Semantic Web Rules

Open Source Prova Projekt

Prof. Dr. Adrian Paschke

Director RuleML Inc.
Director CITT GmbH
Vice-Director STI Berlin

Adrian.Paschke@gmx.de

RuleML
Realize your Knowledge

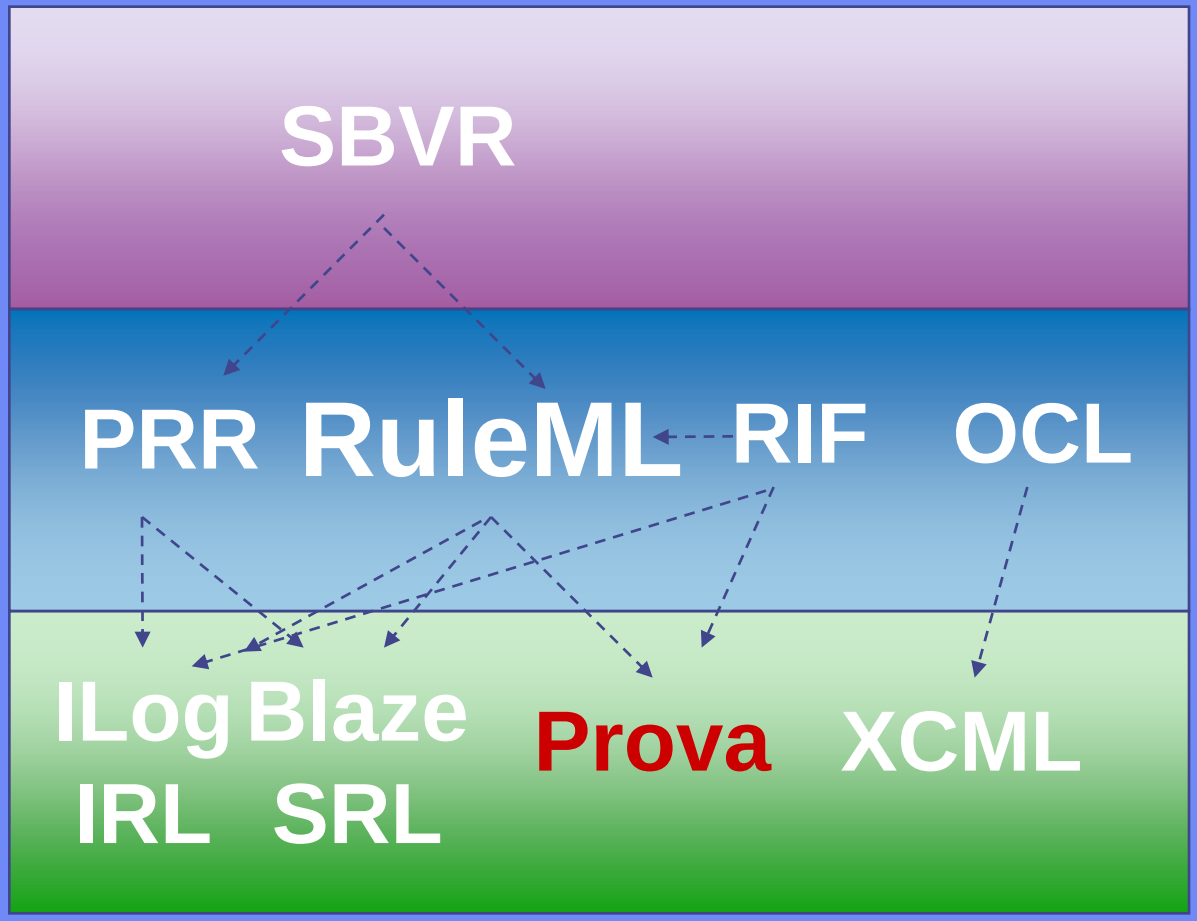
CSW
Corporate Semantic Web

Freie Universität Berlin

PROVA

Distributed Semantic Web Rule Engine

http://prova.ws



Was ist Prova?

- Sourceforge Open-Source Projekt (http://www.prova.ws/) seit 2003
  - Aktivität 86.69% **;** Downloads 28,529; 10 new requests per month, 254 registered users
  - Anwendungsgebiete:
    - **Xalia Core for Services - Resource / Service Allocation: Prova used for** efficiently computing global execution plans http://xdn.xcalia.com/xdn/docs/files/XcaliaCore/4.3.0/documents/html/introduction-4.htm
    - W3C Semantic Web Rule Interchange Format: http://www.w3.org/2005/rules/wg/wiki/List_of_Rule_Systems
    - **Rule-based IT Service and Contract Management: RBSLA Projekt** http://ibis.in.tum.de/projects/rbsla/index.php
    - Rule Responder Pragmatic Agent Web http://ibis.in.tum.de/projects/paw/

- Distributed Semantic Web Rule Engine and Agent/Service-Oriented Architecture:

  "*Prova follows the spirit and design of the recent W3C Semantic Web initiative and combines declarative rules, ontologies and inference with dynamic object-oriented programming and access to external data sources via query language built-ins.*"

Prova

- Prinzipien
  - Trennung von *Logik* (declarative), *Datenzugriff* (procedural attachments and query languages) und *Berechnung* (inference and external procedural computation)
- Verbindet deklarative Programmierung (Regeln) mit OO-Programmierung (Java) und Workflowsprachen (z.B., BPEL)
- Standard ISO Prolog Scripting Syntax +
  - Integration von Java, Semantic Web Sprachen (RDF,RDFS, OWL, RIF) und Anfragesprachen (z.B. SPARQL, XPath/XQuery, SQL)

Prova – Ausgewählte Eigenschaften

- External Data and Object Integration + Query Built-Ins
  - Java Integration
  - XML Integration
  - SQL Integration
  - RDF Integration
- External Type Systems: Order-Sorted Polymorphic Typed Logic
  - Java Class Hierarchies
  - Semantic Web Ontologies
- Input/Output Mode Declarations
- Module Import and Integration: Order Modularized Logic Programs
- Meta Data Labels and Scopes (constructive views)
- Integrity Constraints and Test Cases for Verification and Validation
- Backward-reasoning Derivation rules + ECA-style rules
- Messaging Reaction Rules and Complex Event Processing
- Dynamic Transactional Updates

Prova Beispiel

Prova extends ISO Prolog syntax:

```
% Facts

is_a("anticoagulant","molecular_function").

% Clauses (head true if conditions true in the body)
parent(X,Y) :-
   is_a(Y,X).
parent(X,Y) :-
   has_a(X,Y).

% Goals (note there is no head)
:- solve(parent(Parent,"anticoagulant")).      % Print solutions
:- eval(parent(Parent,"anticoagulant")). % Just run exhaustive search
```

Format of the output for the goal *solve* above:

```
   Parent="molecular_function"
```

Java Method Calls

- Constructors, instance and static methods, and public field access;

- Ability to embed Java calls makes the Prolog-like programming style more suitable for integration and computation workflows.

```
hello(Name):-
   S = java.lang.String("Hello").
   S.append (Name),
   java.lang.System.out.println (S).
```

