personalization services on the Semantic Web, which integrates ideas from the previous sections. We outline the service-based architecture of the Personal Reader framework, and describe first example Readers for two application domains.

II. Reasoning and the Semantic Web: State of the Art

The development of the Semantic Web proceeds in steps, each step building a layer on top of another. At the bottom layer we find *XML*, a language that lets one write structured Web documents with a user-defined vocabulary. XML is particularly suitable for sending documents across the Web, thus supporting syntactic interoperability. *RDF* [22] is the basic Semantic Web language for writing simple statements about Web objects (called resources and identified uniquely by a URI, a Universal Resource Identifier). Statements are triples composed of a binary predicate linking together two resources; they are logically represented as logical facts $P(x, y)$. *RDF Schema* [30] provides a simple language for writing ontologies. Objects sharing similar characteristics are put together to form *classes*. Examples for classes are hotels, airlines, employees, rooms, excursions etc. Individuals belonging to a class are often referred to as instances of that class. Binary *properties* (such as *works_for*) are used to establish connections between classes. The application of predicates can be restricted through the use of *domain and range restriction*s. For example, we can restrict the property *works_for* to apply only to employees (domain restriction), and to have as value only companies (range restriction).

Classes can be put together in hierarchies through the *subclass relationship*: a class C is a subclass of a class D if every instance of C is also an instance of D. For example, the class of island destinations is a subclass of all destinations: every instance of an island destination (e.g. Crete) is also a destination. The hierarchical organization of classes is important due to the notion of *inheritance*: once a class C has been declared a subclass of D, every known instance of C is *automatically* classified also as instance of D. This has far-reaching implications for matching customer preferences to service offerings. For example, a customer may wish to make holidays on an indonesian island. On the other hand, the hotel Noosa Beach advertises its location to be Bali. It is not necessary (nor is it realistic) for the hotel to add information that it is located in Indonesia and on an island; instead, this information is inferred by the ontology automatically.

But there is a need for more powerful ontology languages that expand RDF Schema and allow the representations of more complex relationships between Web objects. For example, cardinality constraints (every course must be taught by at least one lecturer) and special properties of predicates (e.g. transitivity, symmetry etc.). Ontology languages, such as *OWL* [40], are built on the top of RDF and RDF Schema. For an easy yet comprehensive introduction to the Semantic Web see [5].

So far, reasoning on the Semantic Web is mostly reasoning about knowledge expressed in a particular ontology. This is possible because ontology languages are *formal languages*, which, for example, allow us to reason about:

- *Class membership*: If x is an instance of class C, and C is a subclass of D, then we can infer that x is an instance of D.
- *Equivalence of classes*: If class A is equivalent to class B, and B is equivalent to class C, then we can infer that A is equivalent to C.
- *Consistency*: If we have declared that classes C and D are disjoint, and x is an instance of both C

and D, then there is an error.
- *Classification*: If we have declared that certain property-value pairs are sufficient conditions for membership in class A, then if an individual x satisfies such conditions, we can conclude that x must be an instance of A.

Derivations such as the preceding can be made *mechanically* instead of being made by hand. Such reasoning support is important because it allows one to:

- check the consistency of an ontology and the knowledge,
- check for unintended clashes between classes,
- automatically classify instances of classes.

Automated reasoning support allows one to check many more classes than could be checked manually. Checks like the preceding ones are valuable for designing large ontologies, where multiple authors are involved, and for integrating and sharing ontologies from various sources.

A. Introducing Rules

At present, the highest layer that has reached sufficient maturity is the ontology layer in the form of the description logic-based language OWL [40]. The next step in the development of the Semantic Web will be the logic and proof layers, and rule systems appear to lie in the mainstream of such activities. Moreover, rule systems can also be utilized in ontology languages. So, in general rule systems can play a twofold role in the Semantic Web initiative:

(a) they can serve as extensions of, or alternatives to, description logic-based ontology languages; and
(b) they can be used to develop declarative systems on top of (using) ontologies.

Reasons why rule systems are expected to play a key role in the further development of the Semantic Web include the following:

- Seen as subsets of predicate logic, monotonic rule systems (Horn logic) and description logics are orthogonal; thus they provide additional expressive power to ontology languages.
- Efficient reasoning support exists to support rule languages.
- Rules are well known in practice, and are reasonably well integrated in mainstream information technology, such as knowledge bases, etc.

As an exemplary application, rules can be a natural means for expressing personalization information. For example, the following rules says that "If E is an exercise related to concept C and person X has read the material on C, then E can be presented to X.

$$exercise(E, C), hasRead(X, C) \rightarrow presentExercise(E, X)$$

A more thorough discussion of personalization rules is found in Section VIII-B.

Possible interactions between description logics and monotonic rule systems were studied in [55]. Based on that work and on previous work on hybrid reasoning [69] it appears that the best one can do at present is to take the intersection of the expressive power of Horn logic and description logics; one way to view this intersection is the Horn-definable subset of OWL.

One interesting research thread deals with the exchange of rule sets between applications, making use of Semantic Web languages. Works in this direction include the RuleML initiative [87], based on the XML and RDF languages, and SWRL [61], a recent proposal based on OWL.

A few implementations of rule systems, tailored to reasoning on the Web, exist yet. The most important systems are Mandarax [73] and Triple [91].

III. Nonmonotonic Rules for the Semantic Web

Apart from the classical rules that lead to monotonic logical systems, recently researchers started to study systems capable of handling *conflicts among rules*. Generally speaking, the main sources of such conflicts are:

- Default inheritance within ontologies.
- Ontology merging, where knowledge from different sources is combined.
- Rules with exceptions as a natural representation of business rules.
- Reasoning with incomplete information.

Defeasible reasoning [4] is a simple rule-based approach to reasoning with incomplete and inconsistent information. It can represent facts, rules, and priorities among rules. The main advantage of this approach is the combination of two desirable features: enhanced representational capabilities allowing one to reason with incomplete and contradictory information, coupled with low computational complexity compared to mainstream nonmonotonic reasoning. The main features of this approach are:

- Defeasible logics are rule-based, without disjunction.
- Rules may support conflicting conclusions.
- The logics are skeptical in the sense that conflicting rules do not fire. Thus consistency is preserved.
- Priorities on rules may be used to resolve some conflicts among rules.
- The logics take a pragmatic view and have low computational complexity.

Recent system implementations, capable of reasoning with monotonic rules, nonmonotonic rules, priorities, RDF data and RDF Schema ontologies, are DR- DEVICE [14] and the system in [3].

Answer Set Programs are nonmonotonic logic programs based on the Answer Set Semantics by Gelfond and Lifschitz [49], which use extended logic programs for reasoning and problem solving by considering possible alternative scenarios. Apart from expressing knowledge by facts and disjunctive rules in a declarative way, ASP is capable of handling incomplete information and default knowledge. Furthermore, user preferences and desires can be accommodated using constructs for expressing priorities and weak constraints (i.e., constraints that can be violated at a penalty). Several very efficient implementations of ASP reasoners exist, e.g., smodels [81] and DLV [67], the latter providing frontends for preferences extensions as well as brave and cautious reasoning. These systems offer gradually expressiveness complexity in alignment with the (lower) complexity of syntactic fragments.

With respect to an application in the domain of the Semantic Web, the advantages of ASP are:

- High expressiveness.
- Declarative semantics.
- Model generation in addition to inference

Model generation enables a problem-solving paradigm in which the solutions of a problem instance are declaratively encoded by the models of the logic program.

Using ASP in the context of the Semantic Web has first been proposed by [60]. A recent extension of ASP programs provides an interface to description logic knowledge bases (such as OWL ontologies) [44], [45]; such extended programs, so-called *dl-programs*, may contain queries to the ontology. This formalism allows a flow of knowledge from the ontology to the logic program and back, exploiting the possibilities of handling terminological knowledge in a nonmonotonic application. A prototype implementation via Web-interface is available.

IV. Evolution, Updates and Events

Personalization of the Web heavily depends on dynamic aspects: it is not given a priori, but it is *adaptive* – i.e., *evolving*, and *reacting* upon *events*, e.g., inputs of the user. Furthermore, personalization is often implemented via *reactive* behavior – i.e., by (personalized) rules that specify what to do in a given situation.

In [74], we have discussed generic query (see also above section), update, and event languages for the Semantic Web. Evolution of the Web is a twofold aspect: on today's Web, evolution means mainly evolution of individual Web sites that are updated locally. In contrast, considering the Web and the Semantic Web as a "living organism" that *consists* of autonomous data sources, but that will *show* a global "behavior" leads to a notion of evolution of the Web as *cooperative evolution* of its individual resources. Personalization aspects deal primarily with local evolution (e.g., that a portal site adapts to evolving profiles of its registered users) and local reactivity (reacting on a user's interaction). But, in the "background", the personalization also potentially affects the global communication of the node (e.g., to gather special information that a user requests, or to react on remote events that are relevant to some of its users), and, the more "intelligent" such Web nodes get, the more they need global communication to deal with the requirements of being personalized. Even more, there can be data exchange about personalization aspects (user profiles) between personalized nodes (although, here also non-technical issues, what a node is entitled to tell another about a user, come into play).

In the same way as proposed in [74], for languages for *evolution and reactivity* in the Semantic Web, we recommend to follow a modular approach. The first step is to provide local personalization of a node that is –at the beginning– part of the conventional Web (see e.g., [57]). The next steps then (i) extend the results to local personalization of the Semantic Web (i.e., semantics-based personalization), (ii) enhance personalization to a "semantic" service (i.e., an ontology for personalization), and (iii) then apply Semantic Web reasoning on the personalization level. In addition to the global language aspects sketched above, the internal mechanisms for evolution of the local personalization base, e.g., as evolution of a logic program, are to be considered [2], [42].

When considering evolution of and events on the Web, two aspects must be taken into account: there are "local" updates that apply to a given Web site only; and there are changes in distributed scenarios that must be propagated from one Web site to another. This means, that in addition to local update languages there must be a declarative, semantic framework for generically handling and *communicating* changes (in general, not as explicit updates, but as changes of a situation, described wrt. a combined ontology of the application and of generic events).

During the development of (generic) languages for evolution and reactivity, personalized nodes will seamlessly be integrated with the application scenarios to be developed. In course, reactive functionality will be employed for implementing personalization and adaptivity (as shown below, by integrating suitable sublanguages for (atomic) events and actions into the generic languages). Analogously, personalization and adaptivity will be subject of local and global evolution.

A. Language Paradigm: ECA Rules

According to [74], we propose an approach that is in general based on rules, more specifically, *reactive*

rules according to the *Event-Condition-Action* (ECA) paradigm for the specification of reactivity. An important advantage of them is that the *content* of the communication can be separated from the *generic semantics* of the rules themselves. Cooperative and reactive behavior is then based on events (e.g., an update at a data source where possibly others depend on): If a resource detects a relevant event (either it is delivered explicitly, or it is in some way communicated or detectable on the Web), conditions are checked (either simple data conditions, or e.g. tests if the event is relevant, trustable etc.), which are queries to one or several nodes and are to be expressed in the proposed query language. Finally, an appropriate action is taken (e.g., updating own information accordingly). This action can also be formulated as a transaction whose ACID properties ensure that either all actions in a transaction are performed, or nothing of is done. The actions in course raise again events (either explicit updates, or visible as application-level events).

