EXTENDING A MULTI-AGENT REASONING INTEROPERABILITY FRAMEWORK WITH SERVICES FOR THE SEMANTIC WEB LOGIC AND PROOF LAYERS

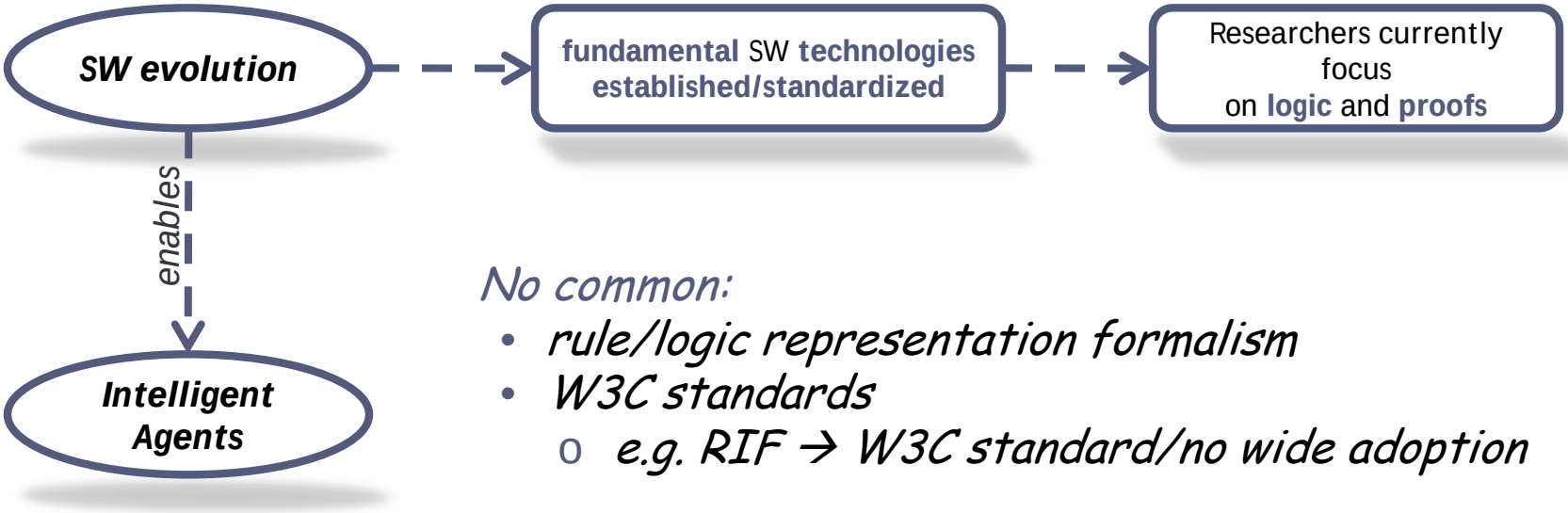
Kalliopi Kravari[1], Konstantinos Papatheodorou[2],
Grigoris Antoniou[2] and Nick Bassiliades[1]

[1]Dept. of Informatics, Aristotle University of Thessaloniki, Greece
[2]Institute of Computer Science, FORTH, Greece
Department of Computer Science, University of Crete, Greece

Overview

- ***Motivation***
- ***EMERALD***
  - ***Reasoners***
- ***DR-Prolog Reasoner***
  - ***RuleMLParser***
  - ***RDFParser***
  - ***ResultParser***
- ***Defeasible Proofing Services***
  - ***Defeasible logic***
  - ***Defeasible Proof Generator Service***
  - ***Proof Validator Service***
- ***Conclusions – Future Work***

Motivation



No common:

- *rule/logic representation formalism*
- *W3C standards*
  - *e.g. RIF → W3C standard/no wide adoption*

*Standards **BUT** not universal languages:*

e.g. RIF, RuleML

- Share syntactic features
- Differ in semantics

Motivation

Agents should somehow share an understanding of each other's position

Solution A: equipping each agent with inference/reasoning mechanism
- agents would require common interchange language
- language translations

Solution B: wrapping reasoning services as IAs
- no need for common logic/rule paradigm
  inferencing tasks conducted by the reasoning services.