External Data and Object Integration

- File Input / Output

```
..., fopen(File,Reader), ...
```

- XML (DOM)

```
document(DomTree,DocumentReader) :-
    XML(DocumenReader),
```

- SQL

```
...,sql_select(DB,cla,[pdb_id,"1alx"],[px,Domain]).
```

- RDF

```
...,rdf(http://...,"rdfs",Subject,"rdf_type","gene1_Gene"),
```

- XQuery

```
..., XQuery = ' for $name in
StatisticsURL//Author[0]/@name/text() return $name',
xquery_select(XQuery,name(ExpertName)),
```

- SPARQL

```
...,sparql_select(SparqlQuery,name(Name),class(Class),
definition(Def)),
```

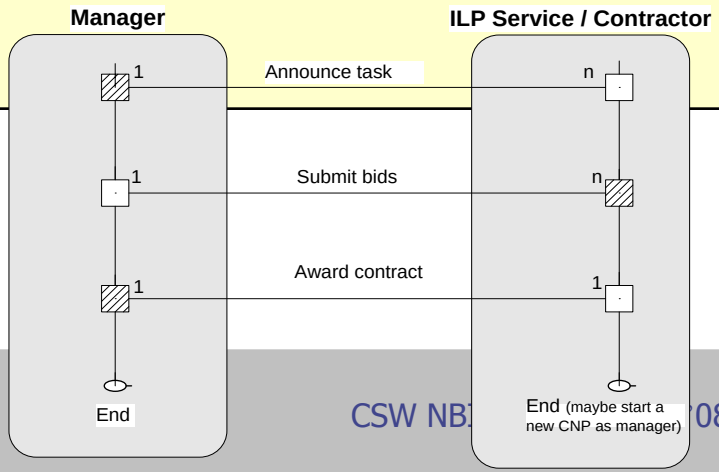
Beispiele: File I/O und SPARQL

```
test_fopen() :-
    fopen(File,Reader),
    % Non-deterministically enumerate lines in the file
    read_enum(Reader,Line),
    println([Line]).        % Print one line at a time
```

```
exampleSPARQLQuery(URL,Type|X) :-
   QueryString =
   ' PREFIX foaf: <http://xmlns.com/foaf/0.1/>
   PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
     SELECT ?contributor ?url ?type
     FROM <http://planetrdf.com/bloggers.rdf>
     WHERE {
       ?contributor foaf:name "Bob DuCharme" .
       ?contributor foaf:weblog ?url .
       ?contributor rdf:type ?type . } ',
   sparql_select(QueryString,url(URL),type(Type)|X),
   println([[url,URL],[type,Type]|X],",").
```

Beispiel: „Mobile Rule Code“

```
% Manager
upload_mobile_code(Remote,File) :
   Writer = java.io.StringWriter(), % Opening a file
   fopen(File,Reader),
   copy(Reader,Writer),
   Text = Writer.toString(),
   SB = StringBuffer(Text),
   sendMsg(XID,esb,Remote,query-ref,consult(SB)).
% Service (Contractor)
rcvMsg(XID,esb,Sender,eval,[Predicate|Args]):-
   derive([Predicate|Args]).
```



Contract Net Protocol

Entwicklung Regel-basierter Systeme

Grundlagen

- V&V&I Pattern:

  Validation: ”Entwickeln wir das richtige Programm*?*”

  Verification: ”Entwickeln wir das Programm richtig*?*”

  Integrity: ”Führen wir das Programm richtig aus*?*”

- Fehler und Anomalien
  - Fehler sind Probleme welche direkt die Ausführung von Regeln beeinflussen, z.B. typographische Fehler, Unvollständigkeit, Wiedersprüche, …
  - Anomalien sind Symptome von wirklichen Fehlern

- Taxonomy of Anomalies (Preece and Shinghal)
  - Semantische Überprüfung z.B. Konsistenz, Vollständigkeit
  - Strukturelle Überprüfung, z.B. Redundanz, Relevanz, Erreichbarkeit

Motivation

- Software Development Lifecycle (SDLC) für viele Open-Source und kleinere Projekte ungeeignet
  - Kostenintensiv, langsam, schwer zu verwalten und zu ändern, bei ständig wechselnden Anforderung
  - Unterschiedliche Anforderungen durch verteilte Web-basierte Systeme oft über Domänengrenzen hinweg

Vorgeschlagene Lösungen:

1. Agile SE (Extreme Programming + andere)
   - Beschleunigen Entwicklungsprozess und unterstützen Redesign
   - Empirische Beweise, dass XP für kleine und mittlere Projekte besser geeignet Layman, L.: "Empirical Investigation of the Impact of Extreme Programming Practices on Software Projects,"
2. Regel-basierte Systeme: Entwicklung von Werkzeugen welche Geschäftsleute befähigen IT Dienste und Geschäftsprozesse flexibel zu ändern
   - Vermeidung des langsamen SDLC
   - Neue Welle an kommerziellen Tools (Regel Engines, CEP-Engines, Workflow Engines).

Könnten 1. + 2. vereint werden ?

Problem: Komplexität von Regelsprachen

- Anzahl der Primitiven in Regelsprache
  - z.B. Anzahl der benutzen Konvektive
- Tiefe des Ableitungsbaums
  - z.B. begrenzt für Logische Programme ohne Funktionssymbole, aber unbegrenzt wenn Konvektive und Terme verschachtelt sind
- Restriktionen der Regelsprache
  - z.B. keine Negation im Regelkopf
- Keine allgemeingültige Sprachelemente und prozedurale Elemente
  - z.B. Prioritäten, Cut, Unterschiedliche Arten von logischen und prozeduralen Konjunktionen, procedural attachments, …
- Unterschiedliche Sprachelemente mit ähnlicher Bedeutung
  - schwache und starke Negation
  - *OR* und *XOR*
- Polymorphische Sprachelemente
  - z.B. polymorphic negation, polymorphic order-sorted types
- Keine Standardinterpretation von bestimmten Sprachelementen
  - z.B. Arten von Modal Logic, Deontische Modalitäten, Negation as Failure
- Kreuzreferenzen zwischen Regeln
  - z.B. Schleifen im Abhängigkeitsgraph zwischen Prädikatssymbolen

Lehren aus dem Software Engineering

- Unterschiedliche SE Ansätze und Methoden
  1. „Schwergewichtige“ Ansätze
     - z.B. Wasserfall-Model, V-Model
     - Führen zu hohen Änderungskosten
     - Können dynamische Verhalten nicht überprüfen und Interaktionen zwischen dynamisch geänderten und ausgetauschten Regelbasen
  2. Einfaches strukturelles / Operationales Debugging
     - Führen Programm aus und überwachen Ausführungsablauf (execution trace)
     - Hohe kognitive Belastung und tiefes Verständnis der zugrunde liegenden Prozesse auf Seiten des Entwicklers nötig
  3. Model Checking, Algebraisch, Graph-basiert, Petri-Net-basierte Methoden
     - z.B. Petri-Nets, Process Algebras, ACTL
     - Sehr Rechen-aufwendig
     - Setzen tiefes Verständnis beider Domänen voraus: Regelsprache und Testsprache / Verifikationsmodelle
     - Oft weitaus komplexer als das Regelprogramm selbst