The focus of the development is on appropriate sublanguages for rules, events, conditions (that are in fact queries) and for the action part of the rules that continue the separation between application-specific *contents* and generic patterns (e.g. for composite events).

a) Events.: An (atomic) event is in general any detectable occurrence on the Web, i.e., (local) system events, incoming messages including queries and answers, transactional events (commit, confirmations etc), updates of data anywhere in the Web, or any occurrences somewhere in an application, that are (possibly) represented in explicit data, or signaled as the event itself. For these *atomic events*, it must also be distinguished between the event itself (carrying application-specific information), and its metadata, like the type of event (update, temporal event, receipt of message, ...), time of occurrence, the time of detection/receipt (e.g., to refuse it, when it had been received too late), the event origin or its generator (if applicable; e.g. in terms of its URI).

Reactive rules often do not specify reactions on atomic events, but use the notion of *composite events*, e.g., "when E_1 happened and then E_2 and E_3, but not E_4 after at least 10 minutes, then do A". Complex events are usually defined in terms of *event algebras*. Thus, a declarative language for describing composite events is required, together with algorithms for handling composite events. This language should not be concerned with what the information contained in the event might be, but only with types of events. For making events themselves part of the Semantic Web, an ontology of composite events has to be defined, together with mappings from and to given event algebras and their implementations.

An important aspect is the integrability of the event meta language, the event contents languages, and the query language. It is desirable that the specification of composite events can be combined with requirements on the state of resources at given intermediate timepoints, e.g. "when at timepoint t_1, a cancellation comes in and somewhere in the past, a reservation request came in in a timepoint when all seats were booked, then, the cancellation is charged with an additional fee". In this case, the composite event handler has to state a query at the moment when a given event arrives. For being capable of describing these situations, a logic (and system) that deals with sequences of events and queries is required. Such approaches have e.g. been presented in *Transaction Logic* [29]; we will also investigate the use of Evolving Logic Programs [1] for this purpose.

So, several integrated languages have to be defined: the surrounding language for composite events, a language for atomic events and their metadata, and languages for expressing the contents of different types of events – e.g., one language based on an ontology for personalization. Note that an ontology for describing data that is relevant to personalization is needed, and a related language for events that are relevant for personalization is required.

b) Events, Knowledge, and Rules.: The view described up to this point is to result in an infrastructure for evolution and reactivity on the Web based on reaction rules that define the behavior of resources upon detection of events. These are in general composite events, based on atomic, application-level ones. Local knowledge is defined by facts, derivation rules, and reaction rules. All of this local knowledge is encoded in XML, and is updatable, in the sense that the update language to be developed must be capable of changing both facts, derivation rules and reactive rules. Here we may rely on the studies done in the context of logic programming about updating derivation and reactive rules [2].

B. Evolution and Reactivity for Personalization

Concepts for personalization and adaptivity will be implemented and supported by the above framework. "Plain" evolving and reactive applications will provide scenarios where personalization is then applied. Expressing personalization by (ECA-) rules is a usual way in today's approaches, which is then extended to a semantic level in various aspects.

For Semantic Web applications, personalization functionality is built upon an ontology-based user model. The ECA rules that implement personalized behavior use –inside the generic languages for the rules and for composite events– sublanguages that combine the user modeling ontology with the respective ontology of application-specific events.

There will prospectively be "typical" rule patterns (i.e., typical structures of composite events that include typical atomic events, and typical action patterns) for expressing personalization issues. These patterns are then "parameterized" by atomic events, special conditions, and actions to yield a certain rule which then belongs to the behavior base of an application. As stated above, this behavior base is also subject to evolution of several kinds:

- reactivity-controlled evolution, which adapts the behavior base according to events on the Web (e.g., when adapting the personal portfolio tracker when a stock to be traced is moved from MDAX to DAX),
- reactivity-controlled evolution, which adapts the behavior base according to changes in the user's profile (e.g., when he is not longer eligible for student tarifs in trains),
- users are enabled to change these rules interactively (via an appropriate graphical interface),
- intelligent evolution by reasoning about the behavior base, etc.

In the context of evolution and reactivity, personalization does not only mean personalized access to the Web –as implemented in today's portals–, but also personalized behavior that is able to raise events. A customer e.g. may have a personalized Web agent for bidding at ebay or for trading stocks. The behavior of such agents is preferably again expressed by ECA rules that can evolve in the same way as described above.

C. Knowledge Base Update and Reasoning About

As pointed out above, the dynamic nature of the desired infrastructure for describing evolution and reactivity on the Web requires not only the capability of updating facts, but also the capability of updating rules. Such updates can be handled in different ways. On the one hand, updates can be performed on an *ad hoc* basis in a static environment, exploiting and adapting methods from the area of knowledge base revision and belief change, see e.g. [48], [92]. On the other hand, and this is for a dynamic environment crucial, updates may occur event-driven. The nature and circumstances of the event which occurred may determine the way in which an entailed update has to be incorporated into the current rule and knowledge base. Here, personalization comes into play since each user might have his or her own view on how this update should materialize. This is supported by user-definable *update policies*, like event-condition-action rules, in which the general change behaviour according to the desires and preferences of the user can be described at a generic level. For instance, the user may define rules which suppress certain unwanted information, or propagate information to parts of the knowledge base which are semantically connected at the meta level.

We envisage a general formal model for expressing different such update approaches, following the method put forth in [43] for capturing different update approaches in the context of (possibly nonmonotonic) knowledge bases. More specifically, such a formal model has different components, taking care of the kind of language, the knowledge base, the change actions, an update policy, etc. under consideration, which can be instantiated in a suitable manner. The accommodation of more general evolving logic programs [1] in it remains to be explored. Moreover, such a formal model provides the basis for defining a temporal logic language for expressing different properties of the evolving knowledge base on top, based on a well-defined semantics. This logical language, in turn, can be used to specify and study general inference and reasoning tasks associated with evolving knowledge and rule bases.

V. Personalization by Reasoning about Actions

Reasoning about action and change is a kind of temporal reasoning where, instead of reasoning about *time* itself, we reason on *phenomena* that take place in time. Indeed, theories of reasoning about action and change describe a *dynamic world* changing because of the execution of actions. Properties characterizing the dynamic world are usually specified by propositions which are called *fluents*. The word *fluent* stresses the fact that the truth value of these propositions depends on time and may vary depending on the changes which occur in the world.

The problem of reasoning about the effects of actions in a dynamically changing world is considered one of the central problems in knowledge representation theory. Different approaches in the literature took different assumptions on the temporal ontology and then they developed different abstraction tools to cope with dynamic worlds. However, most of the formal theories for reasoning about action and change (*action theories*) describe dynamic worlds according to the so-called *state-action model*. In the state-action model the world is described in terms of states and *actions* that cause the transition from a state to another. Typically it is assumed that the world persists in its state unless it is modified by an action's execution that causes the transition to a new state (*persistency assumption*).

The main target of action theories is to use a logical framework to describe the effects of actions on a world where *all* changes are caused by the execution of actions. To be precise, in general, a formal theory for

representing and reasoning about actions allows us to specify:

- *causal laws*, i.e. axioms that describe domain's actions in terms of their *precondition* and *effects* on the fluents;
- action sequences that are executed from the initial state;
- *observations* describing the value of fluents in the *initial state*;
- *observations* describing the value of fluents in later states, i.e after some action's execution.

The term *domain description* is used to refer to a set of propositions that express causal laws, observations of the fluents values in a state and possibly other information for formalizing a specific problem. Given a domain description, the principal reasoning tasks are *temporal projection* (or prediction), *temporal explanation* (or postdiction) and *planning*.

Intuitively, the aim of *temporal projection* is to predict an action's future effects based on even partial knowledge about the current state (reasoning from causes to effect). On the contrary, the target of *temporal explanation* is to infer something on the past states of the world by using knowledge about the current situation. The third reasoning task, planning, is aimed at finding an action sequence that, when executed starting from a given state of the world, produces a new state where certain desired properties hold.

Usually, by varying the reasoning task, a domain description may contain different elements that provide a basis for inferring the new facts. For instance, when the task is to formalize the temporal projection problem, a domain description might contain information on (a), (b) and (c), then the logical framework might provide the inference mechanisms for reconstructing information on (d). Otherwise, when the task is to deal with the planning problem, the domain description will contain the information on (a), (c), (d) and we will try to infer (b), i.e. which action sequence has to be executed on the state described in (c) for achieving a state with the properties described in (d).

An important issue in formalization is known as the *persistency problem*. It concerns the characterization of the invariants of an action, i.e. those aspects of the dynamic world that are not changed by an action. If a certain fluent f representing a fact of the world holds in a certain state and it is not involved by the next execution of an action a, then we would like to have an efficient inference mechanism to conclude that f still hold in the state resulting from a's execution.

Various approaches in the literature can be broadly classified in two categories: those choosing classical logics as the knowledge representation language [63], [76] and those addressing the problem by using non-classical logics [36], [52], [84], [90] or computational logics [11], [13], [50], [72]. Among the various logic-based approaches to reasoning about actions one of the most popular is still the situation calculus, introduced by Mc Carthy and Hayes in the sixties [76] to capture change in first order classical logic. The situation calculus represents the world and its change by a sequence of *situations*. Each situation represents a state of the world and it is obtained from a previous situation by executing an action. Later on, Kowalski and Sergot have developed a different calculus to describe change [63], called *event calculus*, in which *events* producing changes are temporally located and they initiate and terminate action effects. Like the situation calculus, the event calculus is a methodology for encoding actions in first-order predicate logic. However, it was originally developed for reasoning about events and time in a logic-programming setting.

Another approach to reasoning about actions is the one based on the use of modal logics. Modal logics adopts essentially the same ontology as the situation calculus by taking the state of the world as primary and by representing actions as state transitions. In particular, actions are represented in a very natural way by modalities whose semantics is a standard Kripke semantics given in terms of accessibility relations between worlds, while states are represented as sequences of modalities.

Both situation calculus and modal logics influenced the design of logic-based languages for agent programming. Recently the research about situation calculus gained a renewed attention thanks to the cognitive robotic project at University of Toronto, that has lead to the development of a high-level agent programming language, called GOLOG, based on a theory of actions in situation calculus [68]. On the other hand, in DyLOG [12], a modal action theory has been used as a basis for specifying and executing agent behaviour in a logic programming setting, while the language IMPACT is an example of use of deontic logic for specifying agents: the agent's behavior is specified by means of a set of rules (the agent program) which are suitable to specify, by means of deontic modalities, agent policies, that is what actions an agent is obliged to take in a given state, what actions it is permitted to take, and how it chooses which actions to perform.

Let us now show how these concepts can be useful in the Semantic Web, by describing two scenarios where personalization is required. The idea of exploiting reasoning techniques for obtaining adaptation derives from the observation that in many application domains the goal of the user and the interaction occurring with a resource play a fundamental role.