Trust:
users should trust systems
→ systems should explain their actions, sources, and beliefs

Traditional trust models guarantee agents' trustworthiness but not the correctness of the inference service itself

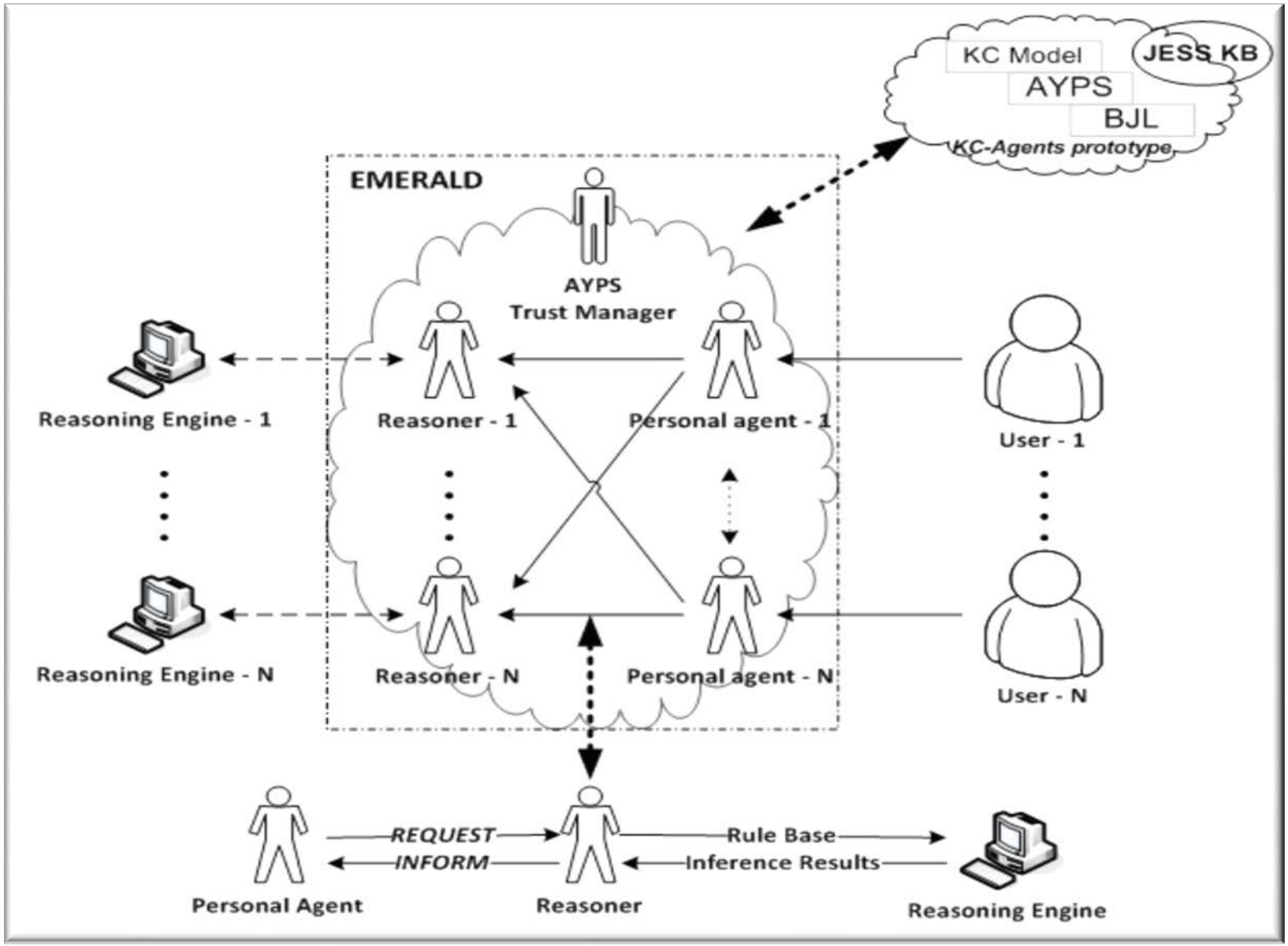
Need for automating proof generation, exchange and validation

Overview

- ***Motivation***
- ***EMERALD***
  - ***Reasoners***
- ***DR-Prolog Reasoner***
  - ***RuleMLParser***
  - ***RDFParser***
  - ***ResultParser***
- ***Defeasible Proofing Services***
  - ***Defeasible logic***
  - ***Defeasible Proof Generator Service***
  - ***Proof Validator Service***
- ***Conclusions – Future Work***