XP – Test Driven Development

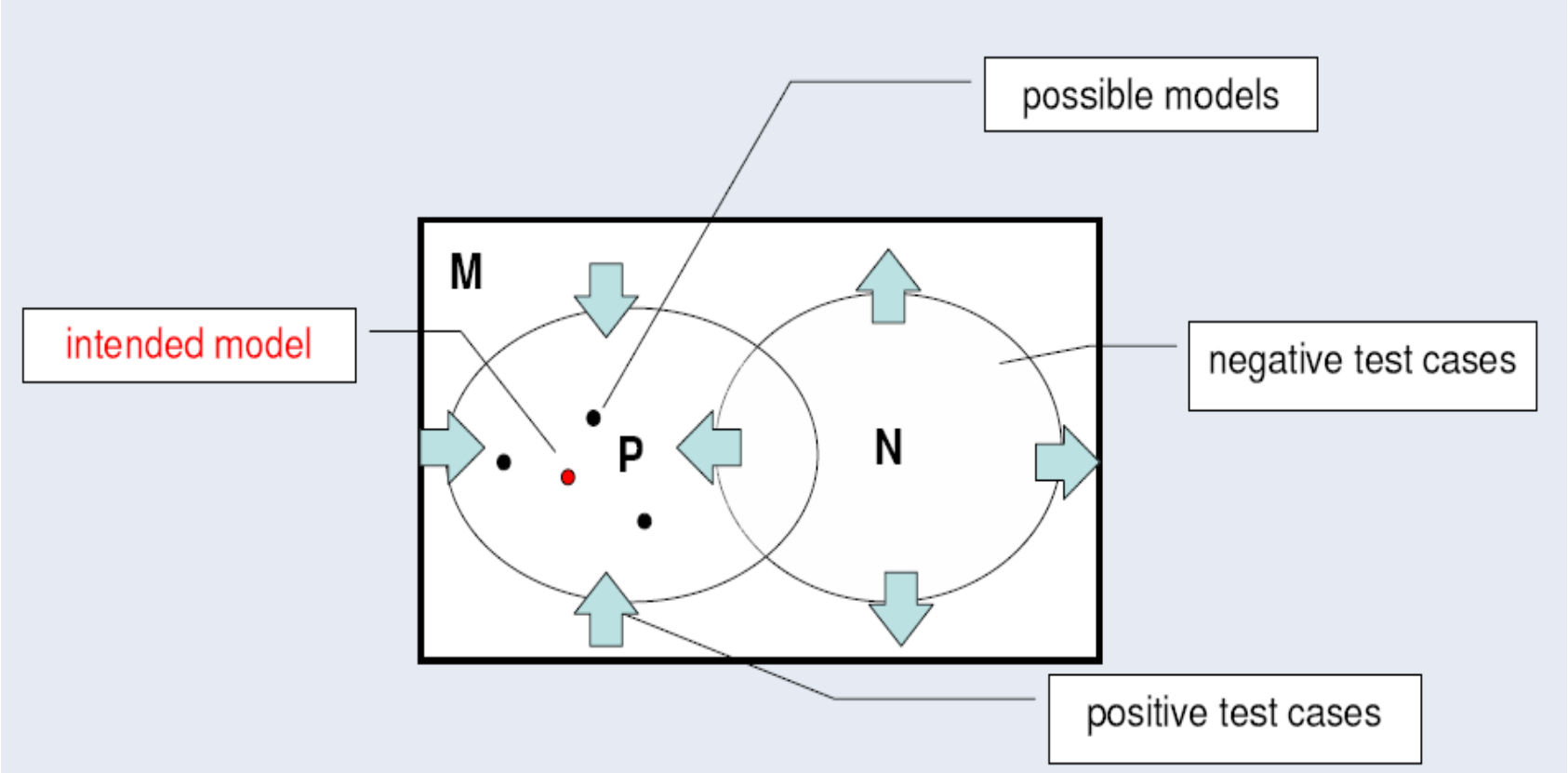
- Eingeführt in den späten 90'ern
- Test-driven Extreme Programming und Agile SE
  - Empirische Beweise, dass XP für kleine und mittlere Projekte besser geeignet
    Layman, L.: "Empirical Investigation of the Impact of Extreme Programming Practices on Software Projects,"
  - Unterstützung durch Industrie, z.B. IBM
  - Vernachlässigt umfangreiche Analyse und Design ist aber trotzdem ein formaler Ansatz
  - Beschleunigt Entwicklungsprozess, Unterstützt "Backtracking" (redesign), Kollaboration von Rollen (Experten, Systementwickler, …), evolutionärer iterativer Ansatz, Unterstützt Regelaustausch und dynamisches Testen

Test Cases für Regelbasen

- Regeln beschränken die Menge an möglichen Modellen (possible worlds)
  - Aber sind relativ komplex (wie vorher argumentiert)
- Test Cases beschränken die möglichen Modelle und entsprechen ungefähr den beabsichtigten Modellen des Entwicklers
  - Vordefinierte Anfragen werden benutzt um die Regelbasis zu testen
  - Äußerst einfach, z.B. keine Variablen, keine Funktionen, keine Negation
- Ein Test Case ist definiert als TC := {A, T_i}, wobei
  - $A \subseteq L$ assertion base (Eingabe Daten, z.b. Fakten)
  - $T_i \in L$, i>0, Testanfragen, wobei T_i:
    1. Eine Testanfrage Q := q(t1, ..tn)?, wobei $Q \in \mathrm{rule}(P)$
    2. Ein Ergebnis R mit wahr "*true*", falsch "*false*" oder "*unknown*" Auszeichnung
    3. Antwortmenge mit erwarteten Variablenbindungen
       $\theta := \{\{X_1/a_1, X_1/a_2, \ldots, X_1/a_n\}, \ldots, \{X_m/c_1, \ldots, X_m/c_k\}\}$
- T = A $\cup$ {Q => R :θ } oder einfach T = {Q => R :θ }
- Beispiel:

