HeKatE

GEIST

Institute of Automatics
AGH University of Science and Technology, POLAND

Hybrid Knowledge Engineering

http://hekate.ia.agh.edu.pl

Outline

1. Rule–Based Systems
2. HeKatE Approach
3. HaDEs
4. Integration

Rule–Based Systems

Features

- **RBS** → type of expert system
- knowledge representation → **rules**
- state of the system → **factbase**

Rules

- easy to understand and intuitive
- high-level declarative knowledge representation
- logical independence of applications
- constis of **only** two parts: *condition* and *decision*

HeKatE Approach

Features

- formal logical system description $ALSV(FD)$
- three stages of hierarchical design
- $ARD+$ $\rightarrow$ rule prototyping
- XTT^2 $\rightarrow$ logical system design
- automated implementation $\rightarrow$ prototype generation
- **formal on-line** verification

Conceptual Design – *ARD+*

ARD+

- *ARD+* → Attribute Relationship Diagram
- supportive method for XTT2
- hierarchical method
- general model $\xrightarrow{\textit{transformation}}$ more detailed model
- the most detailed model $\xrightarrow{\text{automatic transition}}$ schema of XTT2

ARD+ syntax

Definition

Conceptual Attribute. A conceptual attribute A is an attribute describing some general, abstract aspect of the system to be specified and refined.

Definition

Physical Attribute. A physical attribute a is an attribute describing a well-defined, atomic aspect of the system.

ARD+ syntax

A *property* is described by one or more attributes.

Definition

Simple Property. PS is a property described by a single attribute.

Definition

Complex Property. PC is a property described by multiple attributes.

Definition

Dependency. A dependency D is an ordered pair of properties $D = \langle p_1, p_2 \rangle$ where p_1 is the independent property, and p_2 is the one that dependent on p_1.

ARD+ syntax

Definition

Diagram. An ARD diagram G is a pair $G = \langle P, D\rangle$ where P is a set of properties, and D is a set of dependencies.

Constraint

Diagram Restrictions. The diagram constitutes a directed graph with certain restrictions:

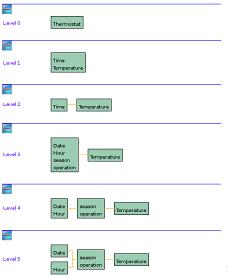
1. *In the diagram cycles are allowed.*
2. *Between two properties only a single dependency is allowed.*

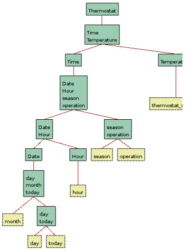
Conceptual Design – *ARD+*

ARD+ Features

- visual method
- two types of diagrams:
  - ARD diagram → attributes relationships diagram
  - TPH diagram → history of ARD transformations

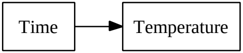
ARD diagram





TPH diagram

Simple diagaram



ARD+ diagaram transformations

- Diagram transformations are one of the core concepts in the ARD.
- They serve as a tool for diagram specification and development.
- For the transformation T such as $T : D_1 \rightarrow D_2$, where D_1 and D_2 are both diagrams, the diagram D_2 carries more knowledge, is more specific and less abstract than the D_1.
- A transformed diagram D_2 constitutes a more detailed *diagram level*.

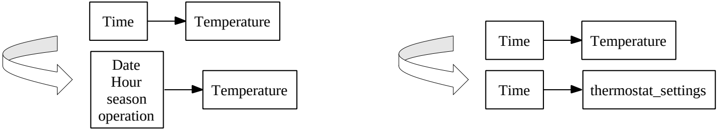
Finalization transformation

Definition

Finalization. Finalization TF is a function of the form

$$TF : P_1 \rightarrow P_2$$

transforming a simple property P_1 described by a conceptual attribute into a P_2, where the attribute describing P_1 is substituted by one or more conceptual or physical attributes describing P_2.



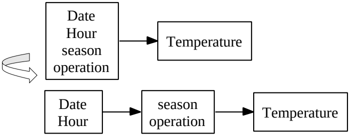
Split transformation

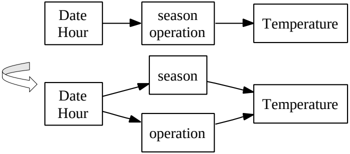
Definition

Split. A split is a function S of the form:

$$S : PS \rightarrow \{PS_1, PS_2, \ldots PS_n\}$$

where a complex property PS is replaced by n properties, each of them described by one or more attributes originally describing PS.





The TPH

The purposes of having the hierarchical model are:

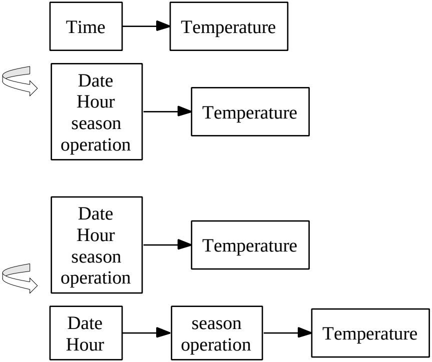
- gradual refinement of a designed system, and particularly
- identification of the origin of given properties,
- ability to get back to previous diagram levels for refactoring purposes,
- big picture perspective of the designed system.

