HeKaTe Methodology - Hybrid Engineering of Intelligent Systems

Grzegorz J. Nalepa and Antoni Ligęza

Institute of Automatics
AGH University of Science and Technology, POLAND

Seminarium Obliczenia Inteligentne w Nowoczesnej Automatyce
Zielona Góra, Poland, October 24, 2008

Supported from the HEKATE Research Project

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

Outline

Knowledge Engineering and Data Engineering

Rules

- rules are some most successful knowledge representation formalism
- number of engineering and business applications (hidden or explicit)
- high-level declarative knowledge representation
- *logical independence of applications* — rules are 'data'
- potential for encoding functions, relations, causality,...
- logical model (background)

Relational Databases – success factors as inspiration

- three-phase, top-down design process
- *physical and logical independence* (E.F. Codd)
- tabular representation with attributes (records/tables/joins/views)
- efficient data selection, relational algebra
- integrity constraints, internal consistency
- algebraic model (background)

Knowledge Engineering and Data Engineering

Rules

- rules are some most successful knowledge representation formalism
- number of engineering and business applications (hidden or explicit)
- high-level declarative knowledge representation
- *logical independence of applications* — rules are 'data'
- potential for encoding functions, relations, causality,...
- logical model (background)

Relational Databases – success factors as inspiration

- three-phase, top-down design process
- *physical and logical independence* (E.F. Codd)
- tabular representation with attributes (records/tables/joins/views)
- efficient data selection, relational algebra
- integrity constraints, internal consistency
- algebraic model (background)

Concepts and Methods

Attributive Logic

- formally describe a system in terms of its attributes
- non-atomic attribute values, complex inference modes
- foundations for an extended rule language

eXtended Tabular Trees

- structured rule representation (decision tables and trees)
- visual analysis and design support
- formalized description and verification

Attribute Relationship Diagrams

- conceptual design for XTT
- requirements engineering with attributes
- visual graph-like representation

Concepts and Methods

Attributive Logic

- formally describe a system in terms of its attributes
- non-atomic attribute values, complex inference modes
- foundations for an extended rule language

eXtended Tabular Trees

- structured rule representation (decision tables and trees)
- visual analysis and design support
- formalized description and verification

Attribute Relationship Diagrams

- conceptual design for XTT
- requirements engineering with attributes
- visual graph-like representation

Concepts and Methods

Attributive Logic

- formally describe a system in terms of its attributes
- non-atomic attribute values, complex inference modes
- foundations for an extended rule language

eXtended Tabular Trees

- structured rule representation (decision tables and trees)
- visual analysis and design support
- formalized description and verification

Attribute Relationship Diagrams

- conceptual design for XTT
- requirements engineering with attributes
- visual graph-like representation

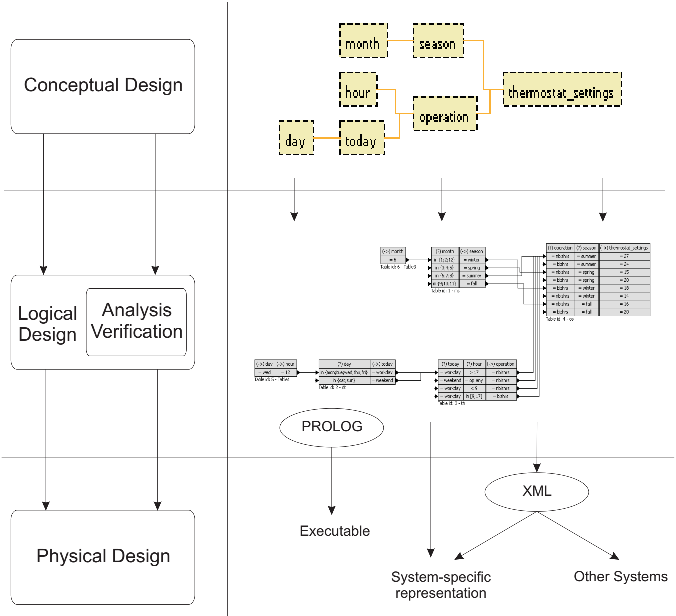
Research Objectives

- provide an integrated, top-down design and analysis methodology for a wide class of rule-based systems
- build a toolchain supporting it
- analyze and design representative usecases
- propose an integration framework with classic Software Engineering methods (UML/OOP)
- implement prototypes for business rules, Java applications, as well as embedded control systems
- formal analysis of declarative rule-based system properties (on-line)
- improve system quality during the design process