  T1 = {p(X) => true : {X/a, X/b, X/c}, q(Y) => false}

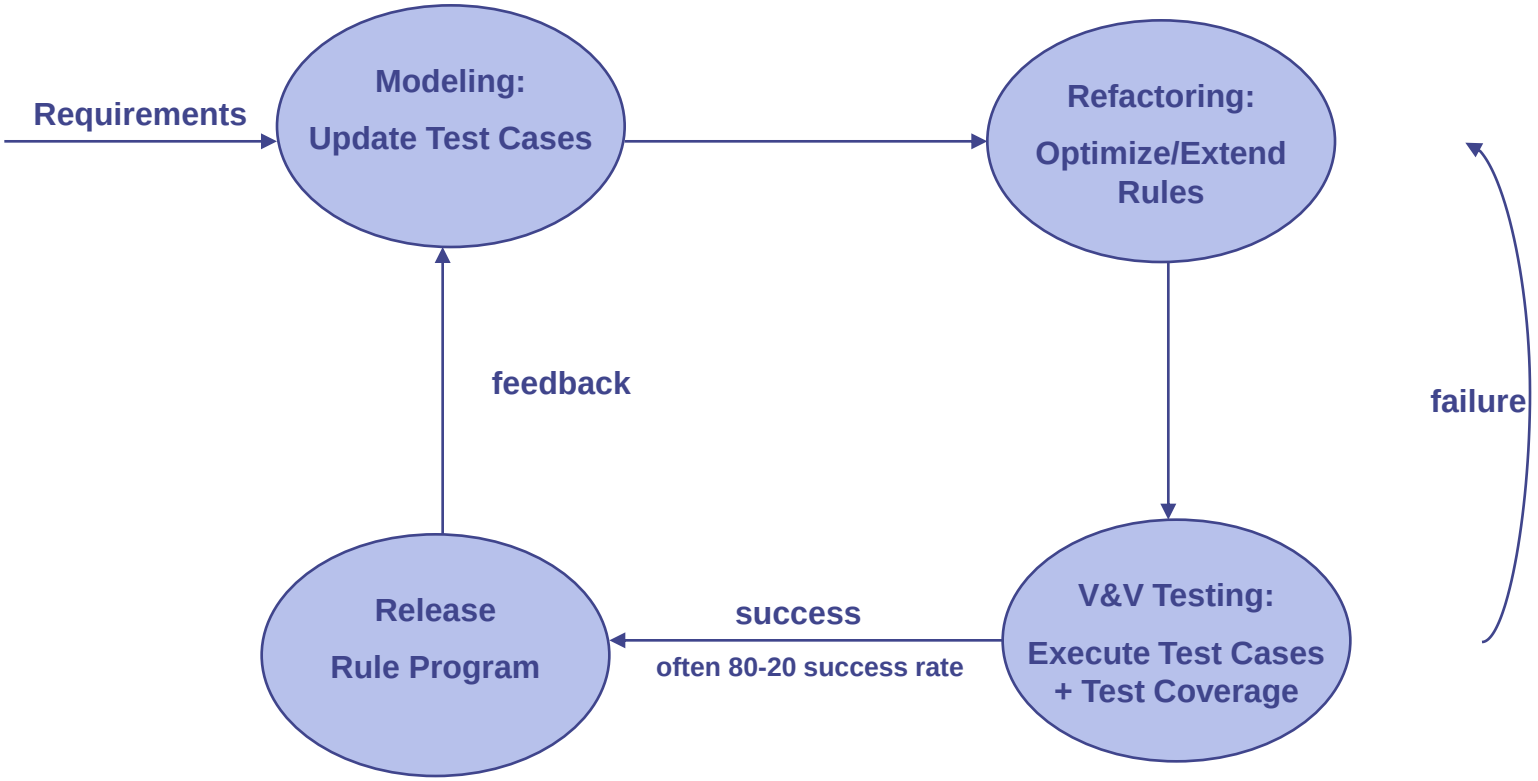
Test Cases für Regelbasen



Test-basierte Entwicklung

1. Schreibe ausführbare Test Cases zuerst
   - In der selben Regelsprache
   - Black Box Sicht
   - Abstraktion vom Regelprogramm -> viel Einfacher
   - Wiederverwendbar für andere Regelprogramme
2. Geringes Vorausdesign aber sich entwickelndes Design. Permanentes Redesign durch Refactoring
   - Optimierung der Regeln ohne Veränderung des Verhaltens
   - Modifizierung und Anpassung der Test Cases und des Programms in einem iterativen Prozess entsprechend der sich verändernden Anforderungen und entdeckter Fehler
3. Kurze Iterationen
   - Tool-Unterstützung für Test, Coverage Measurement, Automated Refactoring, Dependencies Analysis
4. Tests sind eine Annährung an die erwarteten Modelle (erwartete Antworten des Programs)
   - Die Annährung (test coverage) wird dynamisch gesteigert (adaptive modeling)
   - Besser geeignet um die Dynamik von offenen Projekten und Web-basierter Anwendungen zu erfassen als Ansätze mit großem, teuren Vorausdesign
5. Test Cases werden zusammen mit den Regelprogrammen geschrieben, gepflegt, verwaltet und ausgetauscht; in der selben deklarativen Regelsprache

Test-driven Development Feedback Loop



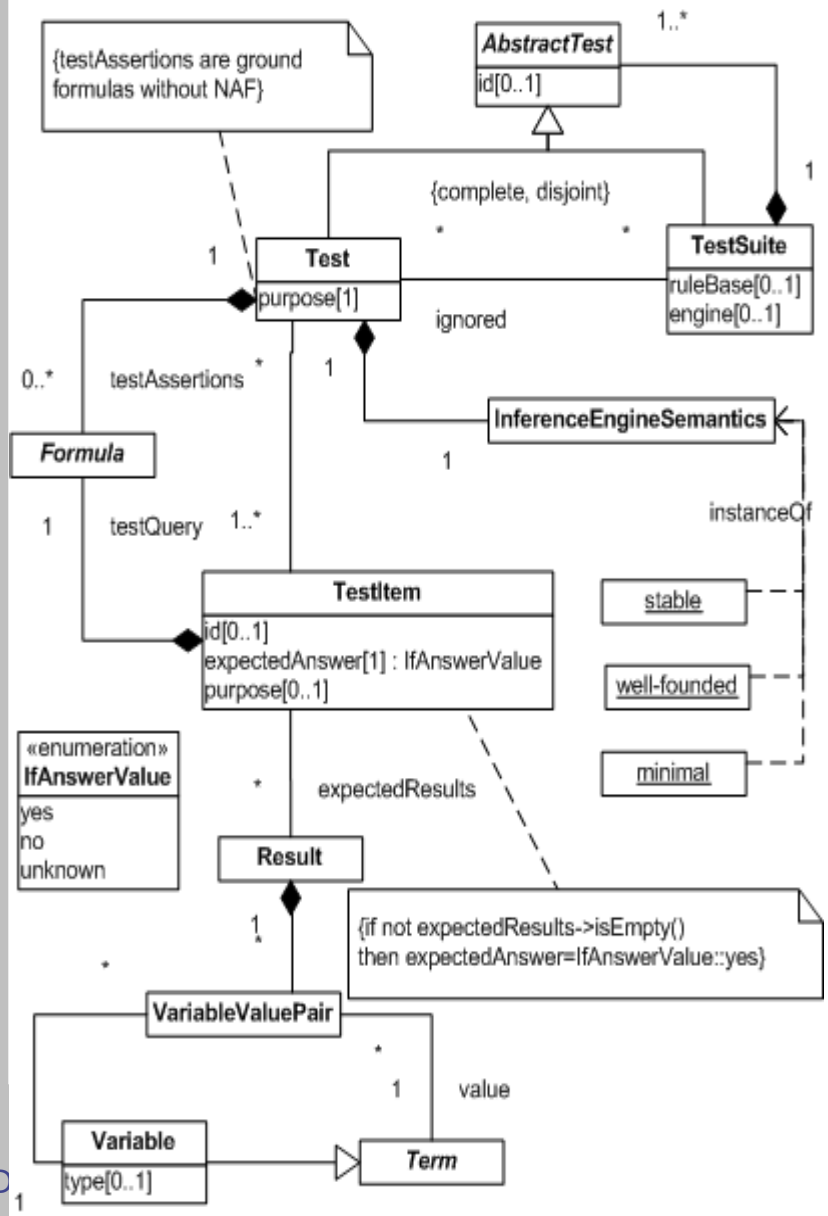
Beispiel Test Case in Prova

```
% testcase oid
testcase("./examples/tc1.test").

% assertions via updates adding one rule and two facts
:-solve(add("tc1.test","a(X):-b(X). b(1). b(2).")).

% positive test with success message for JUnit report
testSuccess("test1","succeeded"):-
  testcase(./examples/tc1.test), testQuery(a(1)).

% negative test with failure message for Junit report
testFailure("test1","can not derive a"):-
  not(testSuccess("test1",Message)).

% define the active tests - used by meta program
runTest("./examples/tc1.test"):-testSuccess("test
  1",Message).
```