Implementation:

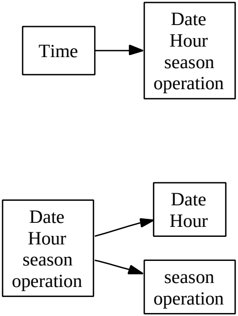
- storing the lowest available, most detailed diagram level, and
- information needed to recreate all of the higher levels: *Transformation Process History*.

The TPH Examples

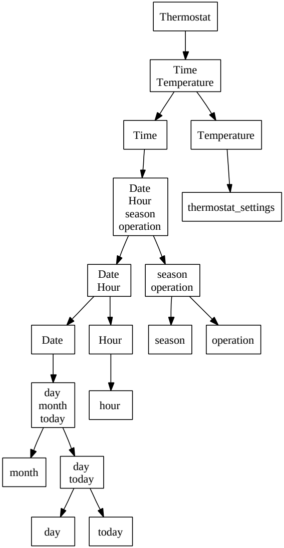
ARD:



TPH:



The TPH Examples



Hekate Rule Language

What is XTT...

- XTT2 → *eXtended Tabular Trees*
- the main stage of the HeKatE design process
- method of the rules design
- it bases on the ALSV(FD) logic

Features

- XTT2 bases on the classic concept of rule languages with certain extensions:
- provides structured rulebase
- provides visualization
- provides formal and on-line verification
- specifies inference control in the knowledge base

ALSV(FD)

ALSV(FD)

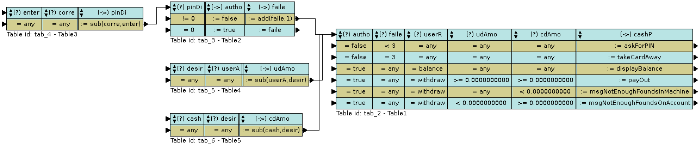
- ALSV(FD) – **A**ttributive **L**ogic with **S**et **V**alues over **F**inite **D**omains
- more expressive than First Order Logic
- it introduces a concept of generalized atrribute

ALSV(FD) in XTT

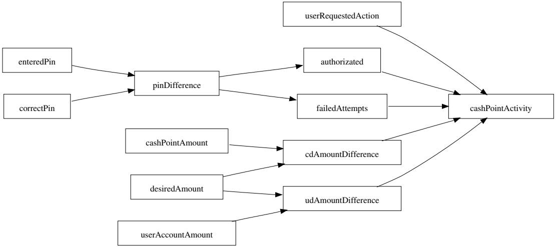
- an attribute A_i is a function of the form $A_i : O \to D_i$
- a rule format:
  $(A_1 \propto_1 V_1) \wedge (A2 \propto_2 V_2) \wedge \cdots \wedge (A_n \propto_n V_n) \to RHS$
- a simple table format:

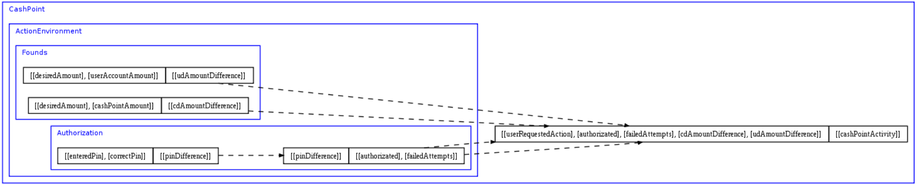
Rule	A_1	A_2	...	A_n	H
1	$\propto_{11}\ t_{11}$	$\propto_{12}\ t_{12}$	...	$\propto_{1n}\ t_{1n}$	h_1
2	$\propto_{21}\ t_{21}$	$\propto_{22}\ t_{22}$	...	$\propto_{2n}\ t_{2n}$	h_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
m	$\propto_{m1}\ t_{m1}$	$\propto_{m2}\ t_{m2}$	...	$\propto_{mn}\ t_{mn}$	h_m

XTT diagram example



ARD → XTT


userRequestedAction
enteredPin
authorizated
pinDifference
correctPin
failedAttempts
cashPointActivity
cashPointAmount
cdAmountDifference
desiredAmount
udAmountDifference
userAccountAmount


CashPoint
ActionEnvironment
Founds
[[desiredAmount], [userAccountAmount]]
[[udAmountDifference]]
[[desiredAmount], [cashPointAmount]]
[[cdAmountDifference]]
Authorization
[[enteredPin], [correctPin]]
[[pinDifference]]
[[pinDifference]]
[[authorizated], [failedAttempts]]
[[userRequestedAction], [authorizated], [failedAttempts], [cdAmountDifference], [udAmountDifference]]
[[cashPointActivity]]

Verification of XTT Components

Within the proposed approach verification of the following theoretical properties is performed:

- redundancy – subsumption of rules,
- indeterminism – overlapping rules,
- completeness – missing rules.