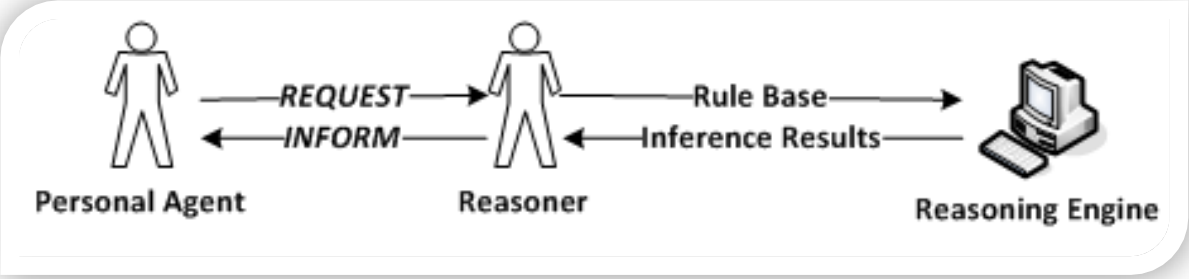
EMERALD

A Multi-Agent Knowledge-Based Framework



Reasoners

- *Built as agents*
- *Act as like web services*
- *Provide the reasoning services*
- *Launch an associated reasoning engine*



Reasoner:

- *stands* by for new requests
- *gets* a valid request → *launches* the reasoning service → *returns* the results

- **EMERALD** supports some reasoning engines that use a variety of logics,

DR-DEVICE, SPINdle , R-DEVICE and PROVA

- **defeasible logic** **datalog-like rules** **prolog-like rules**

Overview

- ***Motivation***
- ***EMERALD***
  - ***Reasoners***
- ***DR-Prolog Reasoner***
  - ***RuleMLParser***
  - ***RDFParser***
  - ***ResultParser***
- ***Defeasible Proofing Services***
  - ***Defeasible logic***
  - ***Defeasible Proof Generator Service***
  - ***Proof Validator Service***
- ***Conclusions – Future Work***

DR-Prolog Reasoner

DR-Prolog[1]:

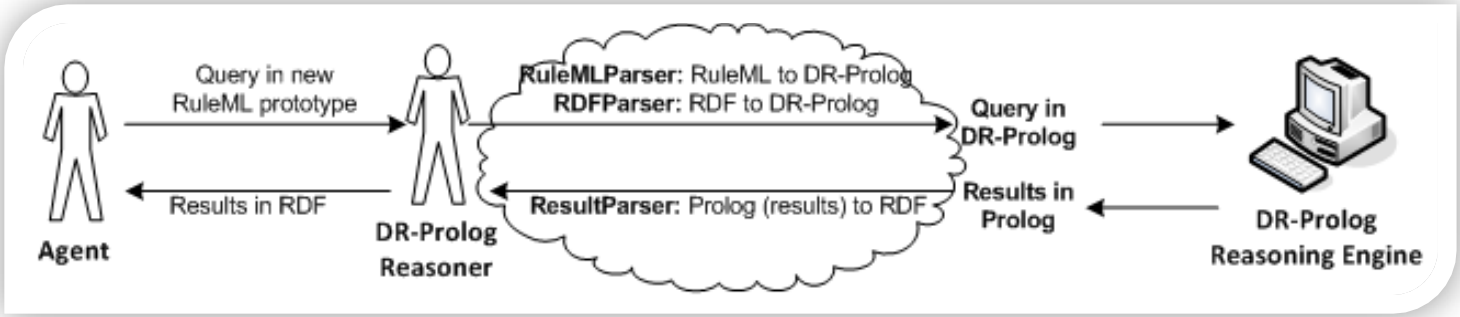
- *built on-top of Prolog*
- *uses defeasible logic rules, facts and ontologies*
- *supports all major SW standards (e.g. RDF, RDFS, OWL, RuleML)*
- *deals with monotonic and nonmonotonic rules, open and closed world assumption and reasoning with inconsistencies*

DR-Prolog Reasoner:

- *follows the EMERALD Reasoners' general functionality*
- *stands* by for new requests
- *gets* a valid request → *launches* DR-Prolog→ *returns* the results