A. Reasoning about Web Services

In the first scenario that we consider, the action metaphor is used for describing (and handling) *Web services*. Generally speaking, a Web service can be seen as any device that can automatically be accessed over the Web. It may alternatively be a software system or a hardware device; a priori no distinction is made. The main difference between a Web service and other devices that are connected to a network stands in the kind of tasks that can be performed: a Web service can be automatically retrieved after a search (that can be thought of as analogous to finding Web pages by means of a search engine, given a set of keywords), it can be automatically invoked, composed with other Web services so to accomplish more complex tasks, it must be possible to monitor its execution, and so on. In order to allow the execution of these tasks, it is necessary to enrich the Web service with a machine-processable description, that contains all the necessary information, such as what the service does, which inputs it requires, which results are returned, and so forth. A lot of research is being carried on in this area and none of the problems that we have just enumerated has met its final solution yet. Nevertheless, there are some proposals, especially due to commercial coalitions, of languages that allow the description of the single services, and their interoperation. In this line, the most successful are WSDL [93] and BPEL4WS. This initiative is mainly carried on by the commercial world, with the aim of standardizing registration, look-up mechanisms and interoperability,

Among the other proposals, OWL-S [82] (formerly DAML-S [38]) is more concerned with providing greater expressiveness to service description in a way that can be *reasoned about* [34]. In particular, a service description has three conceptual levels: the *profile*, used for advertising and discovery, the *process model*, that describes how a service works, and the *grounding*, that describes how an agent can access the service. In particular, the process model describes a service as atomic, simple or composite in a way inspired by the language GOLOG and its extensions [51], [68], [77]. In this perspective, a wide variety of agent technologies based upon the *action metaphor* can be used. In fact, we can view a service as an action (atomic or complex) with preconditions and effects, that modifies the state of the world and the state of agents that work in the world. The process model can, then, be viewed as the description of such an action; therefore, it is possible to design agents, which apply techniques for reasoning about actions and change to Web service process models for producing new, composite, and customized services.

Quoting McIlraith [78]: "[...] *Our vision is that agents will exploit user's constraints and preferences to help customize user's requests for automatic Web service discovery, execution, or composition and interoperation* [...]". In different words, personalization is seen as *reasoning* about the user's constraints and preferences and about the *effects*, on the user's knowledge and on the world, of the *action* "interact with a Web service". Techniques for reasoning about actions and change are applied to produce composite and customized services.

We claim that a better personalization can be achieved by allowing agents to reason also about the *conversation protocols* followed by Web services. Conversation protocols rule the interactions of a service with its interlocutors: the protocol defines all the possible "conversations" that the service can enact. Roughly speaking, we can consider it as a procedure built upon atomic speech acts. So far, however, OWL-S does not represent in a way that can be reasoned about, the communicative behaviour of a service. Let us explain with a simple example how this would be useful: an agent, which is a user's *personal assistant*, is requested to book a ticket at a cinema where they show a certain movie; as a further constraint, the agent does not have to use the user's credit card number along the transaction. While the first is the *user's goal*, the additional request constrains the way in which the agent will *interact* with the service. In this case, in order to personalize the interaction according to the user's request, it is indeed necessary to reason about the service communications.

In [7] a Web service is supposed to follow some (possibly non-deterministic) procedure for interacting with other services/agents. The authors show that by reasoning on the (explicitly given) conversation protocols followed by Web services, it is possible to achieve a *better personalization* of the service fruition. More recently, the same authors have shown that the same kind of reasoning can be exploited for *composing* a set of Web services, which must interoperate in order to accomplish a complex task, that none of them can execute by itself alone. Consider, as an example, the organization of a journey: it is necessary to find and make work together services for finding a flight, renting a car, making a reservation at some hotel, maybe the user's personal calendar, etc. All services that have been developed independently and for simpler purposes. The problem of describing and reasoning about conversation protocols is faced in an *agent logic programming* setting, by exploiting the reasoning capabilities of agents written in the DyLOG language, introduced in [12]. In particular, integrated in the language, a communication kit [8], [83] allows reasoning about the possible interactions ruled by a protocol by answering to existential queries of the kind: is there a possible execution of the protocol, after which a set of beliefs of interest (or goal)

will be true in the agent's mental state?

B. Reasoning about Learning Resources

The second scenario is set in an e-learning framework: a system has to manage a repository of learning resources, helping users to retrieve the documentation that they need, for acquiring some desired expertise. The goal of the system is returning a *personalized reading sequence* through a (sub)set of the available resources, that will allow the specific user to reach his/her learning goal. Notice that resources may be of different kinds, e.g. text, examples, tests, programming patterns, references to books, and so forth.

The same learning object can be used in different reading sequences, maybe aimed at different learning goals. Moreover, a sequence might contain learning objects that are physically located in different repositories.

Based on the experience gained in previous work [9], [10], an approach is to carry on the construction of reading sequences by means of techniques for reasoning about actions, like planning and temporal explanation, applying them to semantically annotated learning resources. Indeed, also in this scenario the adoption of the "action metaphor" is quite straightforward: a *learning resource* can, in fact, be considered as an *action*, with preconditions (what the student should know for understanding the knowledge contents) and effects (what the student is supposed to learn by reading the resource) on the knowledge of the reader. This choice is also supported by research in pedagogy that shows that human learning is goal-driven, and the notions of prerequisite and effect (in our case, knowledge gain) play a fundamental role. In the action-based representation of learning resources, prerequisites and effects are supposed to be expressed by means of "knowledge entities", i.e. terms from a reference ontology.

In this scenario the goal of personalization is to produce reading sequences that fit the specific user's characteristics (i.e. users with different initial knowledge will be suggested different solutions) and the user's learning goal. Notice that, differently than what happens in other approaches, adaptation occurs at the level of the *reading sequence* rather than at the level of page contents (no link hiding or semaphore annotation is supposed to be used), and it is done w.r.t. the user's *learning goal*.

Many reasoning techniques can be applied in this scenario. One way for building personalized reading sequences is to apply planning techniques; on the other hand, temporal explanation can be used to motivate the user to read documents, that apparently have no direct relation to the learning goal. Also techniques for dealing with *failure* and *replanning* are useful: failure occurs when a user is not satisfied of the proposed solution, on the whole or of part of it, and the system is asked to find alternatives. *Non-monotonic* reasoning techniques could help in this case.

In the literature, it is possible to find programming languages based on action logics (like DyLOG and GOLOG) that support some of the mentioned reasoning techniques and many others. For instance, in DyLOG it is possible to exploit a kind of planning, known as *procedural planning*, that rather than combining in all the possible ways the available actions (documents, or resources) searches for solutions in a restricted space, consisting of the set of possible executions of a given procedure. In this case the procedure describes the general schema of the solution to be found, which is kept separate from the specific resources. At planning time, depending on the initial situation and on the available resources, a solution will be built. The use of procedures as schemas allows the achievement of a form of personalization that not only depends on the user's characteristics and goal (whose description is contained in the initial state) but it also depends on preferences given by the providers of the resources. In the scenario in issue, the procedure would correspond to a *learning strategy* described by the lecturer of the course, which takes into account the experience of the teacher and his/her preferences on how the topic should be thought.

VI. Xcerpt: A Query and Transformation Language for Web and Semantic Web Applications

Querying the Web, i.e. retrieving Web and Semantic Web data using queries expressed in a high level language, can considerably ease the realization of personalized information systems on the Web. Doing this using a query language capable of deduction can further simplify conceiving and implementing personalized information systems on the Web.

Xcerpt [24], [89], [94] is an experimental deductive query language developed at the Institute for Informatics of the University of Munich since 2001.

The goal of the Xcerpt project is to investigate ways to ease realizing Web as well as Semantic Web applications, in particular realizing personalized information systems on the Web. One might see the Semantic Web meta-data added to today's Web as semantic indexes similar to encyclopedias. A considerable advantage over conventional encyclopedias printed on paper is that the relationships expressed by Web meta-data can be followed by computers, very much like hyperlinks can be followed by programs, and be used for drawing conclusion using automated reasoning methods:

> For the Semantic Web to function, computers must have access to structured collections of information and sets of inference rules that

they can use to conduct automated reasoning. [28]

A central principle of the Web query language Xcerpt presented in this section is that a common query language capable of inference to be used for querying both the conventional Web and the Semantic Web is desirable and possible. This working hypothesis is one of the salient features of Xcerpt which makes it different from all Web as well as Semantic Web query languages developed so far.

1) Xcerpt's Principles:

a) Referential Transparency.: Referential transparency means that, within a definition scope, all occurrences of an expression have the same value, i.e. denote the same data. Referentially transparent programs are easier to understand and therefore easier to develop, to maintain, and to optimize. Referential transparency surely is one of the essential properties a query language for the Web should satisfy.

b) Answer-Closedness.: We call "answer-closed" a query language such that replacing a subquery in a compound query by possible (not necessarily actual) answers always yields a syntactically valid query. Answer-closed query languages ensure in particular that every data item, i.e. every possible answer to some query, is a syntactically valid query. Functional programs can – but need not – be answer-closed. Answer-closedness eases the specification of queries because it keeps limited the unavoidable shift in syntax from the data sought for, i.e. the expected answer, and the query specifying these data. Xcerpt is answer-closed.

c) Answers as Arbitrary XML Data.: XML is the lingua franca of data interchange on the Web. As a consequence, answers should be expressible as every possible XML application. This includes both text without markup and freely chosen markup and structure. This requirement is obvious and widely accepted for conventional Web query languages but it is not enforced by many Semantic Web query languages.

d) Answer Ranking and Top-k Answers.: It is often desirable to rank answers according to some application-dependent criteria. It is desirable that Web and Semantic Web query languages offer (a) basic means for specifying ranking criteria and, for efficiency reasons, (b) evaluation methods computing only the top-k answers (i.e. a given number k of best-ranked answers according to a user-specified ranking criterium). Xcerpt supports the specification of orders on XML documents and the retrieval of k answers of a query, possibly sorted according to a specified order.

e) Pattern Queries.: Xcerpt uses patterns for binding variables in query expressions instead of path expressions – like e.g. the Web query languages XQuery and XSLT. Query patterns are especially well-suited for a visual language because queries have a structure very close to that of possible answers.

f) Incomplete Query Specifications.: Incomplete queries specifying only part of data to retrieve, e.g. only some of the children of an XML element (referring to the tree representation of XML data called "incompleteness in breadth") or an element at unspecified nesting depth (referring to the tree representation of XML data called "incompleteness in depth"), are important on the conventional Web because of its heterogeneity: one often knows part of the structure of the XML documents to retrieve. For similar reasons, incomplete queries are important on the Semantic Web. Xcerpt supports queries that are incomplete in breadth, in depth, with respect to the element order, and because of optional elements or attributes.

g) Incomplete Data Selections.: Because Web data are heterogeneous in their structures, one is often interested in "incomplete answers". Two kinds of incomplete answers can be considered. First, one might not be interested in some of the children of an XML (sub)document retrieved by a query. Second, one might be interested in some child elements if they are available, but would accept answers without such elements. Xcerpt's construct `except` gives rise to discard a child of an element retrieved by a query, i.e. to express queries of the first kind. Xcerpt's construct `optional` gives rise to select elements only if available, i.e. to express queries of the second kind.

h) Rule-Based, Chaining, and Recursion.: Rules are understood here as means to specify novel, maybe virtual data in terms of queries, i.e. what is called "views" in (relational) databases, regardless of whether this data is materialized or not. Views, i.e. rule-defined data are desirable for both conventional and Semantic Web applications. Xcerpt supports (unrestricted) recursion on possibly cyclic data (relying on a so-called "memorization" or "tabulation" technique).

i) Separation of Queries and Constructions.: Two standard and symmetrical approaches are widespread, as far as query and programming languages for the Web are concerned:

- queries or programs are embedded in a Web page or Web page skeleton giving the structure of answers or data returned by calls to the programs
- parts of a Web page specifying the structure of the data returned to a query or program evaluation are embedded in the queries or programs.