The components are checked if they are minimal and reduction possibilities are suggested.

Physical Design

Physical Design

- XTT2 $\xrightarrow{\text{automatic transition}}$ Physical Implementation (HMR)
- graphical representation → textual representation
- no semantic gap

HMR

- **HMR** – **H**ekate **M**eta **R**epresentation
- **HMR** – textual representation
- **HMR** – PROLOG based representation
- directly interpreted by HeaRT engine

HMR rule representation

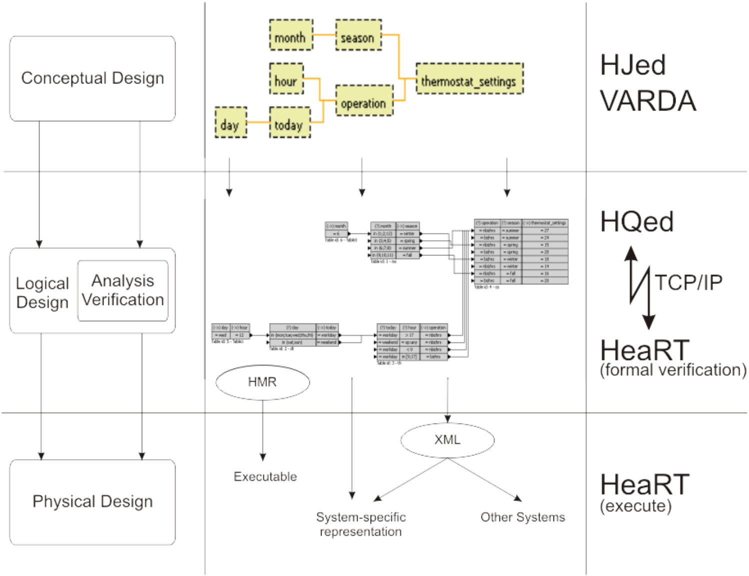
(?) diagn	(?) aller	(?) age	(->) medic
= AcuteSinusitils	= none	> 17	:= Amoxicillin
= AcuteSinusitils	= none	<= 17	:= Cefuroxime
= AcuteSinusitils	= penicillin	= any	:= Levofloxacin

Table id: tab_2 - Table1

```
xschm 'Table1': [diagnosis,allergic,age] ==> [medication].

xrule 'Table1'/1:
       [diagnosis eq 'AcuteSinusitils',
        allergic eq none,
        age gt 17]
     ==>
       [medication set 'Amoxicillin']
     :'Table2'.
xrule 'Table1'/2:
       [diagnosis eq 'AcuteSinusitils',
        allergic eq none,
        age lte 17]
     ==>
       [medication set 'Cefuroxime'].
xrule 'Table1'/3:
       [diagnosis eq 'AcuteSinusitils',
        allergic eq penicillin,
        age eq any]
     ==>
       [medication set 'Levofloxacin'].
```

HeKatE Rule Framework



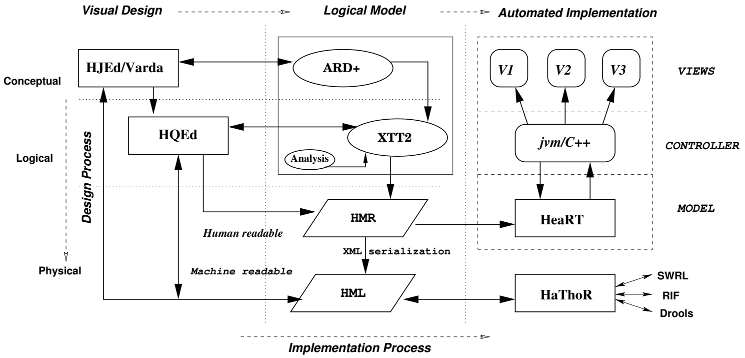
Knowledge Markup

- Knowledge in the HeKatE design process is described in HML (Hekate Markup Language), a machine readable XML-based format.
- HML consists of three logical parts:
  - ATTML – attribute specification,
  - ARDML – attribute and property relationship specification and
  - XTTML – rule specification.
- number of XSLT transformations provide translation to other markups, e.g. W3C RIF.