➢ Need for some new intermediate steps

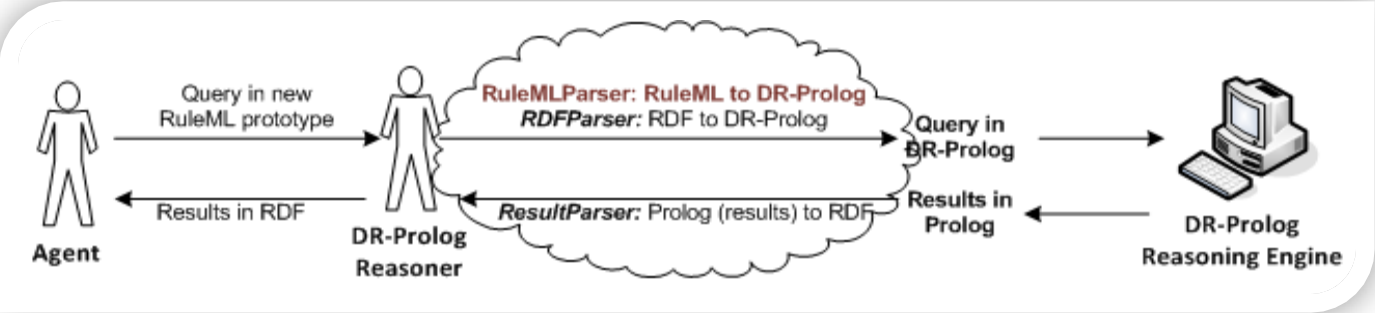
- to process the receiving queries
- to send back the appropriate answer in RDF format



[1]Antoniou, G., Bikakis, A.: DR-Prolog: A System for Defeasible Reasoning with Rules and Ontologies on the SW. IEEE TKDE 19,2

DR-Prolog Reasoner

RuleMLParser: RuleML to DR-Prolog




```
<Implies ruletype="defeasiblerule">
   <oid>r1</oid>
   <head>
      <Atom neg="no">
         <Rel>acceptable</Rel>
         <Slot type="var">X</Slot>
         <Slot type="var">Y</Slot>
         <Slot type="var">Z</Slot>
         <Slot type="var">W</Slot>
      </Atom>
</head>
   <body>
      <part type="atom">
         <Rel>name</Rel>
         <Slot type="var">X</Slot>
         <Slot type="var">X</Slot>
      </part>
         <part type="atom">
         <Rel>price</Rel>
         <Slot type="var">X</Slot>
         <Slot type="var">Y</Slot>
      </part>
         <part type="atom">
         <Rel>size</Rel>
         <Slot type="var">X</Slot>
         <Slot type="var">Z</Slot>
      </part>
         <part type="atom">
         <Rel>size</Rel>
         <Slot type="var">X</Slot>
         <Slot type="var">Z</Slot>
      </part>
         <part type="atom">
          <Rel>gardenSize</Rel>
          <Slot type="var">X</Slot>
          <Slot type="var">W</Slot>
      </part>
       </body>
</Implies>
```

RuleML

```
defeasible(r1,
          acceptable(X,Y,Z,W),
          [name(X,X),price(X,Y),size(X,Z),gardenSize(X,W)]).

defeasibly(acceptable(X,Y,Z,W))
```

DR-Prolog Rule

DR-Prolog Query

RuleMLParser

- receives the RuleML file with the query
- extracts the DR-Prolog rules
- stores them in native DR-Prolog format

- processes the rules in the RuleML file, generating the corresponding DR-Prolog rules
- extracts the queries that are included in the RuleML query, indicating whether it is an "answer" or a "proof" query.

DR-Prolog Reasoner

RDFParser: RDF rule base to DR-Prolog facts


```
<tour:Hotel rdf:ID="CretaMareRoyal">
   <tour:resortID>1</tour:resortID>
   <tour:hotelName>Creta Mare
Royal</tour:hotelName>
   <tour:hotelStars>6</tour:hotelStars>

<tour:hotelCategory>Business</tour:hotelCa
tegory>
   <tour:parking>true</tour:parking>
```