It is a thesis of the Xcerpt project that both approaches to queries or programs are hard to read (and, therefore, to write and to maintain). Instead of either approach, Xcerpt strictly separates queries and "constructions", i.e. expressions specifying the structure of answers. With Xcerpt, constructions are rule heads and queries

are rule bodies. In order to relate a rule's construction, i.e. the rule's head, to a rule's query, i.e. the rule's body, Xcerpt uses (logic programming) variables.

j) A Query Language for both the Standard Web and the Semantic Web.: A thesis underlying the Xcerpt project is that a common query language for both conventional Web and Semantic Web applications is desirable.

k) Specific Reasoning as Theories.: Many practical applications require special forms of reasoning. For this reason, it is desirable that a query language for the (conventional and Semantic) Web can be extended with so-called "theories" implementing specific forms of reasoning.

l) Two Syntaxes: XML Syntax and Compact Human Readable Syntax.: While it is desirable that a query language for the (conventional and/or Semantic) Web has an XML syntax, because it makes it easier to exchange query programs on the Web and to manipulate them using the query language, a second, more compact syntax easier for human being to read and write is desirable.

2) Flavors of Xcerpt: Xcerpt's Core Constructs: An Xcerpt program consists of at least one goal and of some (possibly zero) rules. Goals and rules are built from data, query, and construct terms representing respectively XML documents, query, and XML documents constructed from the answers to queries.

Data, query, and construct terms represent tree-like (or graph-like) structures. In data, query, and construct terms, square brackets (i.e. [ ]) denote ordered term specification (as in standard XML), i.e. the matching subterms in the queried resource are required to be in the same order as in the query term. Curly braces (i.e. { }) denote unordered term specification (as is common in databases), i.e. the matching subterms in the queried resource may be in arbitrary order. Single (square or curly) braces (i.e. [ ] and { }) denote that a matching term must contain matching subterms for all subterms of a term and may not contain additional subterms (total term specification). Double braces (i.e. [[ ]] and {{ }}) denote that the data term may contain additional subterms as long as matching partners for all subterms of the query term are found (partial term specification).

Non-tree graph structures are expressed using references, i.e. symbolic addresses: The construct `id @ t` is a defining occurrence of the identifier `id` as reference handle of a term `t` and the construct `^id` is a referring occurrence.

a) Data terms: represent XML documents (we speak of "XML in disguise"). They are similar to ground functional programming expressions and logical atoms. Data terms may only contain single square and curly braces, but no double braces expressing partial specifications, as an XML document is complete.

The data term in Figure 1 is the shortened representation of an article in Xcerpt syntax. Note that some parts of the article use unordered term specification (e.g. the author entries) since the order is irrelevant.

b) Query terms: are partial patterns that are matched with data terms, augmented by an arbitrary number of variables for selecting data items from a data term. In addition to the constructs used in data terms, query terms have the following additional properties:

1) partial specifications omitting subterms irrelevant to the query are possible (indicated by double square brackets [[ ]] or curly braces {{ }}),
2) it is possible to specify subterms at arbitrary depth using the construct `desc`),
3) query terms may contain term variables and label variables to "select" data.

In the following examples, upper case characters are chosen for variables. The Xcerpt construct `X -> t` (read "`X` as `t`") associates a variable to a query term, so as to specify a restriction of its bindings. The Xcerpt construct `desc` (read "descendant") is used to specify subterms at arbitrary depth. Suppose that the articles of the proceedings of a conference are contained in a `proceedings` element. The following query term selects title and author pairs for each article:

```
proceedings {{
        article {{
            var T -> title {{ }},
            var A -> author {{ }}
        }}
    }}
```

Query terms (in general containing variables) are unified with data or construct terms (in which variables may occur) using a non-standard unification expecially conceived for Xcerpt and called simulation unification [33]. Simulation unification is based on "graph simulation", a relation similar to graph homomorphisms.

The result of unifying a query term with a data term (construct term, resp.) is a set of substitutions for the variables in the query term (in the query term and construct term, resp.), where each substitution represents an alternative solution.

c) Construct terms: serve to reassemble variables (the bindings of which are specified in query terms) so as to construct new data terms. They may only contain single brackets (i.e. [ ] or { }) and variables, but no partial specification (i.e. no double braces [[ ]] or {{ }}) or variable restrictions (i.e. `x -> t`). The rationale of this is to keep variable specifications within query terms, ensuring a strict separation of purposes between query and construct terms. The following construct term creates an Author-Title pair wrapped in a "`result`" element:

Fig. 1. Representation of an article in Xcerpt syntax

```
paper [
        front [
           title [ "Reasoning Methods for Personalization
                     on the Smantic Web " ],
           author {
              fname [ "Grigoris" ],
              surname [ "Antoniou" ],
              address { ... },
              bio [ ... ]
           },
           author {
              fname [ "Nicola" ],
              surname [ "Henze" ],
              address { ... },
              bio [ ... ]
           },
        ],
        body [
           section [
              title [ "Introduction" ], ],
              ...
        ],
        rear [
            acknowl [ ... ],
            bibliog {
                 bibitem [
                    bib [ "XQuery" ],
                    pub [ "XQuery: The XML Query Language ..." ]
                 ],
                 ...
            }
        ]
    ]
```

```
result {
         var A, var T
    }
```

In a construct term, the Xcerpt construct `all t` serves to collect (in the construct term) all instances of `t` that can be generated by alternative substitutions for the variables in `t` (returned by the associated query terms in which they occur). Likewise, `some n t` serves to collect at most `n` instances of `t` that can be generated in the same manner. Referring to the previous query, the following construct term creates a list of publications for each author:

```
results {
          result {
             var A,
             all var T
          }
      }
```

Referring again to the previous query, the following construct term collects all titles for each author:

```
     results {
          all result { var A, all var T }
     }
```

Referring again to the previous query, the following construct term collects all titles for each author:

```
results {
    all result { all var A, var T }
}
```

d) Queries: Query terms are (atomic) *queries*. Query terms can be “and” or “or”-connected yielding (complex) *queries*. A query is always (implicitly or explicitly) associated with a resource, i.e. the program itself, an external Xcerpt program or an (XML or other) document specified by a URI (uniform resource identifier). All occurrences of a variable in a query term and in and-connected queries are always evaluated identically: this is the usual approach to variable binding in the database query language SQL and in logic programming. The query in Figure 2 selects all authors that have published an article in the proceedings of the 2003 and 2004 venues of a conference (it is assumed that the articles are contained in a `proceedings03` resp. `proceedings04` element):

e) Construct-query rules and goals.: An Xcerpt program consists of zero or more *construct-query rules*, one or more *goals* and zero or more data terms. In particular, an XML document, i.e. a data term, is an Xcerpt program. Rules and goals have the forms:

Fig. 2. An example query

```
and {
    in { resource { "file:proceedings03.xml" },
        desc author {{
            fname { var First }, surname { var Last }
        }}
    },
    in { resource { "file:proceedings04.xml" },
        desc author {{ fname { var First }, surname { var Last }
        }}
    }
}
```

```
CONSTRUCT construct term
FROM query
END

GOAL construct term
FROM query
END
```

where a `construct term` is constructed depending on the evaluation of a `query`, i.e. shared variables.

f) Further constructs.: Besides the core constructs presented above, Xcerpt has so-called "advanced constructs". These constructs give rise to expressing (1) functions and aggregations (such as count, average, etc.), (2) that part of a query is "optional", i.e. to be retrieved only if present in the data considered, (3) to express positions of subterms searched for, and (4) negation in queries. Xcerpt's advanced constructs are detailed in [89].

3) Languages Related to Xcerpt: Two companion languages of Xcerpt deserve to be mentioned: visXcerpt and XChange. visXcerpt [23], [25] is a visual language based on the same principles as the textual language presented above. XChange is a reactive language based on Xcerpt for expressing updates and exchanging events on the Web [31], [32].

VII. Web Data Extraction

If, on a hand, today the Semantic Web [27] is still a vision, on the other, the *unstructured Web* already contains millions of documents which are not queryable as a database and heavily mix layout and structure. Moreover, they are not annotated at all. There is a huge gap between Web information and the qualified, structured data as usually required in corporate information systems. According to the vision of the Semantic Web, all information available on the Web will be suitably structured, annotated, and qualified in the future. However, until this goal is reached, and also, towards a faster achievement of this goal, it is absolutely necessary to (semi-)automatically extract relevant data from HTML documents and automatically translate this data into a structured format, e.g., XML. Once transformed, data can be used by applications, stored into databases or populate ontologies.

Whereas information retrieval targets to analyze and categorize documents, information extraction collects and structures entities inside of documents. For Web information extraction languages and tools for accessing, extracting, transforming, and syndicating the Data on the Web are required. The Web should be useful not merely for human consumption but additionally for machine communication. A program that automatically extracts data and transforms it into another format or markups the content with semantic information is usually referred to as *wrapper*. Wrappers bridge the gap between unstructured information on the Web and structured databases.

A number of classification taxonomies for wrapper development languages and environments have been introduced in various survey papers [47], [64], [66].

High-level languages have been developed for Web extraction. These *stand-alone wrapper programming languages* include *Florid* [75], *Jedi* [62], *Tsimmis* and *Araneus* [6]. In general, all manual wrapper generation languages are difficult to use by laypersons.

Machine learning approaches generally rely on learning from examples and counterexamples of a large number of Web pages (*Stalker* [80], Davulcu et al. [39], *Wien* [65]). The *RoadRunner* [37] approach does not need labelled examples, but derives rules from a number of given pages by distinguishing the structure and the content. It uses an interesting generation of pattern names based on offset-criteria in addition to the applied semi-structured wrapping technology. Some approaches such as [46] offer generic wrapping techniques. Such approaches have the advantage that they can wrap arbitrary Web pages never seen before, on the other hand the disadvantage that they are restricted to particular domains (such as detecting addresses).

Interactive approaches allow for semi-automatic extraction generation and offer convenient visual dialogues to generate a wrapper based on a few examples and user interaction. *Supervised interactive wrapper*

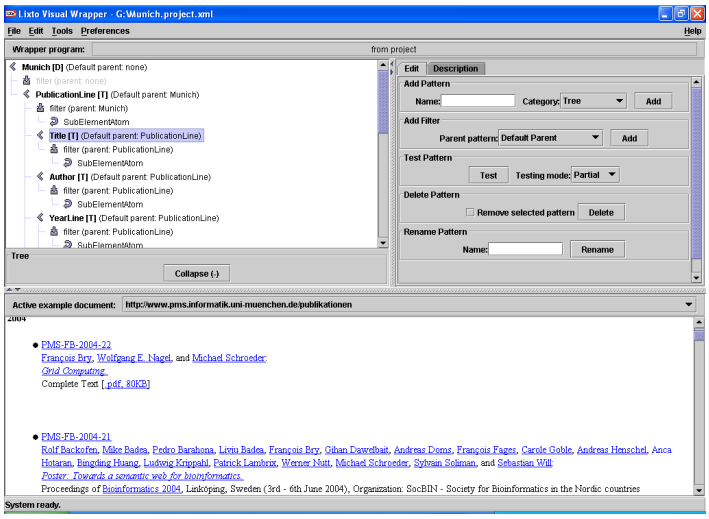

Fig. 3. Lixto Visual Wrapper: Wrapping Publication Pages

generation tools include *W4F* [88], *XWrap* [70], *Wiccap* [71], *SGWrap* [79], and *Wargo* [85] and *DEByE* [86]. In general, many systems neglect the capabilities of Deep Web navigation such as form filling; however, in practice this is highly required, as most information is hidden somewhere in the Deep Web [26].

A. Lixto

Lixto [17] is a methodology and tool for visual and interactive wrapper generation developed at the University of Technology in Vienna. It allows wrapper designers to create so-called "XML companions" to HTML pages in a supervised way. As internal language, Lixto relies on *Elog*. Elog is a Datalog-like language especially designed for wrapper generation. Examples of programs in Elog are given in [16]. The Elog language operates on Web objects, that are HTML elements, lists of HTML elements, and strings. Elog rules can be specified fully visually without knowledge of the Elog language. Web objects can be identified based on internal, contextual, and range conditions and are extracted as so-called "pattern instances".