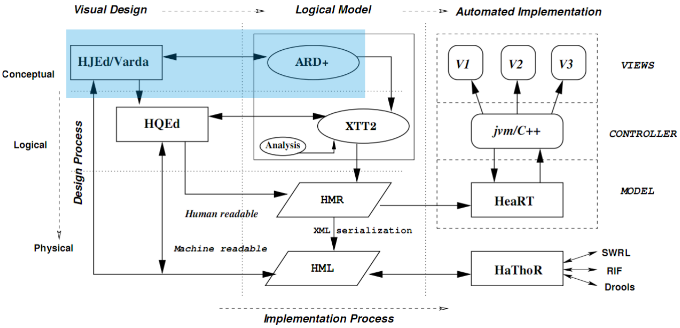
HaDEs

HaDEs

- HaDEs – **H**ekate **D**esign **E**nvironment
- a set of tools that supports design in HeKatE methodology



VARDA

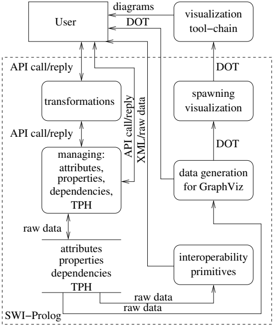


Visual Design
Logical Model
Automated Implementation
Conceptual
Logical
Physical
Design Process
HJEd/Varda
ARD+
HQEd
XTT2
Analysis
HMR
Human readable
XML serialization
Machine readable
HML
V1
V2
V3
VIEWS
jvm/C++
CONTROLLER
HeaRT
MODEL
HaThoR
SWRL
RIF
Drools
Implementation Process

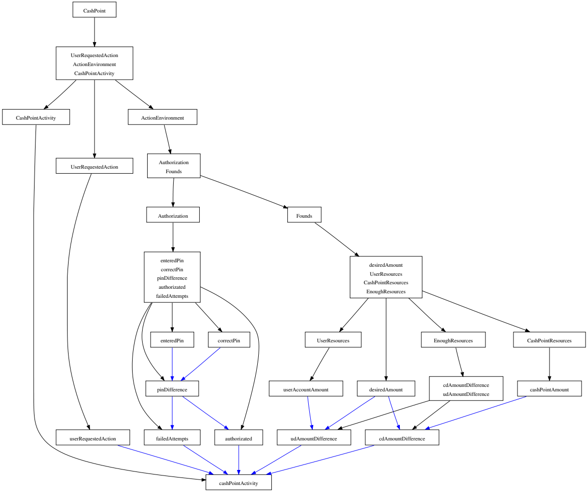
VARDA

VARDA

- VARDA – **V**isual **A**RD **R**apid **D**evelopment **A**lloy
- is a PROLOG based tool
- allows for ARD+ modeling by using PROLOG-based interface
- has command line user interface
- uses GraphViz for visualization



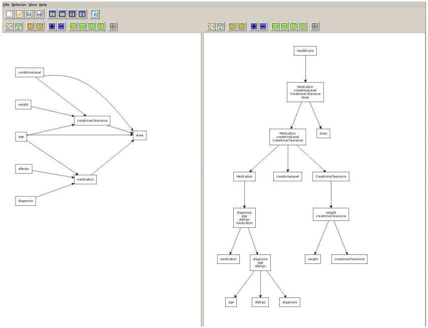
VARDA


CashPoint
UserRequestedAction
ActionEnvironment
CashPointActivity
CashPointActivity
ActionEnvironment
UserRequestedAction
Authorization
Founds
Authorization
Founds
enteredPin
correctPin
pinDifference
authorizated
failedAttempts
desiredAmount
UserResources
CashPointResources
EnoughResources
enteredPin
correctPin
UserResources
EnoughResources
CashPointResources
pinDifference
userAccountAmount
desiredAmount
cdAmountDifference
udAmountDifference
cashPointAmount
userRequestedAction
failedAttempts
authorizated
udAmountDifference
cdAmountDifference
cashPointActivity

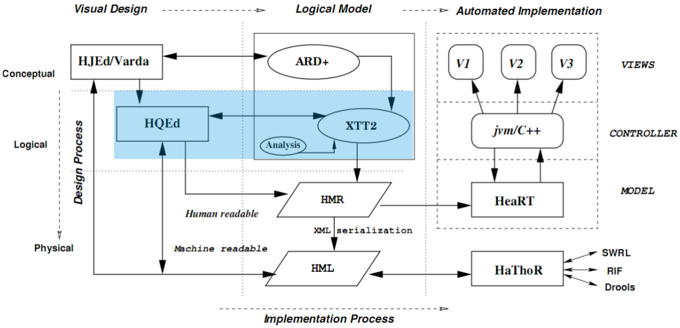
HJed

HJed

- HJed – **H**eKtE **J**ava **ED**itor
- allows for visual ARD+ modeling using GUI
- available under the GNU GPL from `https://ai.ia.agh.edu.pl/wiki/hekate:hjed`