Concepts

1. formal logical system description $ALSV(FD)$
2. structured rule-based system core
3. requirements engineering $\rightarrow$ rule prototyping $ARD+$
4. system design $\rightarrow$ rules in XTT^2
5. automated implementation $\rightarrow$ prototype generation
6. formal verification of the model (partial)

Design Process



Logical Rule Formulation

Regular class hours are 8:00 – 18:00. If all teaching hours are located within regular class hours then the salary is regular. If teaching hours go beyond the regular class hours then salary is special.

The problem is to formalize these two rules with attributive logic. Let *RCH* stays for regular class hours, and *TH* for teaching hours. We can define a fact like:

$$RCH = \{8, 9, 10, 11, 12, 13, 14, 15, 16, 17\},$$

$$TH = \{10, 11, 12, 16, 19, 20\}$$

to specify a case of teaching hours.

To express the rules we need an extended attributive logic employing set values of attributes and some powerful relational symbols. For example:

$$R1: \; TH \subseteq RCH \longrightarrow Salary =' regular'$$

$$R2: \; TH \sim NRCH \longrightarrow Salary =' special'$$

where

$$NRCH = \{0, 1, 2, 3, 4, 5, 6, 7, 18, 19, 20, 21, 22, 23\}$$

is a specifications of non-regular class hours, and $\sim$ a non-empty intersection.

Logical Rule Formulation

- Attributive logics constitute a simple but widely-used tool for knowledge specification and inference.
- In fact in a large variety of applications in various areas of Artificial Intelligence and Knowledge Engineering attributive languages constitute the core knowledge representation formalism (RBS, RDBMS)
- Although Propositional Logic and Predicate Logic have well-elaborated syntax and semantics, presented in details in numerous books, the discussion of attribute-based logic is omitted in such sources.
- In [**?**] the discussion of attributive logic is much more thorough. The added value consist in allowing that attributes can take *set values* and providing formal syntax of the *Set Attributive Logic* (SAL) with respect to its syntax, semantics and selected inference rules.
- The very basic idea is that attributes can take *atomic* or *set* values.

Attribute

- It is assumed that an attribute A_i is a function (or partial function) of the form $A_i\colon O \rightarrow D_i$.
- A generalized attribute A_i is a function (or partial function) of the form $A_i\colon O \rightarrow 2^{D_i}$, where 2^{D_i} is the family of all the subsets of D_i.
- The atomic formulae of SAL can have the following three forms: $A_i = d$, $A_i = t$ or $A_i \in t$, where $d \in D$ is an atomic value from the domain D of the attribute and $t = \{d_1, d_2, \ldots, t_k\}$, $t \subseteq D$ is a set of such values.
- The semantics of $A_i = d$ is straightforward, the attribute takes a single value.
- The semantics of $A_i = t$ is that the attribute takes *all* the values of t (the so-called *internal conjunction*) while the semantics of $A_i \in t$ is that it takes *some* of the values of t (the so-called *internal disjunction*).

In this paper an improved and extended version of SAL is presented in brief. The formalism is oriented toward Finite Domains (FD) and its expressive power is increased through introduction of new relational symbols.