In [53], [54], the expressive power of a kernel fragment of *Elog* has been studied, and it has been shown that this fragment captures monadic second order logic, hence is very expressive while at the same time easy to use due to visual specification.

Besides expressiveness of a wrapping language, robustness is one of the most important criteria. Information on frequently changing Web pages needs to be correctly discovered, even if e.g. a banner is introduced.

Visual Wrapper offers robust mechanisms of data extraction based on the two paradigms of tree and string extraction. Moreover, it is possible to navigate to further documents during the wrapping process. Predefined concepts such such as "is a weekday" and "is a city" can be used. The latter is established by connecting to an ontological database. Validation alerts can be imposed that give warnings in case user-defined criteria are no longer satisfied on a page.

Visually, the process of wrapping is comprised of two steps: First, the identification phase, where relevant fragments of Web pages are extracted (see Figure 3). Such extraction rules are semi-automatically and visually specified by a wrapper designer in an iterative approach. This step is succeeded by the structuring phase, where the extracted data is mapped to some destination format, e.g. enriching it with XML tags. With respect to populating ontologies with Web data instances, another phase is required: Each information unit needs to be put into relation with other pieces of information.

B. Visual Data Processing with Lixto

Heterogeneous environments such as integration and mediation systems require a conceptual information flow model. The usual setting for the creation of services based on Web wrappers is that information is obtained from multiple wrapped sources and has to be integrated; often source sites have to be monitored for changes, and changed information has to be automatically extracted and processed. Thus, push-based information systems architectures in which wrappers are connected to pipelines of postprocessors and integration engines which process streams of data are a natural scenario, which is supported by the Lixto

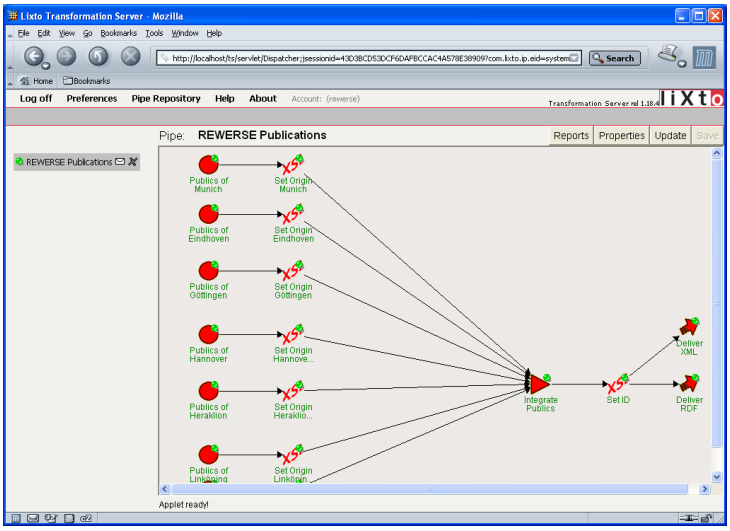

Fig. 4. Lixto Transformation Server: REWERSE Publication Data Flow

Transformation Server [21], [59]. The overall task of information processing is composed into stages that can be used as building blocks for assembling an information processing pipeline. The stages are to

- acquire the required content from the source locations; this component resembles the Lixto Visual Wrapper plus Deep Web Navigation;
- integrate and transform content from a number of input channels and tasks such as finding differences, and
- format and deliver results in various formats and channels and connectivity to other systems.

The actual data flow within the Transformation Server is realized by handing over XML documents. Each stage within the Transformation Server accepts XML documents (except for the wrapper component, which accepts HTML), performs its specific task (most components support visual generation of mappings), and produces an XML document as result. This result is put to the successor components. Boundary components have the ability to activate themselves according to a user-specified strategy and trigger the information processing on behalf of the user. From an architectural point of view, the Lixto Transformation Server may be conceived as a container-like environment of visually configured information agents. The pipe flow can model very complex unidirectional information flows (see Figure 4). Information services may be controlled and customized from outside of the server environment by various types of communication media such as Web services.

C. Web Data Extraction Application Domains

Better software connections are a key challenge to rapid progress in collaborative and e-commerce applications. Rather than waiting for suppliers to recode entire applications to Web service and Semantic Web standards, one can choose the route to better Web connectivity, using today's existing systems. Extraction technologies help to unfold the structure of the desired pieces of information from HTML documents and translate it into XML in a very cost-effective way.

With Lixto some functions that will be tangible only in the future Semantic Web are already turning into reality today. Lixto applications collect data, transform the information into a homogeneous structure and syndicate the semantically enriched data to applications or devices. Lixto's advantages in respect to other wrapper tools and screen-scrapers are its high flexibility, robustness, expressiveness, usability, and its ability to provide interfaces to various data formats and delivery channels [19], [20].

The application domains of extraction technologies are manifold. They e.g. include automatizing portal-based interactions between automotive suppliers, repackaging content for mobile devices, monitoring e.g. price and news data for business intelligence frameworks, and updating address data for CRM databases [15], [18]. Moreover, Web data harvested and syndicated by Lixto can be ideally used by personalization systems, e.g. to offer personalized views on extracted news or publications, as described in Section VIII-D.

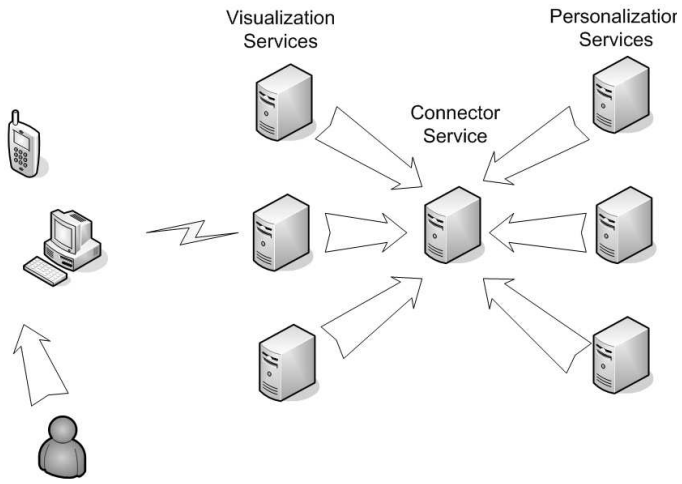

Fig. 5. Architecture of the Personal Reader framework, showing the different components of the Personal Reader: Visualization (user interface), the Personal Reader backbone (consisting of the Connector services, the Reasoning service(s)), and some data-provision services, for RDF data and for the connection with some database for storing user profile information.

VIII. Personalization Services for the Semantic Web: The Personal Reader Framework

How can we establish personalization for the Semantic Web? Personalization can provide guidance, recommendations, hints for a user browsing the Web, it makes the retrieval process of information more effective, it supports users in managing their view on information on the Web, etc. To sum it up, personalization provides an *added value* or a *service* to the end user. One approach for bringing personalization functionality to the (Semantic) Web is therefore to realize *Personalization Web services* which are offered to end user for selection according to their convenience, or to applications for retrieving and integrating additional functionality, as discussed in Section V. In this section, we describe two demonstrator applications for implementing personalization functionality in the Semantic Web, following the approach discussed in Section IV: a Personal Reader Instance for the e-Learning domain, and a Personal Publication Reader.

A. Architectural Overview of the Personal Reader Framework

The Personal Reader Framework[1] is an environment for designing, implementing and maintaining personal Web-content Readers [41], [56]. These personal Web-content Readers allow a user to browse information (the *Reader* part), and to access personal recommendations and contextual information on the currently regarded Web resource (the *Personal* part). We will briefly outline the underlying architecture of the Personal Reader framework, and discuss in more detail how personalization services for two instances of Personal Readers have been implemented.

The architecture of the Personal Reader is a rigorous approach for applying recent Semantic Web technologies. A modular framework of Web services – for constructing *the user interface*, for *mediating* between user requests and currently available personalization services, for *user modeling*, and for offering *personalization functionality* – forms the basis for the Personal Reader. The communications between all components / services is syntactically based on RDF descriptions (see Figure 5).

The common ”understanding” of the services is realized by referring to semantics in the ontologies which provide the valid vocabulary for describing functionality, user interface components, requests, etc. In particular, we employ the following ontologies for describing our objects of discourse, following the logic-based definition of adaptive hypermedia systems [58]:

1) a domain ontology describing the application domain, and a document ontology.
2) a user model ontology (attribute–value pairs for user characteristics, preferences, information on the devices the user is using for accessing the Personal Reader, etc.);
3) an observation ontology (for describing the different kinds of user observations made during runtime);
4) and an adaptation ontology for describing the adaptation functionality which is provided by the adaptation services.

The underlying architecture of the Personal Reader Framework allows to design, implement and maintain

[1] www.personal-reader.de

Fig. 6. Determining details for the currently used learning resource

```
FORALL LO, LO_DETAIL detail_learningobject(LO, LO_DETAIL) <-
  EXISTS C, C_DETAIL(detail_concepts(C, C_DETAIL)
    AND concepts_of_LO(LO, C) AND concepts_of_LO(LO_DETAIL, C_DETAIL))
    AND learning_resource(LO_DETAIL) AND NOT unify(LO,LO_DETAIL).
```

Personal Web Content Readers. In the following, we describe two Personal Reader instances which have been recently developed: A Personal Reader for the e-Learning domain, and a Personal Publication Reader developed for the publications of the Network of Excellence REWERSE[2].

B. A Personal Reader Instance: Personal Reader for e-Learning

Let us start with a specific scenario, involving a user, Alice, interested in learning Java programming:

> Alice is currently learning about variables in Java by accessing some learning resource in an online tutorial. During her studies she realizes that she needs some clarifications on naming variables. The Personal Reader shows where detailed information on variables can be found in this online tutorial, and also points out recommended references for deeper understanding. For ensuring that Alice understands the use of variables, the Personal Reader provides several quizzes. When practicing, Alice does some of the recommended exercises. For the chosen exercises, the Personal Reader provides Alice with appropriate links to the Java API, and some already solved exercises. A further source of information are the JAVA FAQ references pointed out to Alice by the Personal Reader.

The Personal Reader for e-Learning (PR-eL) provides a learner with such a personal interface for studying learning resources: the *Personal Annotation service* recommends the learner next learning steps to take, points to examples, summary pages, more detailed information, etc., and always recommends the most appropriate of these information according to the learner's current knowledge, his/her learning style, learning goal, background, etc.

We provide some examples of personalization rules from the Personal Annotation services of the PR-eL for learning the Java programming language. This Personal Reader helps the learner to view the learning resources from the Sun Java Tutorial [35], a freely available online Tutorial on Java programming, in a context: more *details* related to the topics of the learning resource, the *general topics* the learner is currently studying, *examples*, *summaries*, *quizzes*, etc. are generated and enriched with personal recommendations according to the learner's current learning state.

For implementing the reasoning rules, we currently use the TRIPLE [91] query and rule language for the Semantic Web. Rules defined in TRIPLE can reason about RDF-annotated information resources (required translation tools from RDF to triple and vice versa are provided). An RDF statement (which is a triple) is written as `subject[predicate -> object]`.

RDF *models* are explicitly available in TRIPLE: Statements that are true in a specific model are written as "@model". This in particular is important for constructing the *temporal knowledge bases* as required in the Personal Reader. Connectives and quantifiers for building logical formulae from statements are allowed as usual: `AND`, `OR`, `NOT`, `FORALL`, `EXISTS`, `<-`, `->`, etc. are used.