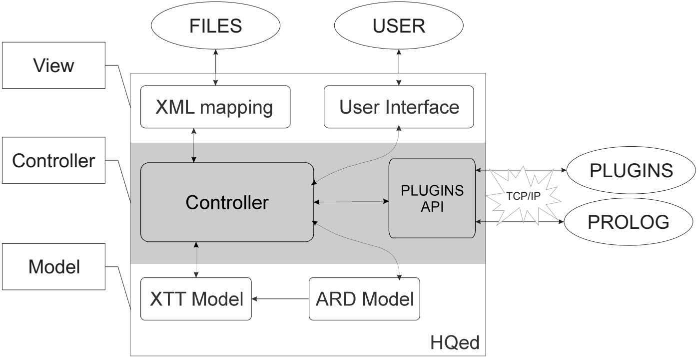
HQed


Visual Design
Logical Model
Automated Implementation
Conceptual
Logical
Physical
Design Process
HJEd/Varda
ARD+
HQEd
XTT2
Analysis
HMR
HML
Human readable
Machine readable
XML serialization
V1
V2
V3
VIEWS
jvm/C++
CONTROLLER
HeaRT
MODEL
HaThoR
SWRL
RIF
Drools
Implementation Process

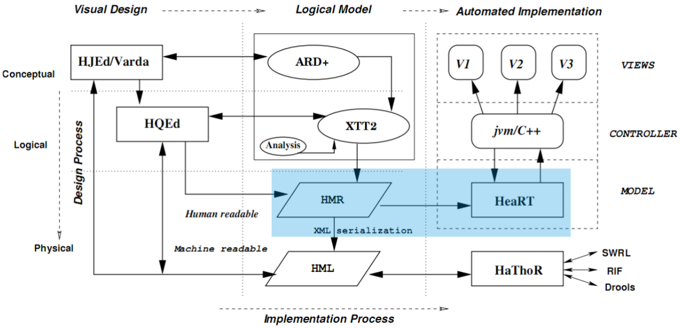
HQEd intro

- a complex visual XTT^2 editor
- using the rule prototypes generated with VARDA, it allows for the actual logical rule design
- the editor allows for gradual refinement of rules, with an online checking of attribute domains, as well as simple table properties
- a plugin framework allows for integrating Prolog-based analysis plugins to check formal properties of the XTT rule base
- HQEd is a crossplatform tool written in C++, that depends only on the Qt library
- it is available under the GNU GPL from `https://ai.ia.agh.edu.pl/wiki/hekate:hqed`
- the output from the editor is a complete rulebase encoded in Prolog
- it can be executed using a Prolog-based inference engine
- the rulebase can be integrated into a larger application as a logical core

HQed architecture



HeaRT


Visual Design
Logical Model
Automated Implementation
Conceptual
Logical
Physical
Design Process
HJEd/Varda
HQEd
ARD+
XTT2
Analysis
HMR
HML
Human readable
Machine readable
XML serialization
V1
V2
V3
jvm/C++
HeaRT
HaThoR
VIEWS
CONTROLLER
MODEL
SWRL
RIF
Drools
Implementation Process

HeaRT

HeaRT (HeKatE Run Time)

- dedicated inference engine for the XTT^2 rule bases
- cross-platform tool written in PROLOG

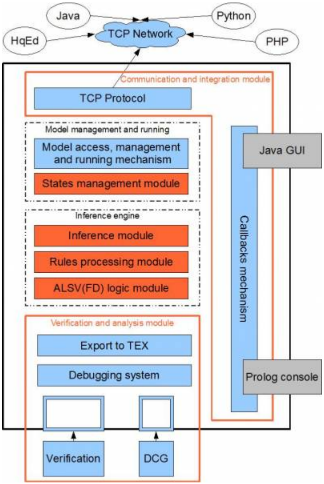
Functionality

- store and export models in HMR files
- verify HMR syntax and logic (HalVA)

Communication and integration

- direct interaction via Prolog console
- through TCP/IP protocol offers integration mechanism
- supports Java integration based on callbacks mechanism and Prolog JPL library

HeaRT architecture



HalVA

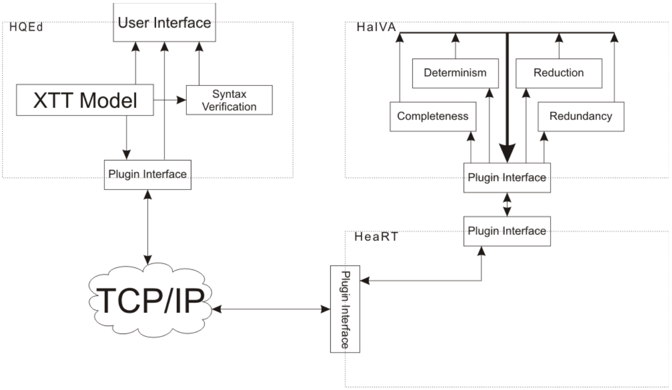
HalVA

- the verification framework for XTT^2 model
- consists of several PROLOG-based plugins