Prova - LPCover Werkzeug

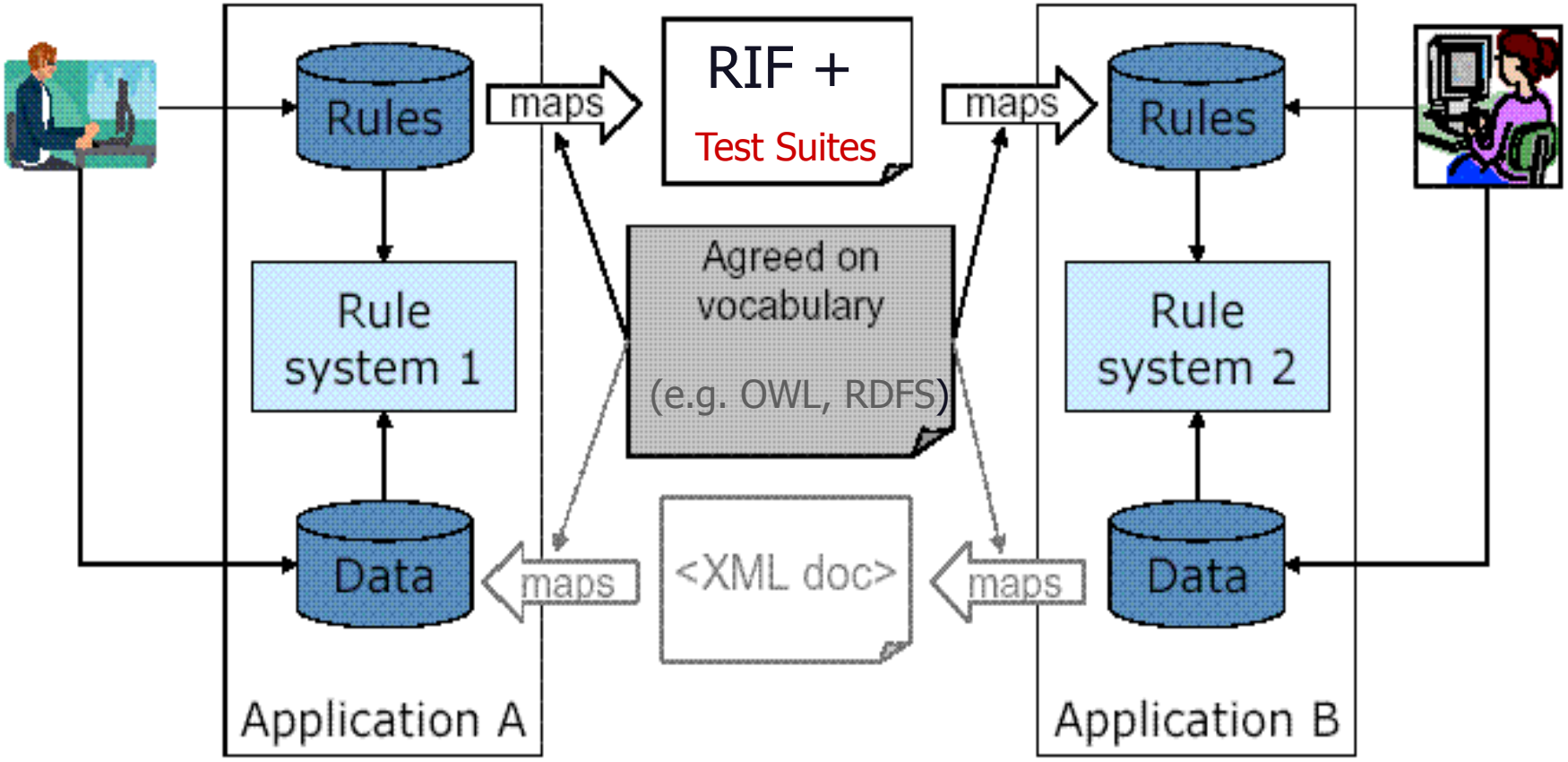
- JUnit Test Reports
- Test Coverage Measurement
- Automated Refactoring
  - Basierend auf Smells (structures which need to be improved), z.B.
    - Redundancy – There are redundant rules or rule fragments (for instance, shared subsets of prerequisites).
    - Inconsistency – Different, inconsistent results are supported by the same rule set.
    - Incompleteness – Certain queries can not be answered by a rule set.

Test Cases / Test Suites in RuleML




```
<TestSuite ruleBase="SampleBase.xml">
   <Test id="ID001" purpose="...">
       <testAssertions><RuleMLFormula>
          <ruleml:And>
            <ruleml:Atom>

   <ruleml:Rel>parent</ruleml:Rel>
              <ruleml:Ind>John</ruleml:Ind>
              <ruleml:Ind>Mary</ruleml:Ind>
          </ruleml:Atom>
        </ruleml:And>
      </RuleMLFormula></testAssertions>
  <TestItem expectedAnswer="yes">
      <testQuery><RuleMLFormula>
        <ruleml:Atom closure="universal">
          <ruleml:Rel>uncle</ruleml:Rel>
          <ruleml:Ind>Mary</ruleml:Ind>
          <ruleml:Var>Nephew</ruleml:Var>
        </ruleml:Atom>
      </RuleMLFormula></testQuery>
      <expectedResults>
        <VariableValuePair>
            <ruleml:Var>Nephew</ruleml:Var>
            <ruleml:Ind>Tom</ruleml:Ind>
        </VariableValuePair>
        <VariableValuePair>
            <ruleml:Var>Nephew</ruleml:Var>
            <ruleml:Ind>Irene</ruleml:Ind>
        </VariableValuePair>
    </expectedResults>
  </TestItem>

    <InferenceEngineSemantics>minimal</Infere
    nceEngineSemantics>
</Test></TestSuite>
```