ALSV(FD)

- Let us consider:
  - **A** – a finite set of attribute names,
  - **D** – a set of possible attribute values (the *domains*).
- Let $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$ be all the attributes such that their values define the state of the system under consideration.
- It is assumed that the overall domain **D** is divided into *n* sets (disjoint or not), $\mathbf{D} = D_1 \cup D_2 \cup \ldots \cup D_n$, where D_i is the domain related to attribute A_i, $i = 1, 2, \ldots, n$. Any domain D_i is assumed to be a finite (discrete) set.
- As we consider dynamic systems, the values of attributes can change over time (or state of the system). We consider both *simple* attributes of the form $A_i\colon T \rightarrow D_i$ (i.e. taking a single value at any instant of time) and *generalized* ones of the form $A_i\colon T \rightarrow 2^{D_i}$ (taking a set of values at a time).

Syntax of ALSV

Let A_i be an attribute of $\mathbf{A}$ and D_i the sub-domain related to it. Let V_i denote an arbitrary subset of D_i and let $d \in D_i$ be a single element of the domain.

Definition

The legal atomic formulae of ALSV for simple attributes are:
$A_i = d$, $A_i \neq d$, $A_i \in V_i$, $A_i \notin V_i$.

Definition

The legal atomic formulae of ALSV for generalized attributes are:
$A_i = V_i$, $A_i \neq V_i$, $A_i \subseteq V_i$, $A_i \supseteq V_i$, $A \sim V$, $A_i \not\sim V_i$.

- In case V_i is an empty set (the attribute takes in fact no value) we shall write $A_i = \{\}$.
- In case the value of A_i is unspecified we shall write $A_i = \texttt{NULL}$ (a database convention).
- If we do not care about the current value of the attribute we shall write $A = \_$ (a PROLOG convention).

Semantics of ALSV

- In case of the first four possibilities we consider A_i to be a simple attribute taking exactly one value.
- In case of next two the value is precisely defined, while in case of the third (subset) any of the values $d \in V_i$ satisfies the formula.
- In other words, $A_i \in V_i$ is equivalent to $(A_i = d_1) \otimes (A_i = d_2) \otimes \ldots \otimes (A_i = d_k)$, where $V_i = \{d_1, d_2, \ldots, d_k\}$ and $\otimes$ stays for exclusive-or.
- The semantics of next group is that A_i is a generalized attribute taking a set of values equal to V_i (and nothing more), different from V_i (at at least one element), being a subset of V_i, being a superset of V_i, having a non-empty intersection with V_i or disjoint to V_i, respectively.
- More complex formulae can be constructed with *conjunction* ($\wedge$) and *disjunction* ($\vee$); both the symbols have classical meaning and interpretation.
- There is no explicit use of negation.

Inference Rules for ALSV(FD)

Let V and W be two sets of values such that $V \subseteq W$. We have the following straightforward inference rules for atomic formulae:

$$\frac{A \supseteq W}{A \supseteq V} \tag{1}$$

i.e. if an attribute takes all the values of a certain set it must take all the values of any subset of it (downward consistency). Similarly

$$\frac{A \subseteq V}{A \subseteq W} \tag{2}$$

i.e. if the values of an attribute takes values located within a certain set they must also belong to any superset of it (upward consistency).

Inference rules for atomic formulae for simple attributes

$\models$	$A = d_j$	$A \neq d_j$	$A \in V_j$	$A \notin V_j$
$A = d_i$	$d_i = d_j$	$d_i \neq d_j$	$d_i \in V_j$	$d_i \notin V_j$
$A \neq d_i$	_	$d_i = d_j$	$V_j = D \setminus \{d_i\}$	$V_j = \{d_i\}$
$A \in V_i$	$V_i = \{d_j\}$	$d_j \notin V_i$	$V_i \subseteq V_j$	$V_i \cap V_j = \emptyset$
$A \notin V_i$	$D \setminus V_i = \{d_j\}$	$V_i = \{d_j\}$	$V_j = D \setminus V_i$	$V_j \subseteq V_i$