Features

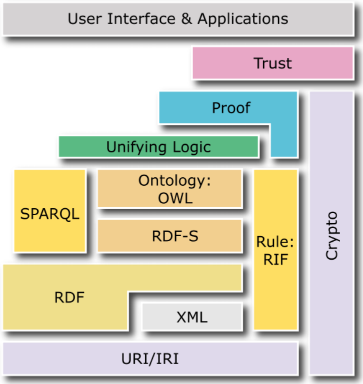
- it has a simple debugging mechanism
- allows for export entire model to LaTeX
- it supports syntactic analysis of HMR using a DCG grammar
- **it provides logical verification of models:** completeness, determinism and redundancy
- it can be run from the interpreter or indirectly from HQEd using the communication protocol

Verification cycle



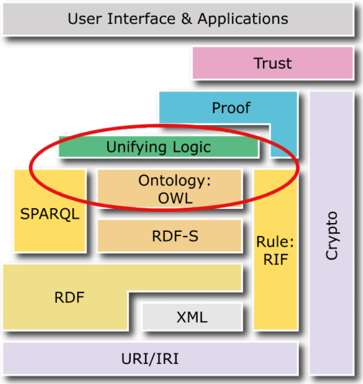
SemWeb Architecture (a.k.a. "layer cake")

- application interface
- safety
- rules, logic (SWRL,RIF, ...)
- ontologies (RDFS, OWL)
- metadata (RDF)
- serialization (XML)
- resource identification (URI)

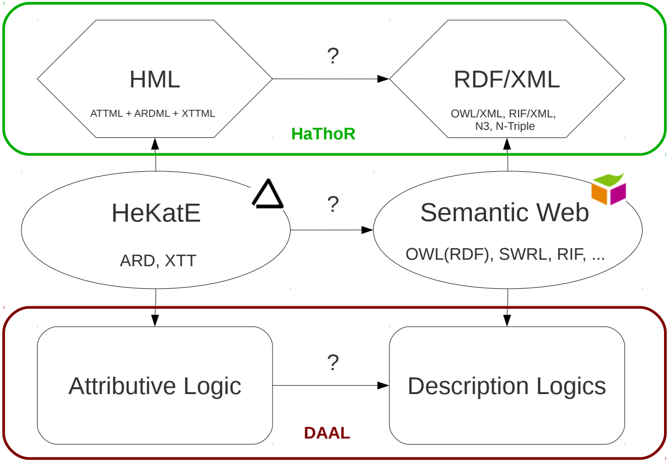


SemWeb Architecture (a.k.a. "layer cake")

- application interface
- safety
- rules, logic (SWRL,RIF, ...)
- ontologies (RDFS, OWL)
- metadata (RDF)
- serialization (XML)
- resource identification (URI)



Big Picture


HML
ATTML + ARDML + XTTML
?
RDF/XML
OWL/XML, RIF/XML, N3, N-Triple
HaThoR
HeKatE
ARD, XTT
?
Semantic Web
OWL(RDF), SWRL, RIF, ...
Attributive Logic
?
Description Logics
DAAL

Motivation and challenges for integration

Motivation

- Expressive rule language for the Semantic Web applications
- Visual rule design support
- Methodology for a rule base design, not only single rules
- Formal verification and validation

Integration challenges

- ALSV(FD) – dynamic changes of a system, DL – terminological structural knowledge
- CWA in ALSV(FD), OWA in DL
- UNA in ALSV(FD)
- nonmonotonicity in ALSV(FD), no *assert*/*retract* functions in DL

DAAL Approach

Observation

- KR HeKatE based on Attributive Logic
- Ontologies based on Description Logics

Let's understand the foundations and how they are related

Research track

- Analysis of knowledge representation in
  - Attributive Logic
  - Description Logics
- Mapping proposal

Description And Attributive Logics (DAAL)

1. Conceptual modelling: attributes in AL, concepts in DL

Attr. Name	Attr. Domain	Concept Constructors
A_i	D_i $D = \{a_1, a_2, \ldots, a_n\}$	$A_i \equiv D_i$ $D \equiv \{a_1, a_2, \ldots, a_n\}$

2. Rule formulas – DL axioms

AL Formula	DL Axiom	AL Formula	DL Axiom
$A_i = d$ $A_i \in V_i$	$A_i \equiv \{d\}$ $A_i \equiv V_i$	$A_i \neq d$ $A_i \notin V_i$	$A_i \equiv \neg\{d\}$ $A_i \equiv \neg V_i$

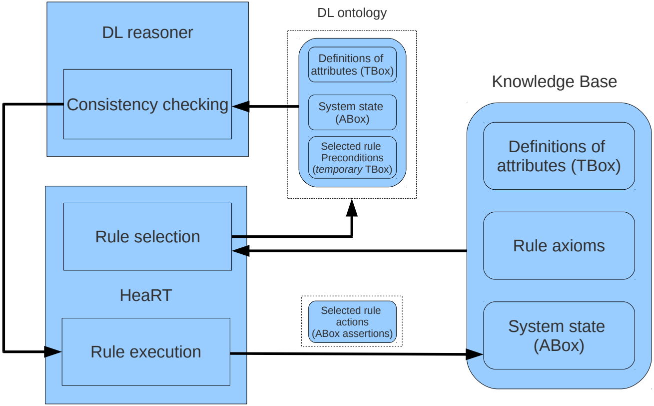
3. State representation – World description