In the following, we will describe some of the rules that are used by the Personal Reader for learning resources to determine appropriate adaptation strategies.

a) Providing a Context by Displaying Details of a Learning Resource.: Generating links to more detailed learning resources is an adaptive functionality in this example Personal Reader.

The adaptation rule takes the isA hierarchy in the domain ontology, in this case the domain ontology for Java programming, into account to determine domain concepts which are details of the current concept or concepts that the learner is studying on the learning resource. In particular, more details for the currently used learning resource are determined by `detail_learningobject(LO, LO_DETAIL)`, see Figure 6, where `LO` and `LO_Detail` are learning resources, and where `LO_DETAIL` covers more specialized learning concepts which are determined with help of the domain ontology.

N. B. the rule does neither require that `LO_DETAIL` covers all specialized learning concepts, nor that it exclusively covers specialized learning concepts. Further refinements of this adaptation rule are of course possible and should, in a future version of the Personal Reader, be available as tuning parameters under control of the learner. The rules for embedding a learning resource into more general aspects with respect to the current learning progress are similar.

[2]rewerse.net

b) Providing Pointers to Quizzes.: Another example, Figure 7, of an *adaptation rule* for generating embedding context is the recommendation of quiz pages. A learning resource `Q` is recommended as a quiz for a currently learned learning resource `LO` if it is a quiz (the rule for determining this is not displayed) and if it provides questions to at least some of the concepts learned on `LO`.

c) Calculating Recommendations.: Recommendations are personalized according to the current learning progress of the user, e. g. with respect to the current set of course materials. The rule in Figure 8 determines that a learning resource `LO` is `recommended` if the learner studied at least one more general learning resource (`UpperLevelLO`).

Additional rules deriving stronger recommendations (e. g., if the user has studied *all* general learning resources), less strong recommendations (e.g., if one or two of these haven't been studied so far), etc., are possible, too.

Recommendations can also be calculated with respect to the current domain ontology, Figure 9. This is necessary if a user is regarding course materials from different courses at the same time.

However, the first recommendation rule, which reasons within one course will be more accurate because it has more fine–grained information about the course and thus on the learning process of a learner taking part in this course.

d) Reasoning Rules for User Modeling.: The Personal Reader requires only view information about the user's characteristics. Thus, for our example we employed a very simple user model: This user model traces the user's path in the learning environment and registers whenever the user has visited some learning resource. This information is stored in the user's profile, which is bound to RDF as shown in Figure 10.

From this information, we derive whether a particular user learned some concept. The rule in Figure 11 derives all learned concepts.

Similarly, it can be determined whether a learning object has been learned by a user.

C. A Personal Reader Instance: The Personal Publication Reader

Again, let us consider a scenario first for describing the idea of the Personal Publication Reader:

> Bob is currently employed as a researcher in a university. Of course, he is interested in making his publications available to his colleagues, for this he publishes all his publications at his insitute's Web page. Bob is also enrolled in a research project. From time to time, he is requested to notify the project coordination office about his new publications. Furthermore, the project coordination office maintains a member page where information about the members, their involvement in the project, research experience, etc. is maintained.

Can we simplify this process? And, furthermore, can we use this information to provide new, syndicated information? From the scenario, we may conclude that most likely the partners of a research project have their own Websites where they publish their research papers. In addition, information about the role of researchers in the project like "Bob is participating mainly in working group X, and working group X is strongly cooperating with working groups Y and Z" might be available. If we succeed in making this information available to machines to reason about, we can derive new information like: "This research paper of Bob is related to working group X, other papers of working group X on the same research aspects are A, B, and C, etc."

To realize a Personal Publication Reader (PR-R), we extract the publication information from the various websites of the partners in the REWERSE project: All Web-pages containing information about publications of the REWERSE network are periodically crawled and new information is automatically detected, extracted and indexed in the repository of semantic descriptions of the REWERSE network (see Section VIII-D). This information, together with extracted information on the project REWERSE, on people involved in the project, their research interests, etc., is used to provide more information on each publication: who has authored it, which research groups are related to this kind of research, which other publications are published by the research group, which other publications of the author are available, which other publications are on the similar research, etc. (see Section VIII-E)

D. Gathering Data for Semantic Web Applications

Each institute and organization offers access to its publications on the Web. However, each presentation is usually different, some use e.g. automatic conversions of *bibtex* or other files, some are manually maintained. Such a presentation is well suited for human consumption, but hardly usable for automatic processing. Consider e.g. the scenario that we are interested in all publications of REWERSE project members in the year 2003 which contain the word "personalization" in their title or abstract. To be able to formulate such queries and to generate personalized views on heterogeneously presented publications it is necessary to first have access to the publication data in a more structured form.

Fig. 7. Adaptation rule example

```
 FORALL Q quiz(Q) <-
   Q['http://www.w3.org/1999/02/22-rdf-syntax-ns#':type ->
         'http://ltsc.ieee.org/2002/09/lom-educational#':'Quiz']
FORALL Q, C concepts_of_Quiz(Q,C) <- quiz(Q) AND concept(C)
   AND  Q['http://purl.org/dc/elements/1.1/':subject -> C].
FORALL LO, Q quiz(LO, Q) <- EXISTS C (concepts_of_LO(LO,C)
   AND concepts_of_Quiz(Q,C)).
```

Fig. 8. Recommending a resource

```
 FORALL LO1, LO2 upperlevel(LO1,LO2) <-
   LO1['http://purl.org/dc/terms#':isPartOf -> LO2].
FORALL LO, U learning_state(LO, U, recommended) <-
   EXISTS UpperLevelLO (upperlevel(LO, UpperLevelLO)
   AND p_obs(UpperLevelLO, U, Learned)).
```

Fig. 9. Recommendation with respect to the current domain ontology

```
 FORALL C, C_DETAIL detail_concepts(C, C_DETAIL) <-
   C_DETAIL['http://www.w3.org/2000/01/rdf-schema#':subClassOf -> C]
   AND concept(C) AND concept(C_DETAIL).

FORALL LO, U learning_state(LO, U, recommended) <-
  EXISTS C, C_DETAIL (concepts_of_LO(LO, C_DETAIL)
     AND detail_concepts(C, C_DETAIL) AND p_obs(C, U, Learned) ).
```

Fig. 10. Storing information in the user's profile

```
 <rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:j.0="http://semweb.kbs.uni-hannover.de/rdf/l3s.rdf#" >
 <rdf:Description rdf:about="http://semweb.kbs.uni-hannover.de/user#john">
   <rdf:type rdf:resource="http://hoersaal..../rdf/l3s.rdf#User"/>
   <j.0:hasVisited>http://java.sun.com/.../variables.html</j.0:hasVisited>
    ...
```

Fig. 11. Rule deriving all learned concepts

```
 FORALL C, U  p_obs(C, U, Learned) <-
   EXISTS LO (concepts_of_LO(LO, C) AND
   U['http://semweb.kbs.uni-hannover.de/rdf/l3s#':hasVisited ->LO]).
```

Fig. 12. Sample RDF output entry

```
<rdf:Description rdf:about="http://www.example.org/id/16">
  <rewerse:origin>University of Heraklion</rewerse:origin>
  <rewerse:title>Describing Knowledge Representation Schemes:
   A Formal Account</rewerse:title>
  <rewerse:author>
    <rdf:Seq>
      <rdf:li rdf:resource="#Giorgos Flouris" />
      <rdf:li rdf:resource="#Dimitris Plexousakis" />
      <rdf:li rdf:resource="#Grigoris Antoniou" />
    </rdf:Seq>
  </rewerse:author>
  <rewerse:year>2003</rewerse:year>
  <rewerse:link>ftp://ftp.ics.forth.gr/tech-reports/2003/
   2003.TR320.Knowledge_Representation_Schemes.pdf</rewerse:link>
  <rewerse:abstract>The representation and manipulation of knowledge
   has been drawing a great deal of attention since the early [...]
  </rewerse:abstract>
</rdf:Description>
```

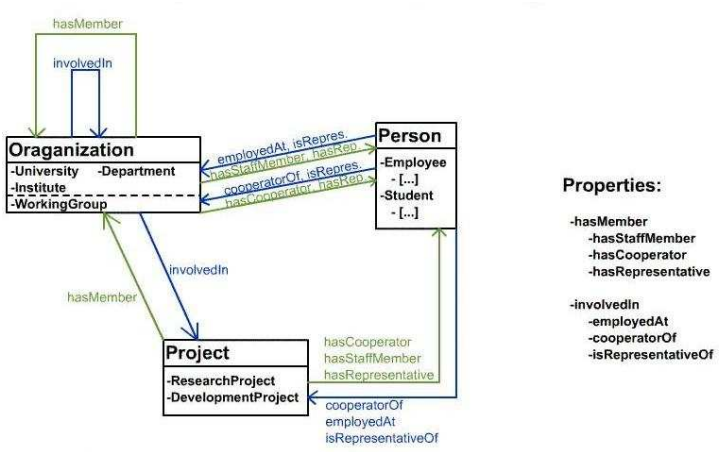

Fig. 13. Part of the Ontology on Researchers used in the Personal Publication Reader

Fig. 14. Example of a rule determining all authors of a publication

```
FORALL A, P all_authors(A, P) <-
  EXISTS X, R (
  P['http://.../rewerse#':author -> X]@'http:...#':publications
  AND X[R -> 'http://www.../author':A]@'http:...#':publications).
```

Fig. 15. Example rule determining the employer of a project member

```
FORALL A,I works_at(A, I) <-
   EXISTS A_id,X (name(A_id,A)
     AND ont:A_id[ont:involvedIn -> ont:I]@'http:...#':researcher
     AND ont:X[rdfs:subClassOf ->
               ont:Organization]@rdfschema('http:...#':researcher)
     AND ont:I[rdf:type -> ont:X]@'http:...#':researcher).
```

In Section VII we discussed data extraction from the Web and the Lixto methodology. Here, we apply Lixto to regularly extract publication data from all REWERSE members. As Figure 4 illustrates, the disks are Lixto wrappers that regulary (e.g. once a week) navigate to the page of each member (such as Munich, Hannover, Eindhoven) and apply a wrapper that extracts at least author names, publication titles, publication year and link to the publication (if available). Figure 3 illustrates the visual wrapper specification on the Munich page.

In the "XSL" components publication data is harmonized to fit into a common structure and an attribute "origin" is added containing the institution's name. The triangle in Figure 4 represents a data integration unit; here data from the various institutions is put together and duplicate entries are removed. IDs are assigned to each publication in the next step. Finally, the XML data structure is mapped to a predefined RDF structure (this happens in the lower arc symbol in Figure 4) and passed on to the Personal Publication Reader as described below. A second deliverer component delivers the XML publication data additionally in RDF. One sample RDF output entry is depicted in Figure 12.

This Lixto application can be easily enhanced by connecting further Web sources. For instance, abstracts from `www.researchindex.com` can be queried for each publication lacking this information and joined to each entry, too. Moreover, using text categorization tools one can rate and classify the contents of the abstracts.