Inference rules for atomic formulae for generalized attributes

$\models$	$A = W$	$A \neq W$	$A \subseteq W$	$A \supseteq W$	$A \sim W$	$A \not\sim W$
$A = V$	$V = W$	$V \neq W$	$V \subseteq W$	$V \supseteq W$	$V \cap W \neq \emptyset$	$V \cap W = \emptyset$
$A \neq V$	_	$V = W$	$W = D$	_	$W = D$	_
$A \subseteq V$	_	$V \subset W$	$V \subseteq W$	_	$W = D$	$V \cap W = \emptyset$
$A \supseteq V$	_	$W \subset V$	$W = D$	$V \supseteq W$	$V \cap W \neq \emptyset$	_
$A \sim V$	_	$V \cap W = \emptyset$	$W = D$	_	$V = W$	_
$A \not\sim V$	_	$V \cap W \neq \emptyset$	$W = D$	_	$W = D$	$V = W$

Rules in ALSV(FD)

Consider a set of n attributes $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$.
Any rule is assumed to be of the form:

$$(A_1 \propto_1 V_1) \wedge (A_2 \propto_2 V_2) \wedge \ldots (A_n \propto_n V_n) \longrightarrow RHS$$

where $\propto_i$ is one of the admissible relational symbols in ALSV(FD), and RHS is the right-hand side of the rule covering conclusion and possibly the retract and assert definitions if necessary.

XTT Table

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

Rule Firing

- The current values of all the attributes are specified with the contents of the knowledge-base (including current sensor readings, measurements, etc.).
- From logical point of view it is a formula of the form:

$$(A_1 = S_1) \wedge (A_2 = S_2) \wedge \ldots \wedge (A_n = S_n), \quad (3)$$

where $S_i = d_i$ ($d_i \in D_i$) for simple attributes and $S_i = V_i$, ($V_i \subseteq D_i$) for complex.

- The rules having all the preconditions satisfied can be fired.
- In general, rules can be fired in parallel (at least in theory) or sequentially.
- For the following analysis we assume the classical, sequential model, i.e. the rules are examined in turn in the top-down order and fired if the preconditions are satisfied.

ARD+ Goal

Goal

- Support the designer at a very general design level, the conceptualization.
- A *requirements specification* method.
- *Input*: a general system description in the natural language.
- *Output*: a model capturing knowledge about relationships between attributes describing system properties.
- The model is subsequently used in the next design stage, where the actual logical design with *rules* is carried out.

ARD+ Concepts

Main concepts

attributive logic use of *attributes* for denoting properties in a system

functional dependency a general relation between two or more attributes

graph notation simple expressive knowledge specification

visualization is the key concept in the practical design support

gradual refinement the design is being specified in number of steps, each step being more detailed than the previous one

structural transformations *formalized*, well defined syntax and semantics

hierarchical model captures all of the subsequent design steps, with no semantic gaps

knowledge-based approach *declarative* and transparent model specification.

ARD syntax I

Definition

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

Conceptual attribute names are capitalized, e.g.: `WaterLevel`.

Definition

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

Names of physical attributes are not capitalized, e.g. `theWaterLevelInTank1`.

ARD syntax II

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*.*

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.*

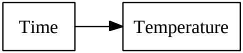
Constraint

ARD syntax III

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.*

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*.

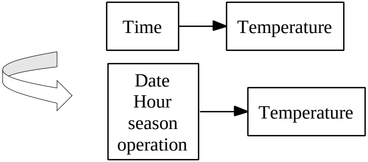
Finalization transformation I

Definition

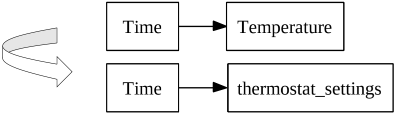
Finalization. Finalization TF is a function of the form

$$TF : P_1 \rightarrow P_2$$

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.