Attr. Type	AL Formula	DL Assertion
simple	$A_i := d_i$	$A(d)$
generalized	$A_i := V_i$, $V_i = \{v_{i_1}, \ldots, v_{i_n}\}$	$A(v_{i_1}).A_i(v_{i_2}).\ldots.A_i(v_{i_n})$

4. Reasoning – consistency checking of temporary ontologies (rule preconditions + system state)

Inference

Hybrid system: HeaRT engine + DL reasoner



HaThoR Approach

Observation

- HeKatE Markup Language
- RDF/XML syntax for ontologies

How to translate one format to the other?

Research track

- Analysis of serialization formats
- XSLT translators from HML to RDF/XML
- HaThoR 2 Online:
  `http://home.agh.edu.pl/wtf/onto/hathor2/www`

HeKatE Translator (HaThoR)

1. Attributes ↔ instances
2. Class constraints: datatype properties for types and object properties linking attributes and types

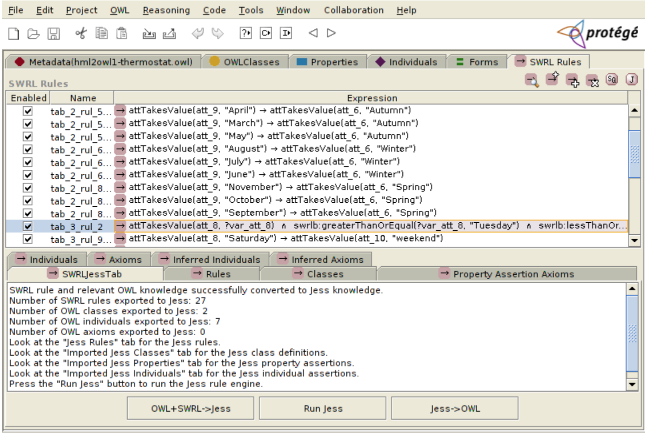
Attributive Logic		Description Logic
Attr. Name	Attr. Domain	DL Axioms
A_i	D_i $D_i = \{a_1, \ldots, a_n\}$	$Type(T_i), Attribute(A_i)$ $attHasType(A_i, T_i)$, $domainOfT_i(T_i, D_i)$

3. State: role assertions

Attributive Logic		Description Logic
Attribute Type	Formula	Assertion in ABox
simple	$A_i := d_i$	$attTakesValue(A_i, d_i)$
generalized	$A_i := V_i$	$attTakesValue(A_i, v_1), \ldots$ $\ldots, attTakesValue(A_i, v_n)$

Inference

SWRLTab in Protegé



Design Process Ontology

Motivation

Overcome limitations of ARD:

- allow to specify different classes of functional dependencies,
- provide more expressive means for the history description,
- allow to build a single coherent model combining both functional dependencies and history information.

Idea

Use an ontology to capture the functional dependencies present in the main ARD diagram and history information captured in the TPH.

Design Process Ontology

DPO is a proposal of a *task ontology*. Its aim is to capture the system characteristics together with dependencies among them, as well as represent the gradual refinement of the design process.

DPO Concepts

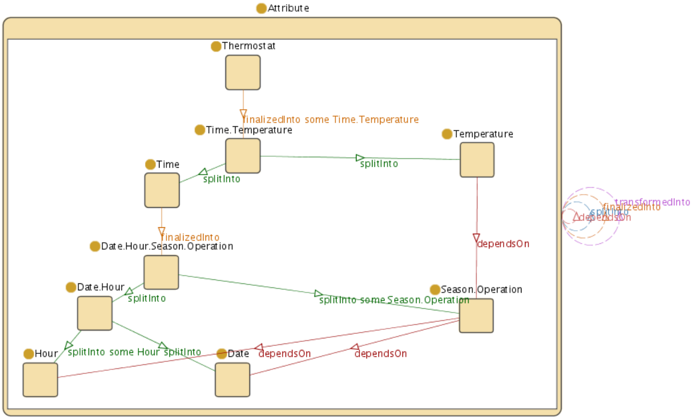
Main classes and properties

- a geneneral class *Attribute*
- 4 properties,
  - `dependsOn` – *functional dependencies*
  - `transformedInto`, `splitInto` and `finalizedInto` – the design process transformations.