E. Content Syndication and Personalized Views

In addition to the extracted information on research papers that we obtain as described in the previous section, we collect the data about the members of the research project from the member's corner of the REWERSE project. We have constructed an ontology for describing researchers and their envolvment in

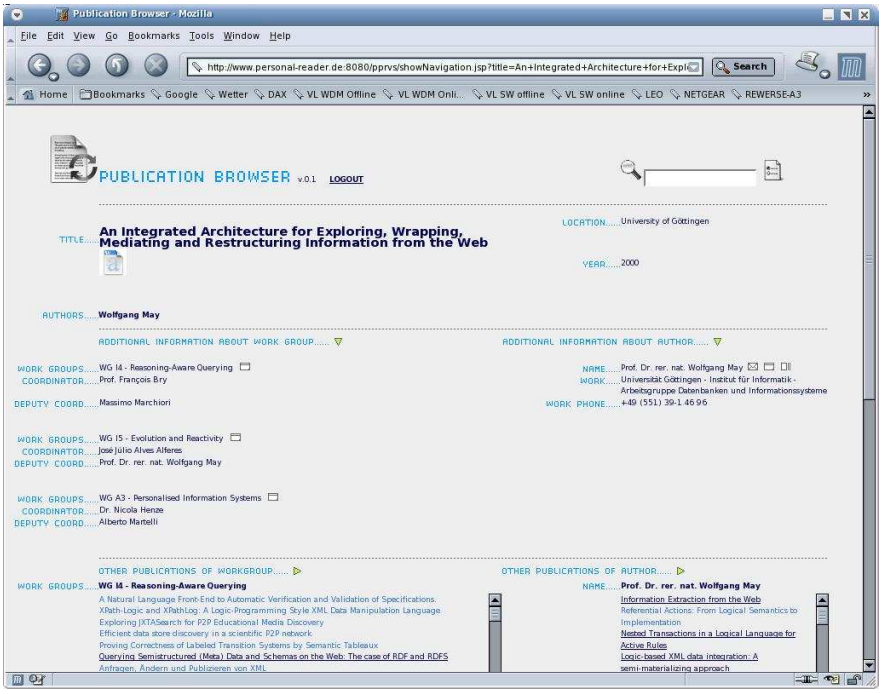

Fig. 16. Screenshot of the Personal Publication Reader

REWERSE. A part of this ontology can be seen in Figure 13

All the collected information is then used in a Personalization service which provides the end user with an interface for browsing publications of the REWERSE project, and having instantly access to further information on authors, the working groups of REWERSE, recommended related publications, etc.

The Personalization service of the PR-R uses, similar to the PR-eL, personalization rules for deriving new facts, and for determining recommendations for the user. As an example, the rule in Figure 14 determines all authors of a publication:

Further rules combine information on these authors from the researcher ontology with the author information. E.g. the rule in Figure 15 determines the employer of a project member, which might be a company, or a university, or, more generally, some instance of a subclass of an organization:

The screenshot in fig. 16 depicts the output of the visualization service of the PR-R.

By further exploiting the Web service architecture of the Personal Reader, it is possible to *link* to the PR other (reasoning) services, such as a personal sequencing service, implemented as a planner by exploiting the action metaphor, or making use of the non-monotonic reasoning functionality or the ECA paradigm and expressiveness for more advanced personalization functionality.

IX. Acknowledgement

This research has been carried out in connection with the Network of Excellence REWERSE[3] which strives for a (minimal) set of rule and reasoning languages for the Semantic Web.

X. Conclusions

This paper discusses recent approaches for shaping the logic layer of the Semantic Web, and for supporting approaches to personalization in the Semantic Web. We demonstrate approaches for rules and rule-languages in the logic layer of the Semantic Web. Special attention is devoted to the important aspects of evolution, updates and events, and their consequences for personalization and reasoning. Approaches to personalization via reasoning about actions is examplified for different scenarios.

Query- and transformation languages as well as Web data extraction for maintaining and constructing semantic descriptions are discussed. Finally, personalized Web systems making use of these reasoning techniques, semantic descriptions and extractions, are introduced.

[3]REWERSE - Reasoning on the Web, Network of Excellence founded in the 6th European Framework Programme, rewerse.net

REFERENCES

[1] ALFERES, J. J., BROGI, A., LEITE, J. A., AND PEREIRA, L. M. Evolving logic programs. In *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA'02)* (2002), S. Flesca, S. Greco, N. Leone, and G. Ianni, Eds., vol. 2424 of *LNCS*, Springer-Verlag, pp. 50–61.

[2] ALFERES, J. J., LEITE, J. A., PEREIRA, L. M., PRZYMUSINSKA, H., AND PRZYMUSINSKI, T. C. Dynamic updates of non-monotonic knowledge bases. *The Journal of Logic Programming 45*, 1–3 (2000), 43–70. A shorter version appeared in "Principles of Knowledge Representation and Reasoning '98".

[3] ANTONIOU, G., BIKAKIS, A., AND WAGNER, G. A system for nonmonotonic rules on the web. In *Proc. of RuleML-2004* (2004), Springer LNCS.

[4] ANTONIOU, G., BILLINGTON, D., GOVERNATORI, G., AND MAHER, M. Representation Results for Defeasible Logic. *ACM Transactions on Computational Logic 2,2* (2002), 255–287.

[5] ANTONIOU, G., AND VAN HARMELEN, F. *A Semantic Web Primer*. MIT Press, 2004.

[6] ATZENI, P., AND MECCA, G. Cut and paste. In *Proc. of PODS* (1997).

[7] BALDONI, M., BAROGLIO, C., MARTELLI, A., AND PATTI, V. Reasoning about interaction for personalizing web service fruition. In *Proc. of WOA 2003: Dagli oggetti agli agenti, sistemi intelligenti e computazione pervasiva* (Villasimius (CA), Italy, September 2003), G. Armano, F. De Paoli, A. Omicini, and E. Vargiu, Eds., Pitagora Editrice Bologna.

[8] BALDONI, M., BAROGLIO, C., MARTELLI, A., AND PATTI, V. Reasoning about self and others: communicating agents in a modal action logic. In *Proc. of ICTCS'2003* (2003), vol. 2841 of *LNCS*, Springer, pp. 228–241.

[9] BALDONI, M., BAROGLIO, C., AND PATTI, V. Web-based adaptive tutoring: an approach based on logic agents and reasoning about actions. *Artificial Intelligence Review 22*, 1 (2004), 3–39.

[10] BALDONI, M., BAROGLIO, C., PATTI, V., AND TORASSO, L. Reasoning about learning object metadata for adapting scorm courseware. In *AH 2004: Workshop Proceedings, Part I, EAW 2004: Engineering the Adaptive Web* (Eindhoven, Holland, August 2004), L. Aroyo and C. Tasso, Eds., CS-Report 04-18, Technische Universiteit Eindhoven, pp. 4–13.

[11] BALDONI, M., GIORDANO, L., MARTELLI, A., AND PATTI, V. An Abductive Proof Procedure for Reasoning about Actions in Modal Logic Programming. In *Proc. of NMELP'96* (1997), J. Dix et.al., Ed., vol. 1216 of *LNAI*, Springer-Verlag, pp. 132–150.

[12] BALDONI, M., GIORDANO, L., MARTELLI, A., AND PATTI, V. Programming Rational Agents in a Modal Action Logic. *Annals of Mathematics and Artificial Intelligence, Special issue on Logic-Based Agent Implementation 41*, 2-4 (2004), 207–257.

[13] BARAL, C., AND SON, T. C. Formalizing Sensing Actions - A transition function based approach. *Artificial Intelligence 125*, 1-2 (January 2001), 19–91.

[14] BASSILIADES, N., ANTONIOU, G., AND VLAHAVAS, I. A defeasible logic system for the semantic web. In *Principles and Practice of Semantic Web Reasoning* (2004), Springer LNCS 3208.

[15] BAUMGARTNER, R., EICHHOLZ, S., FLESCA, S., GOTTLOB, G., AND HERZOG, M. Semantic Markup of News Items with Lixto. In *Annotation for the Semantic Web* (2003).

[16] BAUMGARTNER, R., FLESCA, S., AND GOTTLOB, G. Declarative Information Extraction, Web Crawling and Recursive Wrapping with Lixto. In *Proc. of LPNMR* (2001).

[17] BAUMGARTNER, R., FLESCA, S., AND GOTTLOB, G. Visual web information extraction with Lixto. In *Proc. of VLDB* (2001).

[18] BAUMGARTNER, R., FLESCA, S., GOTTLOB, G., AND HERZOG, M. Building dynamic information portals - a case study in the agrarian domain. In *Proc. of IS* (2002).

[19] BAUMGARTNER, R., GOTTLOB, G., AND HERZOG, M. Lixto - Halfway to the Semantic Web. *OEGAI-Journal 1* (2003), 19–24.

[20] BAUMGARTNER, R., GOTTLOB, G., HERZOG, M., AND SLANY, W. Interactively Adding Web Service Interfaces to Existing Web Applications. In *Proc. of SAINT* (2004).

[21] BAUMGARTNER, R., HERZOG, M., AND GOTTLOB, G. Visual programming of web data aggregation applications. In *Proc. of IIWeb-03* (2003).

[22] BECKETT, D. Rdf/xml syntax specification. http://www.w3.org/TR/rdf-syntax-grammar/.

[23] BERGER, S., BRY, F., AND SCHAFFERT, S. A Visual Language for Web Querying and Reasoning. In *Proceedings of Workshop on Principles and Practice of Semantic Web Reasoning, Mumbai, India (9th–13th December 2003)* (2003), vol. 2901 of *LNCS*.

[24] BERGER, S., BRY, F., SCHAFFERT, S., AND WIESER, C. Xcerpt and visXcerpt: From Pattern-Based to Visual Querying of XML and Semistructured Data. In *Proceedings of 29th Intl. Conference on Very Large Data Bases, Berlin, Germany (9th–12th September 2003)* (2003).

[25] BERGER, S., BRY, F., AND WIESER, C. Visual Querying for the Semantic Web. In *Proceedings of 23rd International Conference on Conceptual Modeling, Shanghai, China (8th–12th November 2004)* (2004).

[26] BERGMAN, M. K. The deep web: Surfacing hidden value. BrightPlanet White Paper, http://www.brightplanet.com/technology/deepweb.asp.

[27] BERNERS-LEE, T., HENDLER, J., AND LASSILA, O. The semantic web. *Scientific American* (May 2001).

[28] BERNERS-LEE, T., HENDLER, J., AND LASSILA, O. The Semantic Web – A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American* (May 2001).

[29] BONNER, A. J., AND KIFER, M. An overview of transaction logic. *Theoretical Computer Science 133* (1994).

[30] BRICKLEY, D., AND GUHA, R. Rdf vocabulary description language 1.0: Rdf schema. http://www.w3.org/TR/rdf-schema/.

[31] BRY, F., FURCHE, T., PĂTRÂNJAN, P.-L., AND SCHAFFERT, S. Data Retrieval and Evolution on the (Semantic) Web: A Deductive Approach. In *Proceedings of Workshop on Principles and Practice of Semantic Web Reasoning, St. Malo, France (6th–10th September 2004)* (2004), REWERSE.

[32] BRY, F., AND PĂTRÂNJAN, P.-L. Reactivity on the Web: Paradigms and Applications of the Language XChange. In *20th Annual ACM Symposium on Applied Computing (SAC'2005)* (2005).

[33] BRY, F., AND SCHAFFERT, S. Towards a Declarative Query and Transformation Language for XML and Semistructured Data: Simulation Unification. In *Proceedings of International Conference on Logic Programming, Copenhagen, Denmark (29th July–1st August 2002)* (2002), vol. 2401 of *LNCS*.

[34] BRYSON, J., MARTIN, D., MCILRAITH, S., AND STEIN, L. A. Agent-based composite services in DAML-S: The behavior-oriented design of an intelligent semantic web, 2002.

[35] CAMPIONE, M., AND WALRATH, K. The java tutorial, 2003. http://java.sun.com/docs/books/tutorial/.