RDF

```
fact(type(CretaMareRoyal,Hotel)).
fact(resortID(CretaMareRoyal,1)).
fact(hotelName(CretaMareRoyal,Creta Mare
Royal)).
fact(hotelStars(CretaMareRoyal,6)).
fact(hotelCategory(CretaMareRoyal,Business
)).
fact(parking(CretaMareRoyal,true)).
fact(swimmingPool(CretaMareRoyal,true)).
fact(breakfast(CretaMareRoyal,true)).
```

DR-Prolog facts

- turning the initial RuleML query and rulebase into DR-Prolog is not enough
- the fact base has to be translated too

RDF → Prolog

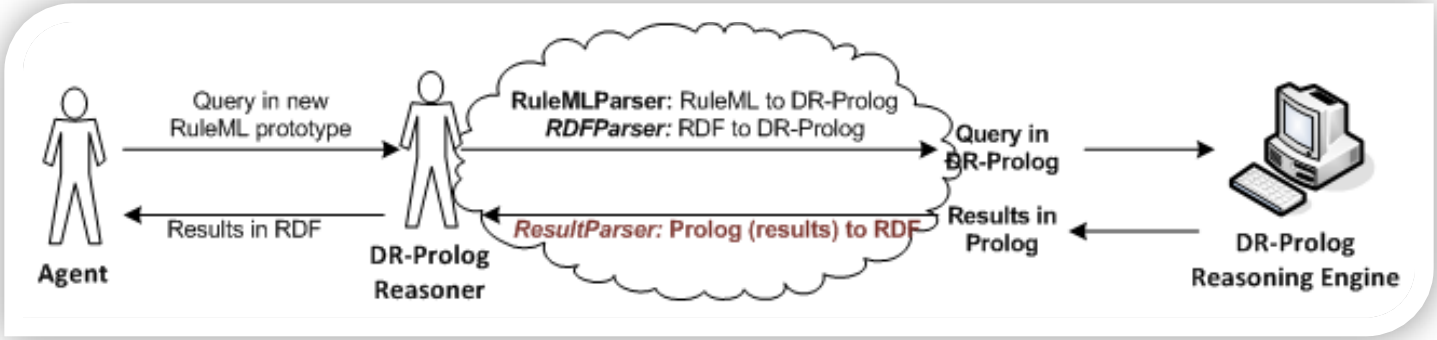
RDFParser

- uses
  - the SW Knowledge Middleware,
  - a set of tools for SW (RDF) KBs

  → parsing, storage, manipulation, querying
- extract the RDF triples
- turn them to Prolog facts

DR-Prolog Reasoner

ResultParser: Prolog to RDF



ResultParser

- receives
  - the initial query (in DR-Prolog)
  - the results (a prolog list)
- returns the query results in RDF

➢ the returned RDF results contain only the results that are required by the initial query and not the complete RB available information

```
<dr-device:acceptable rdf:about="#acceptable0">
        <dr-device:X>a3</dr-device:X>
        <dr-device:Y>350</dr-device:Y>
        <dr-device:Z>65</dr-device:Z>
        <dr-device:W>0</dr-device:W>
</dr-device:acceptable>
```

Overview

- ***Motivation***
- ***EMERALD***
  - ***Reasoners***
- ***DR-Prolog Reasoner***
  - ***RuleMLParser***
  - ***RDFParser***
  - ***ResultParser***
- ***Defeasible Proofing Services***
  - ***Defeasible logic***
  - ***Defeasible Proof Generator Service***
  - ***Proof Validator Service***
- ***Conclusions – Future Work***

Defeasible Logic Intro

- Facts e.g. student(Sofia)
- Strict Rules e.g. student(X) $\rightarrow$ person(X)
- Defeasible Rules e.g. person(X) $\Rightarrow$ works(X)
- Priority Relation between rules

  e.g. r: person(X) $\Rightarrow$ works(X)

  r′: student(X) $\Rightarrow \neg$works(X)