Finalization transformation II



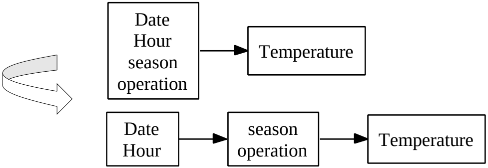
Split transformation I

Definition

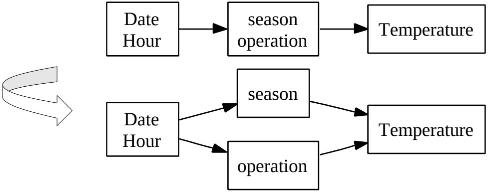
Split. A split is a function S of the form:

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

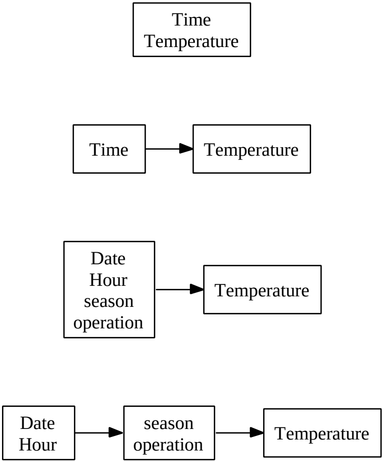
where a complex property PS is replaced by n properties, each of them described by one or more attributes originally describing PS.



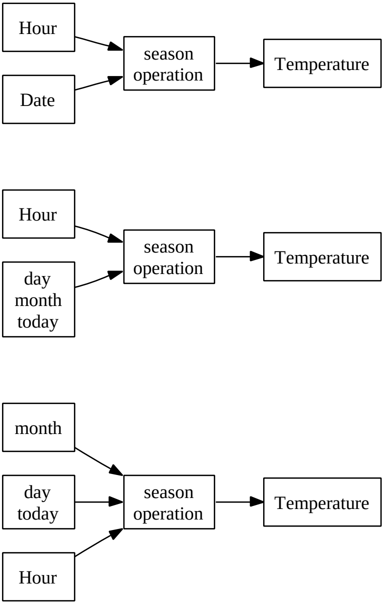
Split transformation II



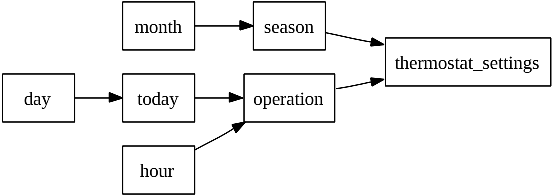
Example I


Time
Temperature
Time
Temperature
Date
Hour
season
operation
Temperature
Date
Hour
season
operation
Temperature

Example II


Hour
Date
season operation
Temperature
Hour
day month today
season operation
Temperature
month
day today
Hour
season operation
Temperature

Example III


month
season
thermostat_settings
day
today
operation
hour

The TPH

The purposes of having the hierarchical model are:

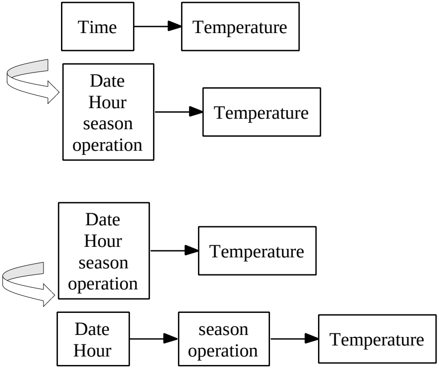
- gradual refinement of a designed system, and particularly
- identification of the origin of given properties,
- ability to get back to previous diagram levels for refactoring purposes,
- big picture perspective of the designed system.

Implementation:

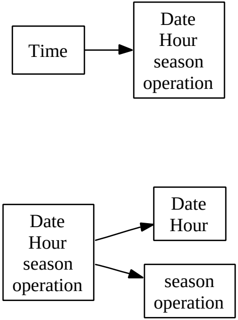
- storing the lowest available, most detailed diagram level, and
- information needed to recreate all of the higher levels: *Transformation Process History*.