[36] CASTILHO, M., GASQUET, O., AND HERZIG, A. Modal tableaux for reasoning about actions and plans. In *Proc. ECP'97* (1997), S. Steel, Ed., LNAI, pp. 119–130.

[37] CRESCENZI, V., MECCA, G., AND MERIALDO, P. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of 27th International Conference on Very Large Data Bases* (2001), pp. 109–118.

[38] DAML-S. http://www.daml.org/services/daml-s/0.9/. version 0.9, 2003.

[39] DAVULCU, H., YANG, G., KIFER, M., AND RAMAKRISHNAN, I. Computational aspects of resilient data extractraction from semistructured sources. In *Proc. of PODS* (2000).

[40] DEAN, M., AND SCHREIBER, G. Owl web ontology language reference. http://www.w3.org/TR/owl-ref/.

[41] DOLOG, P., HENZE, N., NEJDL, W., AND SINTEK, M. The Personal Reader: Personalizing and Enriching Learning Resources using Semantic Web Technologies. In *Proccedings of the 3nd*

International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH 2004) (Eindhoven, The Netherlands, 2004).

[42] EITER, T., FINK, M., SABBATINI, G., AND TOMPITS, H. Declarative update policies for nonmonotonic knowledge bases. In *Logics for Emerging Applications of Databases*, J. Chomicki, R. van der Meyden, and G. Saake, Eds. Springer-Verlag, 2003, ch. 3, pp. 85–129.

[43] EITER, T., FINK, M., SABBATINI, G., AND TOMPITS, H. Reasoning about Evolving Nonmonotonic Knowledge Bases. *ACM Transactions on Computational Logic* (2004). To appear.

[44] EITER, T., LUKASIEWICZ, T., SCHINDLAUER, R., AND TOMPITS, H. Combining answer set programming with description logics for the semantic web. In *Proceedings KR-2004* (2004), pp. 141–151. http://www.kr.tuwien.ac.at/staff/roman/semweblp/.

[45] EITER, T., LUKASIEWICZ, T., SCHINDLAUER, R., AND TOMPITS, H. Well-founded semantics for description logic programs in the semantic web. In *Proceedings RuleML 2004 Workshop, ISWC Conference, Hiroshima, Japan* (2004), Springer, pp. 81–97. http://www.kr.tuwien.ac.at/staff/roman/semweblp/.

[46] ETZIONI, O., CAFARELLA, M., DOWNEY, D., KOK, S., POPESCU, A., SHAKED, T., SODERLAND, S., WELD, D. S., AND YATES, A. Web-Scale Information Extraction in KnowItAll (Preliminary Results). In *Proceedings of the World Wide Web Conference 2004* (2004).

[47] FLESCA, S., MANCO, G., MASCIARI, E., RENDE, E., AND TAGARELLI, A. Web wrapper induction: a brief survey. *Journal of the ACM, 51(1)* (2004).

[48] GABBAY, D., AND PH.SMETS, Eds. *Handbook on Defeasible Reasoning and Uncertainty Management Systems*, vol. III: Belief Change. Kluwer Academic, 1998.

[49] GELFOND, M., AND LIFSCHITZ, V. Classical negation in logic programs and disjunctive databases. In *In* New Generation Computing (1991), vol. 9, pp. 365–385.

[50] GELFOND, M., AND LIFSCHITZ, V. Representing action and change by logic programs. *Journal of Logic Programming 17* (1993), 301–321.

[51] GIACOMO, G. D., LESPÈRANCE, Y., AND LEVESQUE, H. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence 121* (2000), 109–169.

[52] GIORDANO, L., MARTELLI, A., AND SCHWIND, C. Dealing with concurrent actions in modal action logic. In *Proc. ECAI-98* (1998), pp. 537–541.

[53] GOTTLOB, G., AND KOCH, C. Monadic datalog and the expressive power of languages for Web Information Extraction. In *Proc. of PODS* (2002).

[54] GOTTLOB, G., AND KOCH, C. Monadic Datalog and the Expressive Power of Web Information Extraction Languages. *AI Communications Vol.17/2* (2004).

[55] GROSOF, B. N., HORROCKS, I., VOLZ, R., AND DECKER, S. Description logic programs: Combining logic programs with description logic. In *Twelfth International World Wide Web Conference* (Budapest, Hungary, May 2003).

[56] HENZE, N., AND HERRLICH, M. The Personal Reader: A Framework for Enabling Personalization Services on the Semantic Web. In *Proceedings of the Twelfth GI- Workshop on Adaptation and User Modeling in Interactive Systems (ABIS 04)* (Berlin, Germany, 2004).

[57] HENZE, N., AND KRIESELL, M. Personalization functionality for the semantic web: Architectural outline and first sample implementation. In *Proccedings of the 1st International Workshop on Engineering the Adaptive Web (EAW 2004), held at the Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH 2004)* (Eindhoven, The Netherlands, 2004). To appear.

[58] HENZE, N., AND NEJDL, W. A logical characterization of adaptive educational hypermedia. *New Review of Hypermedia 10*, 1 (2004).

[59] HERZOG, M., AND GOTTLOB, G. InfoPipes: A flexible framework for M-Commerce applications. In *Proc. of TES workshop at VLDB* (2001).

[60] HEYMANS, S., AND VERMEIR, D. Integrating semantic web reasoning and answer set programming. In Answer Set Programming, Advances in Theory and Implementation, Proc. 2nd Intl. ASP'03 Workshop, Messina, Italy (2003), pp. 194–208.

[61] HORROCKS, I., PATEL-SCHNEIDER, P., BOLEY, H., TABET, S., AND GROSOF, B. Swrl: A semantic web rule language combining owl and ruleml. http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

[62] HUCK, G., FANKHAUSER, P., ABERER, K., AND NEUHOLD, E. JEDI: Extracting and synthesizing information from the web. In *Proc. of COOPIS* (1998).

[63] KOWALSKI, R., AND SERGOT, M. A Logic-based Calculus of Events. *New Generation of Computing 4* (1986), 67–95.

[64] KUHLINS, S., AND TREDWELL, R. Toolkits for generating wrappers. In *Net.ObjectDays* (2002).

[65] KUSHMERICK, N., WELD, D., AND DOORENBOS, R. Wrapper induction for information extraction. In *Proc. of IJCAI* (1997).

[66] LAENDER, A. H., RIBEIRO-NETO, B. A., DA SILVA, A. S., AND TEIXEIRA, J. S. A brief survey of web data extraction tools. In *Sigmod Record 31/2* (2002).

[67] LEONE, N., PFEIFER, G., FABER, W., EITER, T., GOTTLOB, G., PERRI, S., AND SCARCELLO, F. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic* (2004). To appear. Available via `http://www.arxiv.org/ps/cs.AI/0211004`.

[68] LEVESQUE, H. J., REITER, R., LESPÉRANCE, Y., LIN, F., AND SCHERL, R. B. GOLOG: A Logic Programming Language for Dynamic Domains. *J. of Logic Programming 31* (1997), 59–83.

[69] LEVY, A., AND ROUSSET, M.-C. Combining horn rules and description logics in carin. *Artificial Intelligence 104(1-2)* (1998), 165–209.

[70] LIU, L., PU, C., AND HAN, W. XWrap: An extensible wrapper construction system for internet information. In *Proc. of ICDE* (2000).

[71] LIU, Z., LI, F., AND NG, W. K. Wiccap Data Model: Mapping Physical Websites to Logical Views. In *Proceedings of the 21st International Conference on Conceptual Modelling (ER2002)* (Tempere, Finland, October 7-11 2002).

[72] LOBO, J., MENDEZ, G., AND TAYLOR, S. R. Adding Knowledge to the Action Description Language $\mathcal{A}$. In *Proc. of AAAI'97/IAAI'97* (Menlo Park, 1997), pp. 454–459.

[73] The mandarax project. http://www.mandarax.org.

[74] MAY, W., ALFERES, J. J., AND BRY, F. Towards generic query, update, and event languages for the semantic web. In *Principles and Practice of Semantic Web Reasoning (PPSWR)* (2004), no. 3208 in LNCS, Springer, pp. 19–33.

[75] MAY, W., HIMMERÖDER, R., LAUSEN, G., AND LUDÄSCHER, B. A unified framework for wrapping, mediating and restructuring information from the web. In *WWWCM* (1999), Sprg. LNCS 1727.

[76] MCCARTHY, J., AND HAYES, P. Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Machine Intelligence 4* (1963), 463–502.

[77] MCILRAITH, S., AND SON, T. Adapting Golog for Programming the Semantic Web. In *5th Int. Symp. on Logical Formalization of Commonsense Reasoning* (2001), pp. 195–202.

[78] MCILRAITH, S. A., SON, T. C., AND ZENF, H. Semantic Web Services. *IEEE Intelligent Systems* (March/April 2001), 46–53.

[79] MENG, X., WANG, H., LI, C., AND KOU, H. A schema-guided toolkit for generating wrappers. In *Proc. of WEBSA2003* (2003).

[80] MUSLEA, I., MINTON, S., AND KNOBLOCK, C. A hierarchical approach to wrapper induction. In *Proc. of 3rd Intern. Conf. on Autonomous Agents* (1999).

[81] NIEMELÄ, I., AND SIMONS, P. Implementation of the stable model and well-founded semantics for normal logic programs. In *In J. Dix, U. Furbach, and A. Nerode, editors*, Proc. 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-97) (1997), Springer, pp. 420–429.

[82] OWLS. http://www.daml.org/services/owl-s/. version 1.0, 2004.

[83] PATTI, V. *Programming Rational Agents: a Modal Approach in a Logic Programming Setting*. PhD thesis, Dipartimento

di Informatica, Università degli Studi di Torino, Italy, 2002. Available at `http://www.di.unito.it/~patti/`.

[84] PRENDINGER, H., AND SCHURZ, G. Reasoning about action and change. a dynamic logic approach. *Journal of Logic, Language, and Information 5*, 2 (1996), 209–245.

[85] RAPOSO, J., PAN, A., ALVAREZ, M., HIDALGO, J., AND VINA, A. The Wargo System: Semi-Automatic Wrapper Generation in Presence of Complex Data Access Modes. In *Proceedings of DEXA 2002* (Aix-en-Provence, France, 2002).

[86] RIBEIRO-NETO, B., LAENDER, A. H. F., AND DA SILVA, A. S. Extracting semi-structured data through examples. In *Proc. of CIKM* (1999).

[87] The rule markup initiative. http://www.ruleml.org.

[88] SAHUGUET, A., AND AZAVANT, F. Building light-weight wrappers for legacy web data-sources using W4F. In *Proc. of VLDB* (1999).

[89] SCHAFFERT, S., AND BRY, F. Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In *Proceedings of Extreme Markup Languages 2004, Montreal, Quebec, Canada (2nd–6th August 2004)* (2004).

[90] SCHWIND, C. B. A logic based framework for action theories. In *Language, Logic and Computation* (1997), J. Ginzburg et al., Ed., CSLI, pp. 275–291.

[91] SINTEK, M., AND DECKER, S. TRIPLE - an RDF Query, Inference, and Transformation Language. In *International Semantic Web Conference (ISWC)* (Sardinia, Italy, 2002), I. Horrocks and J. Hendler, Eds., LNCS 2342, pp. 364–378.

[92] WINSLETT, M. *Updating Logical Databases*. Cambridge University Press, 1990.

[93] WSDL. http://www.w3c.org/tr/2003/wd-wsdl12-20030303/. version 1.2, 2003.

[94] http://xcerpt.org.