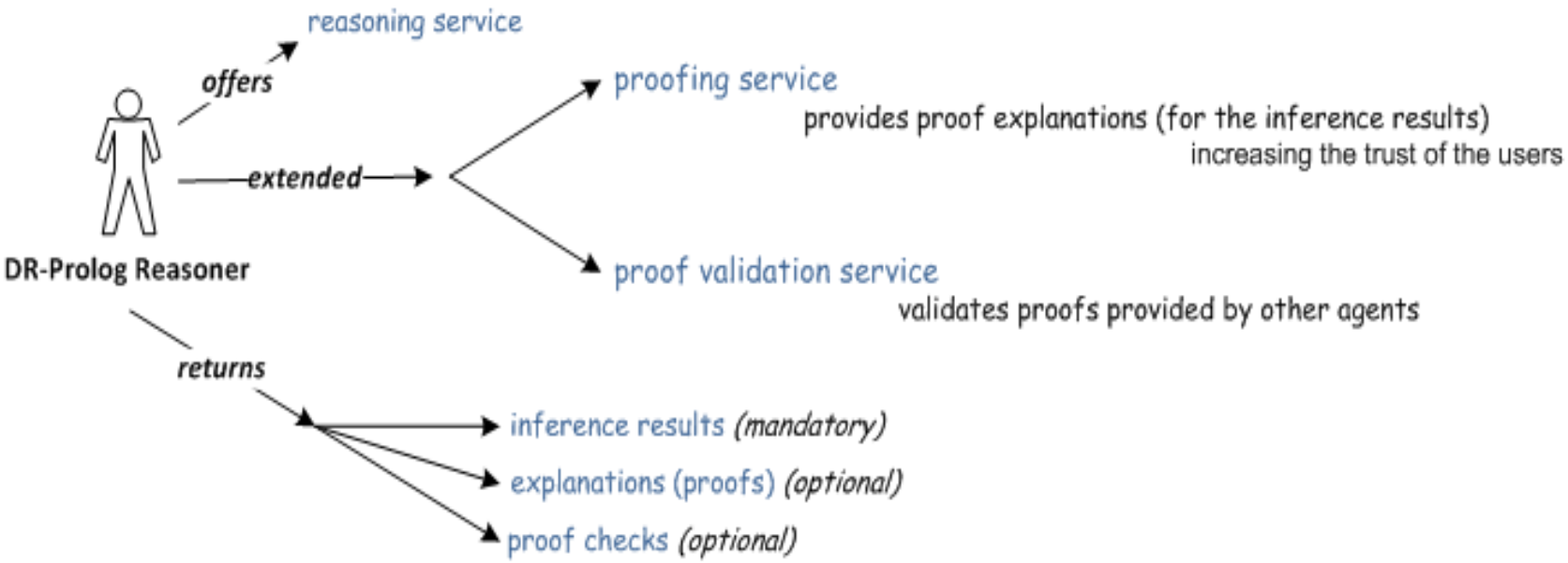
  r′ > r

- *Proof theory example:*
  - A literal q is defeasibly provable if:
    - supported by a rule whose premises are all defeasibly provable AND
    - $\neg q$ is not definitely provable AND
    - each attacking rule is non-applicable or defeated by a superior counter-attacking rule