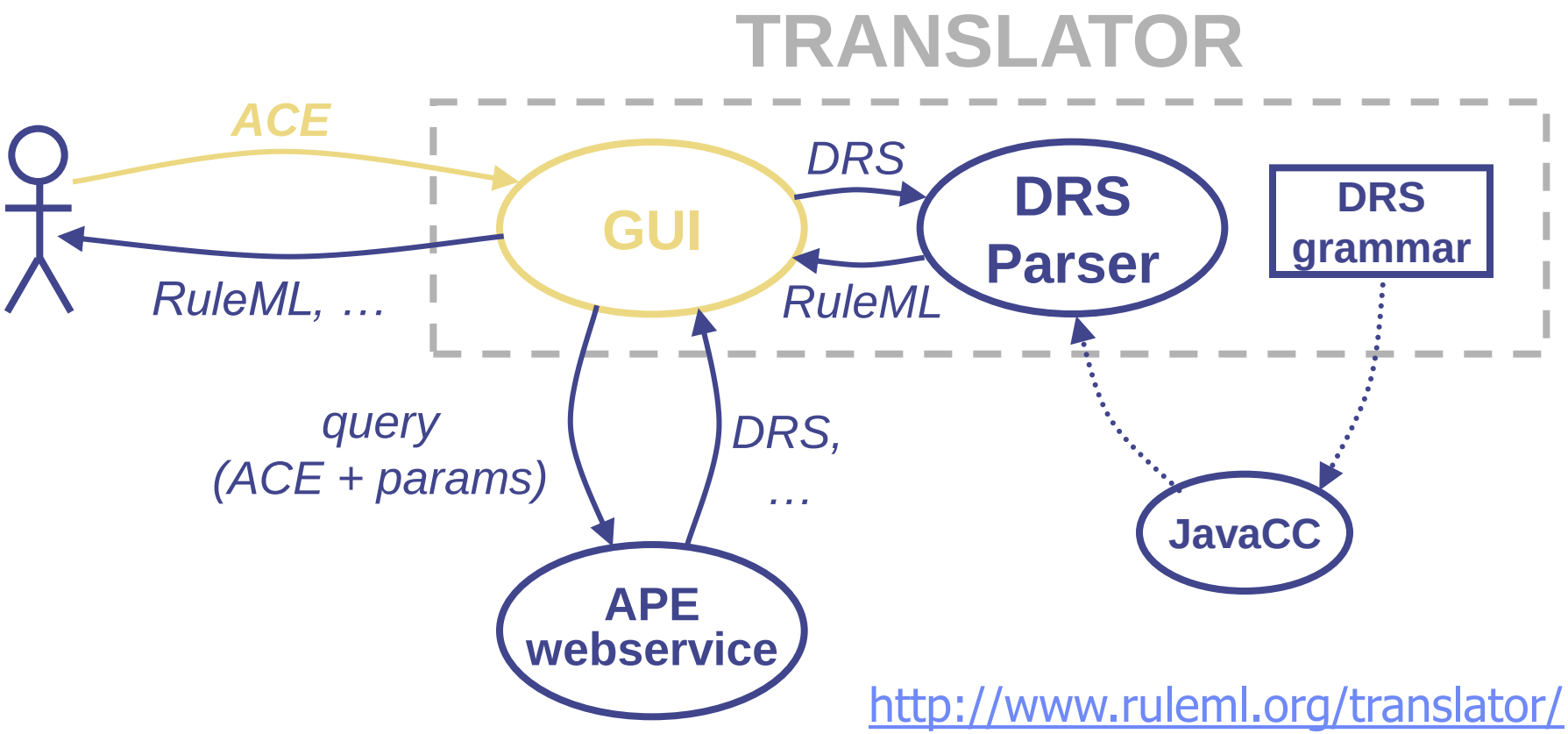
Regelaustausch: Selbst-validierende Regelbasen



Entwicklung von Regelbasen

- **Computational Independent Model (CIM)**
  - Natürliche Sprache
    - z.B. Controlled English
  - Template-driven, Design Patterns
    - Vordefinierte Templates, Blueprints, Vokabulare, Design Patterns
  - Visuelle Sprache
    - Graphisch, z.B. UML-basiert

Attempto Controlled Englisch Translator



http://www.ruleml.org/translator/

Every honest student who does not procrastinate receives a good mark and easily passes the course.

(ACE)

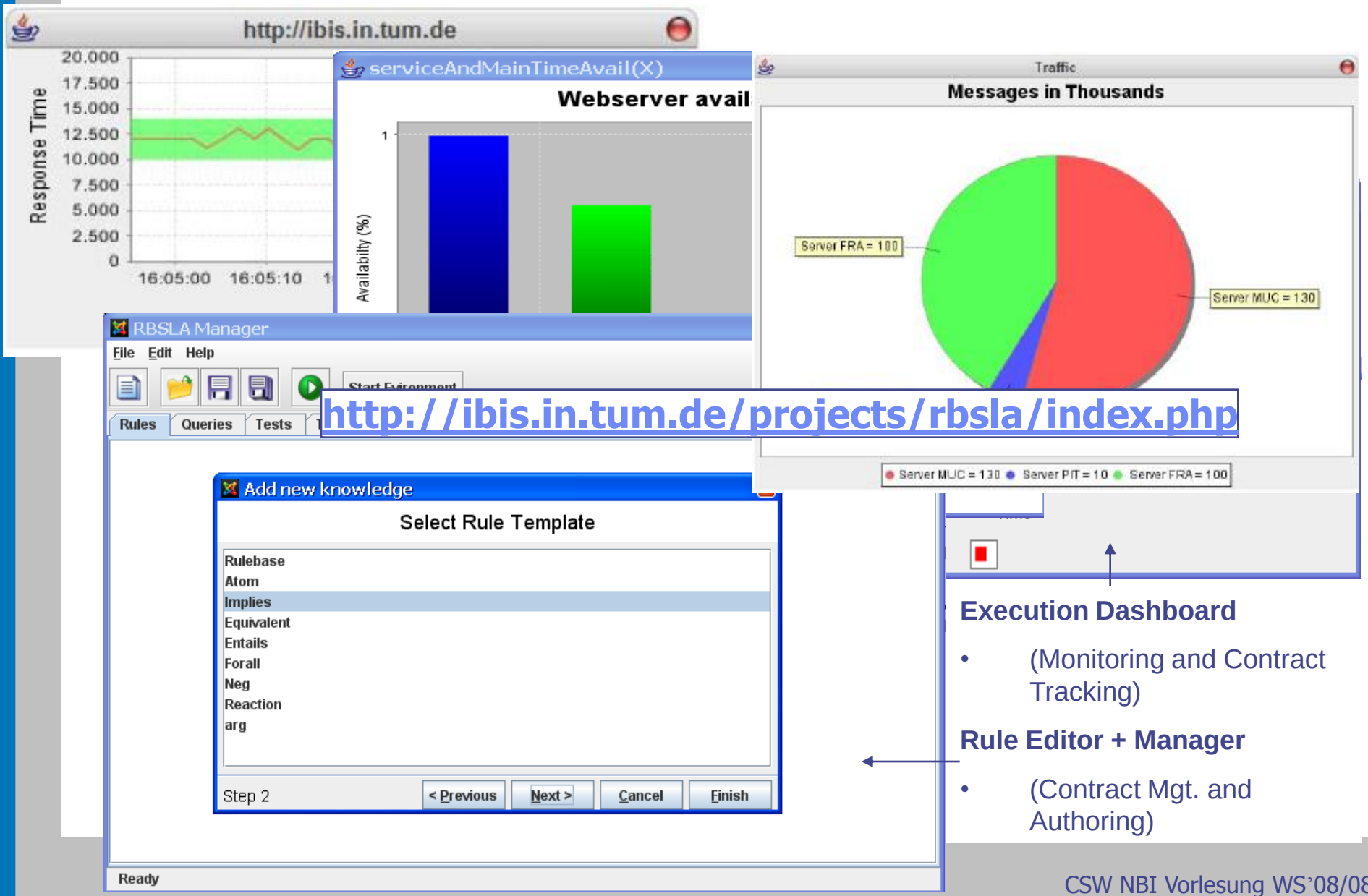

```
[]
   [A]

   object(A, … student, …)
   property(A, honest)
      NOT

      [B]
      predicate(B, … procrastinate, A)

   =>

   [C, D, E, F]

   object(C, … mark, …)
   property(C, good)
   predicate(D, … receive, A, C)
   predicate(E, … pass, A, F)
   modifier(E, manner, … easily)
   object(F, … course, …)
```