The TPH Examples I

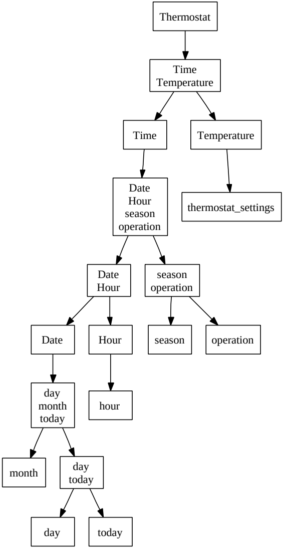
ARD:



TPH:



The TPH Examples II



Hekate Rule Language

- An extended rule language is proposed. It is based on the XTT language.
- The XTT rule language is based on the classic concepts of rule languages with certain important extensions and features.
- In XTT the rule base is explicitly structured. The rules with same sets of attributes are grouped within decision tables.
- On the rule level explicit inference control is allowed. In this way, a set of tables is interconnected using links, corresponding to inference control.
- This makes up a decision-tree like structure, with tables in the tree nodes. In a general case, the XTT is a directed graph, with cycles optionally allowed.
- In XTT these expressions are in the the *attributive logic*.

XTT Example

(?) month	(->) season
in 1;2;12	= winter
in 3;4;5	= spring
in 6;7;8	= summer
in 9;10;11	= fall

Table id: 1 - ms

(?) day	(->) today
in mon;tue;wed;thu;fri	= workday
in sat;sun	= weekend

Table id: 2 - dt

(?) today	(?) hour	(->) operation
= workday	> 17	= nbizhrs
= weekend	= ANY	= nbizhrs
= workday	< 9	= nbizhrs
= workday	in 9,17	= bizhrs

Table id: 3 - th

(?) operation	(?) season	(->) hermostat_setting
= nbizhrs	= summer	= 27
= bizhrs	= summer	= 24
= nbizhrs	= spring	= 15
= bizhrs	= spring	= 20
= nbizhrs	= winter	= 14
= bizhrs	= winter	= 18
= nbizhrs	= fall	= 16
= bizhrs	= fall	= 20

Table id: 4 - os

AGH

XTT^2 **Rule Design**

1. generate rule prototypes (XTT table schemes) automatically [**?**],
2. build table rows to specify actual *rules*
3. specify inference in the knowledge base

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.

Analysis of subsumption

Consider two rules, r and r' given below (simplified XAT scheme):

rule	A_1	A_2	...	A_j	...	A_n	H
r	t_1	t_2	...	t_j	...	t_n	h
r'	t'_1	t'_2	...	t'_j	...	t'_n	h'

The condition for subsumption in case of tabular rule format takes the algebraic form $t'_j \subseteq t_j$, for $j = 1, 2, \ldots, n$ and $h' \subseteq h$.
If it holds, then rule r' can be eliminated leaving the more general rule:

rule	A_1	A_2	...	A_j	...	A_n	H
r	t_1	t_2	...	t_j	...	t_n	h

Subsumption example

In the following tabular system the first rule subsumes the second one:

rule	A_1	A_2	A_3	A_4	H
r	7	$[2, 9]$	$[3, 5]$	$\{r, g, b\}$	$\{a, b, c\}$
r'	7	$[3, 5]$	4	$\{b, r\}$	$\{a, c\}$

Hence, rule r' can be eliminated.

Subsumption, defined as above, covers also detection and elimination of identical rules and equivalent rules; moreover, it is performed with purely algebraic means. In the example case of the Thermostat specification there are no subsumed rules.

Analysis of indeterminism

- In order to have two rules applicable in the same context, their preconditions must have non-empty intersection.
- For any attribute A_j there is an atom of the form $A_j = t_j$ in r and $A_j = t'_j$ in r', $i = 1, 2, \dots, n$.
- Now, one has to find the intersection of t_j and t'_j — if at least one of them is empty (e.g. two different values; more generally $t_{1,j} \cap t_{2,j} = \emptyset$) then the preconditions are disjoint and thus the rules are deterministic.
- The check is to be performed for any pair of rules.

In the example case of the Thermostat specification there are no indeterministic rules.