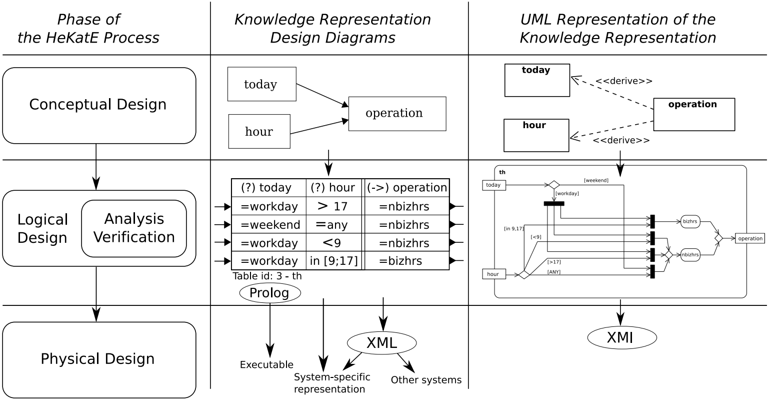
DPO for concrete tasks

DPO may be specialized by concrete ontologies for specific design tasks. In this case system characteristics subclass the `Attribute` class. The properties may be specialized accordingly.

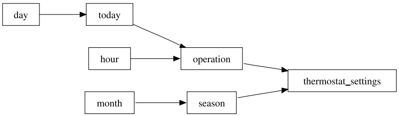
Simple DPO in OWL designed in Protegé

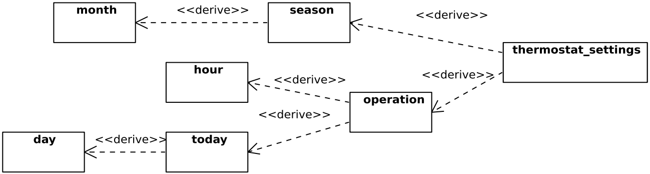


HeKatE process and UML representation



An example of ARD diagram and its UML representation

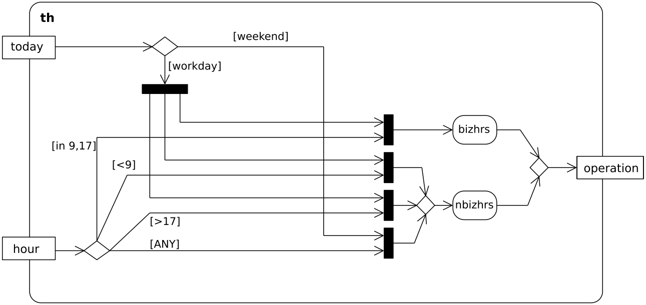




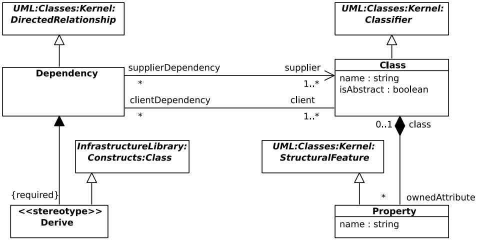
An example of XTT table and its UML representation

(?) today	(?) hour	(->) operation
=workday	> 17	=nbizhrs
=weekend	=any	=nbizhrs
=workday	<9	=nbizhrs
=workday	in [9;17]	=bizhrs

Table id: 3 - th



Metamodel for UML representation of ARD



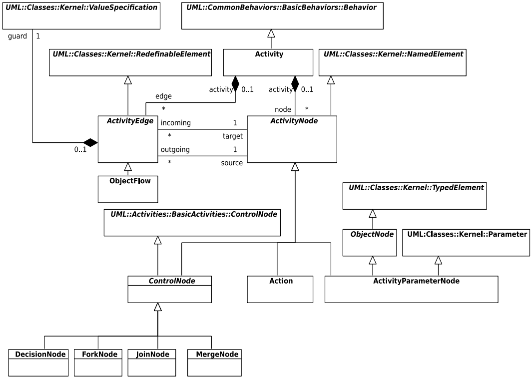
Example of the OCL constraints for Class:

```
context Class
 inv:

 Class.allInstances()->one( c |
  c.isAbstract = false
  and
  c.association->size() = 0
 )

 self.name = self.name.toLower() implies
  self.supplierDependency->size() = 0
  and self.association->size() = 0
```

Metamodel for UML representation of XTT



Example of the OCL constraints for ActivityParameterNode:

```
context ActivityParameterNode
 inv:

 self.incoming->forAll( edge |
  edge.source.oclIsTypeOf(MergeNode)
   xor
   edge.source.oclIsKindOf(Action)
 )

 self.outgoing->forAll( edge |
   edge.target.
      oclIsTypeOf(DecisionNode)

 )
```

Summary

The proposed representation:

- provides two abstraction levels (tables and system)
- does not introduce new artifacts or extend the UML language
- shows both structure of dependecies
  and behaviour of the system (rule processing)

The proposed solution:

- provides XMI serialization
- provides translation between UML representation (XMI)
  and HeKatE project representation (HML)

The End