(DRS)

```
…
<Forall>
 <Var>A</Var>
 <Implies>
  <And>
   <Atom><Rel>object</Rel>…<Ind>student</Ind>…</Atom>
   <Atom><Rel>property</Rel>…<Ind>honest</Ind></Atom>
   <Neg>
    <Exists>
     <Var>B</Var>
     <Atom><Rel>predicate</Rel>…<Ind>procrastinate</Ind> …
    </Exists>
   </Neg>
  </And>
  <Exists>
   <Var>C</Var><Var>D</Var><Var>E</Var><Var>F</Var>
   <And>
    <Atom><Rel>object</Rel>…<Ind>mark</Ind>…</Atom>
    <Atom><Rel>property</Rel>…<Ind>good</Ind></Atom>
    <Atom><Rel>predicate</Rel>…<Ind>receive</Ind>…</Atom>
    <Atom><Rel>predicate</Rel>…<Ind>pass</Ind>…</Atom>
    <Atom><Rel>modifier</Rel>…<Ind>easily</Ind></Atom>
    <Atom><Rel>object</Rel>…<Ind>course</Ind>…</Atom>
   </And>
  </Exists>
 </Implies>
</Forall>
…
```

(RuleML)

Templates, Patterns, Vocabularies



Execution Dashboard

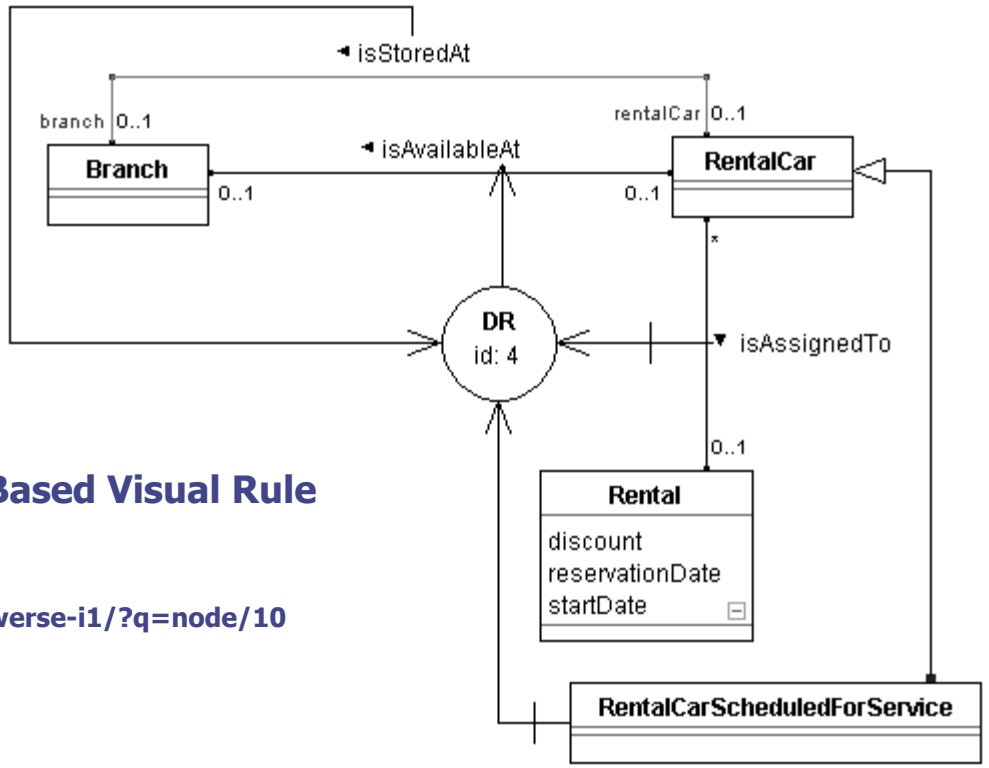
- (Monitoring and Contract Tracking)

Rule Editor + Manager

- (Contract Mgt. and Authoring)

UML-basierte Visualisierung

If a rental car is stored at a branch, is not assigned to a rental and is not scheduled for service, then the rental car is available at the branch.



REWERSE Strelka -- An URML-Based Visual Rule Modeling Tool

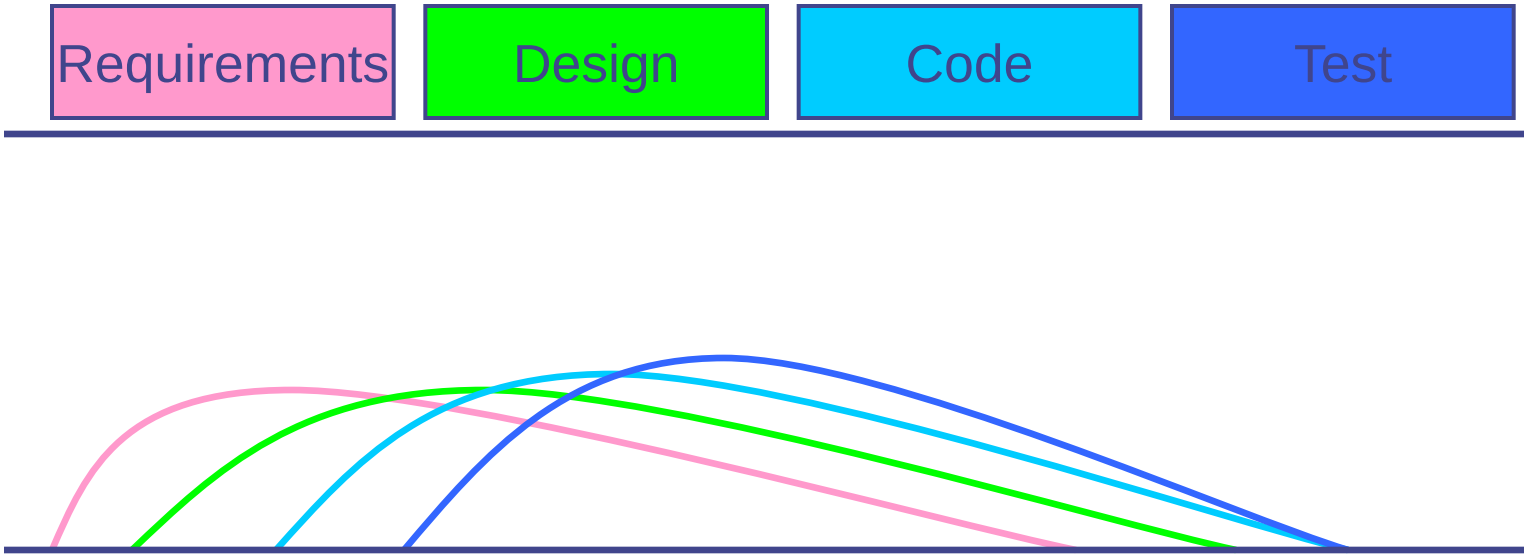
http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/10