Defeasible Proofing Services

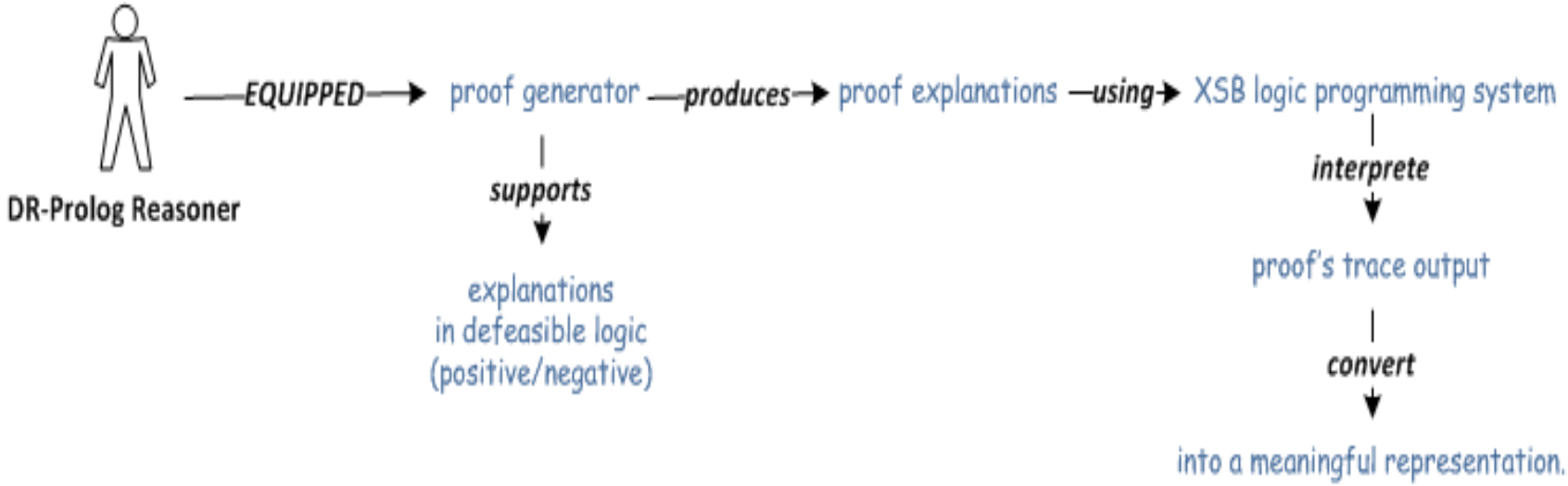
(SW) Proof Layer

- assumed to answer why agents should believe the results
- no W3C recommended technology