Exkurs: Open Source Projektentwicklung Agiles Manifest – das Wertesystem

- **Individuen und Interaktionen** gelten mehr als Prozesse und Tools
- **Funktionierende Programme** gelten mehr als ausführliche Dokumentation
- Die **stetige Zusammenarbeit** steht im Vordergrung
- Der **Mut und** die **Offenheit für Änderungen** steht über dem Befolgen eines festgelegten Plans

http://www.agilemanifesto.org

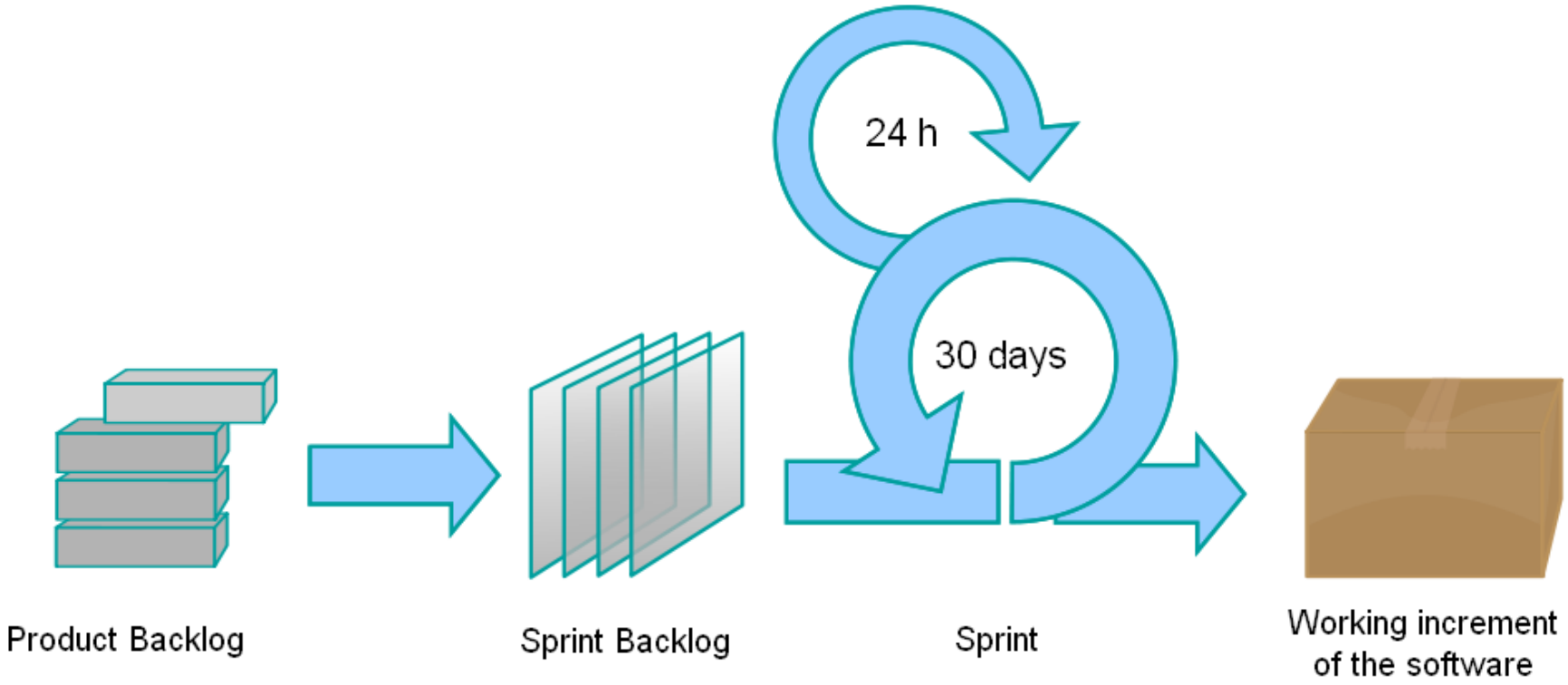
Sequentielle vs. überlappende Entwicklung



SCRUM

- Einer der „agilen Prozesse“
- Selbst-Organisierende Teams
- Produkt schreitet in Serien von monatlichen “Sprints” fort
- Anforderungen sind als Listeneinträge im “Produkt-Backlog” festgehalten
- Keine spezifischen Entwicklungsvorgehen vorgeschrieben
- Benutzt generative Regeln um ein agiles Umfeld für die Auslieferung von Produkten zu schaffen

SCRUM Überblick



Sprints

- Scrum-Projekte schreiten in Serien von “Sprints” voran
  - Analog zu XP-Iterationen
- Angestrebte Dauer ist ein Monat
  - +/- ein oder zwei Wochen
    - Eine konstante Dauer führt zu einem besseren Rhythmus
- Das Produkt wird während des Sprints entworfen, kodiert und getestet

Links

- RuleML http://www.ruleml.org/
- Reaction RuleML http://ibis.in.tum.de/research/ReactionRuleML/
- W3C RIF http://www.w3.org/2005/rules/wiki/RIF_Working_Group
- OMG PRR http://www.omg.org/docs/dtc/07-11-04.pdf
- OMG SBVR http://www.omg.org/spec/SBVR/1.0/
- Rule Responder http://responder.ruleml.org/
- Prova
  - http://prova.ws/
  - Paschke, A.: Rule-Based Service Level Agreements - Knowledge Representation for Automated e-Contract, SLA and Policy Management, ISBN 978-3-88793-221-3, Idea Verlag GmbH, München, 2007.
- V&V&I of Rule Bases
  http://www.biotec.tu-dresden.de/~adrianp/docs/SWPW06_Validation.pdf

Vielen Dank für die Aufmerksamkeit