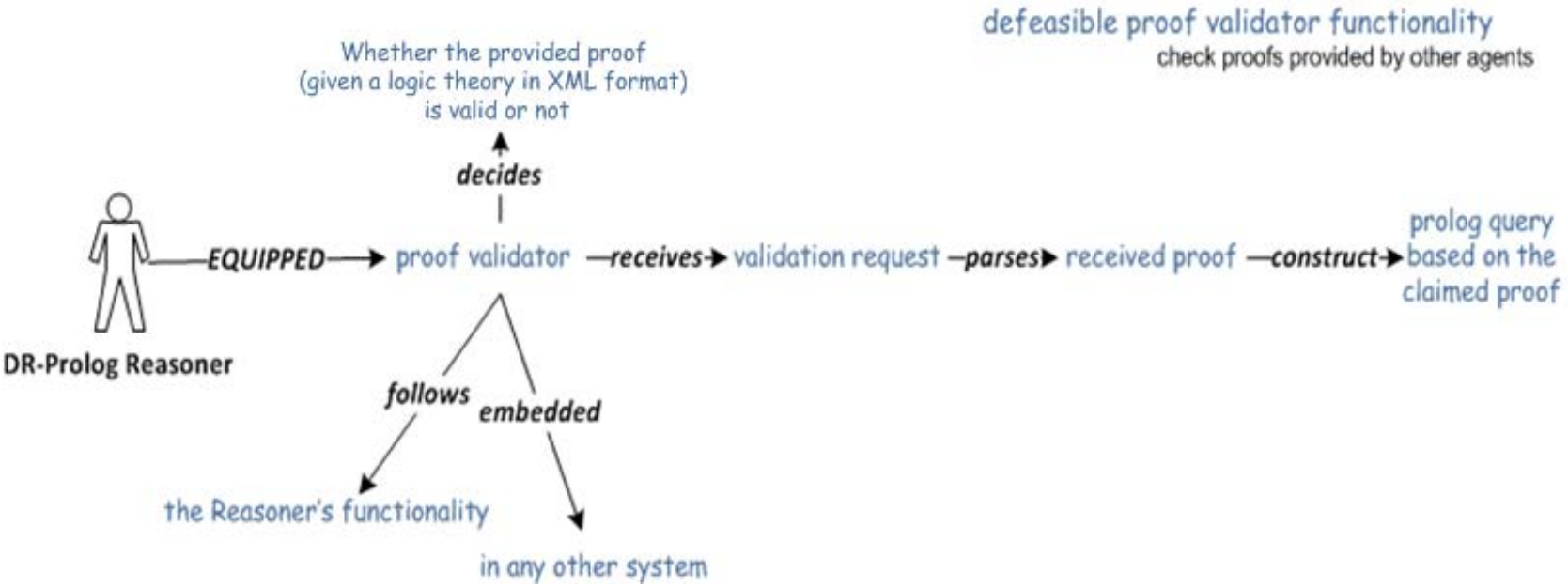
Defeasible Proofing Services

Defeasible Proof Generator Service



Defeasible Proofing Services

Proof Validator Service



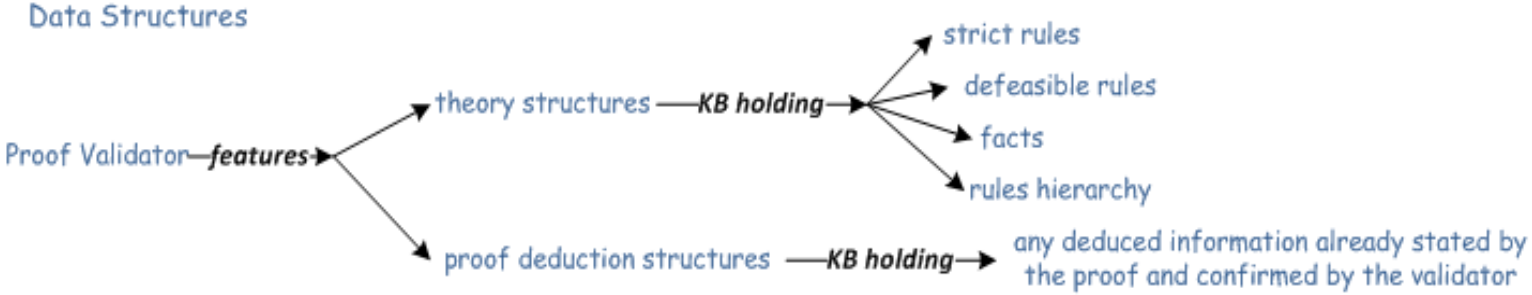
Implementation assumptions:

- The theory is the one given to the proof generator.
  - Any given theory is accepted as valid without any checks.
- No checks are performed recursively.
  - Any information is required in depth more than one a priori.
- Any knowledge facts given in the theory is considered to be definitely provable
  - not taking into account statements present in the proof supporting it.
- The minimal information that will contribute to the proof checking process is required.

Defeasible Proofing Services

Defeasible Proof Validator Service

Data Structures



Example: *a rule that adds knowledge to the definite KB is the following*

```
definitelyCheck(X, printOn) :-
        factkb(F),memberchk(X,F),addDefinitely(X).
```

Facts

all given theory facts are added in the definite knowledge base.

→ any stated fact is by default considered by the proof validator as definitely proved.

Rules

not stated explicitly in the contents of the proof validation result

→ *since they are a priori accepted and considered valid.*

Defeasible Proofing Services

Proof Validator Service

Example:

r1: a => e. r2: b => e. r1 > r3. r2 > r4.

r3: c => ~e. r4: d => ~e. a. b. c. d.

A valid proof:

defeasibly(a), defeasibly(b),

defeasibly(c), defeasibly(d),

defeasibly(e, r2)

Need of proof checks

1. Is there any rule "r2" in the theory, with head equal to e? → *Yes.*
2. Is there any attacking rule? → *Yes, r3, r4.*
   - 2.1 Is r2 of higher priority than r3? → *No.*
     - 2.1.1 Are the conditions of r3 (i.e. c) defeasibly provable? → *Yes.*
     - 2.1.2 Is there any attacking rule of r3 (different from r2), which defeats r3? → *Yes, r1, because its conditions are met and r1 > r3.*
   - 2.2 Is r2 of higher priority than r4? → *Yes.*

Overview

- ***Motivation***
- ***EMERALD***
  - ***Reasoners***
- ***DR-Prolog Reasoner***
  - ***RuleMLParser***
  - ***RDFParser***
  - ***ResultParser***
- ***Defeasible Proofing Services***
  - ***Defeasible logic***
  - ***Defeasible Proof Generator Service***
  - ***Proof Validator Service***
- ***Conclusions – Future Work***

Conclusions – Future Work

A. EMERALD

- a fully FIPA-compliant MAS developed on top of JADE
- proposes the use of trusted, independently-developed reasoning services (REASONERS)

1. Can offer inferencing on a variety of logics
2. Can be used for related services such as
    a) proof explanations on the inference results
    b) Proof validations on exchanged proofs

B. DR-Reasoner

- new types of logic embedded in new Reasoners
- new proofing services designed and embedded

In future:

- Integrate broader variety of reasoning and proof validation engines
- integrate the generated proofs with trust mechanisms

EMERALD available at:
http://lpis.csd.auth.gr/systems/emerald

CS-566 Project available at:
http://www.csd.uoc.gr/~hy566/project2010.html

Thank you!
Any Questions?