Conflict and inconsistency

- Problems of *conflicting* and *inconsistent* rules are specific cases of lack of indeterminism.
  - *conflict* when two (or more) rules are applicable to the same input situation but the results are conflicting (under the assumed interpretation)
  - *inconsistency* when purely logical inconsistency occurs.
- Detection of indeterminism is a necessary condition for eliminating conflict and inconsistency.
- Moreover, in tabular systems with no explicit negation purely logical inconsistency cannot occur; it always follows from the intended interpretation and thus it falls into the class of conflicts.

Analysis of reduction

Several rules having identical conclusion part can be glued to a single, equivalent rule according to the following scheme:

rule	A_1	A_2	$\ldots$	A_j	$\ldots$	A_n	H
r^1	t_1	t_2	$\ldots$	t_{1j}	$\ldots$	t_n	h
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$
r^k	t_1	t_2	$\ldots$	t_{kj}	$\ldots$	t_n	h
r	t_1	t_2	$\ldots$	T	$\ldots$	t_n	h

provided that $t_{1j} \cup t_{2j} \cup \ldots \cup t_{kj} = T$. If T is equal to the complete domain, then $T = \_$.

(The rules $r^1, r^2, \ldots, r^k$ are just some selected rows of the original table containing all of the rules.)

Reduction in the Thermostat

Info	Prec		Retract	Assert	Decision	Ctrl	
I	*aTD*	*aTM*	*aOP*	*aOP*	*H*	*N*	*E*
3	*wd*	[9:00, 17:00]	–	*dbh*		3.7	2.4
4	*wd*	[00:00, 09:00]	–	*ndbh*		3.7	2.5
5	*wd*	[17:00, 24:00]	–	*ndbh*		3.7	2.6
6	*wk*	–	–	*ndbh*		3.7	2.3

rules 4 and 5 can be glued, provided that the time specification can be expressed with non-convex intervals: [00:00-09:00]∪[17:00-24:00]

Reduction in the Thermostat

Info	Prec		Retract	Assert	Decision	Ctrl	
I	aSE	aOP			$aTHS$	N	E
11	*spr*	*dbh*			20	1.1	4.12
12	*spr*	*ndbh*			15	1.1	4.13
13	*sum*	*dbh*			24	1.1	4.14
14	*sum*	*ndbh*			17	1.1	4.15
15	*aut*	*dbh*			20	1.1	4.16
16	*aut*	*ndbh*			16	1.1	4.17
17	*win*	*dbh*			18	1.1	4.18
18	*win*	*ndbh*			14	1.1	1.1

rules 11 and 15 can be glued to a single rule, in this case the preconditions would read $aSE \in \{spr, sum\} \wedge aOP = dbh$

Analysis of completeness

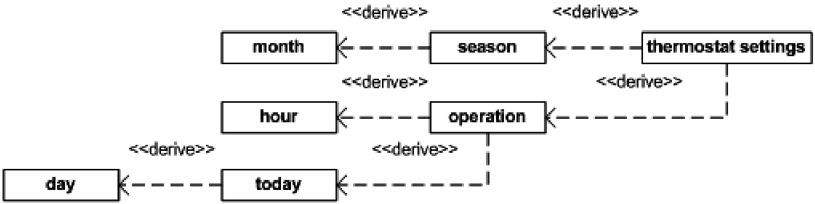
The system is complete in the sense that there are no admissible (correct) inputs which are uncovered.

1. First some maximal reduction is performed on the precondition part of a selected table.
2. In the ideal case an empty table (full completeness) is confirmed.
3. In other case we check which input specifications are not covered.
4. Thanks to allowing for non-atomic values of attributes it is *not necessary* to perform the so-called *exhaustive enumeration check*
5. The attribute domains can be divided into subsets (granularized) corresponding to the values occurring in the table
6. Hence the check is performed in a more abstract level and with increased efficiency.
7. Uncovered input specifications define the potentially missing rule preconditions.

Objectives

- provide a bridge for the classic SE methods and tools
- build a UML representation for ARD and XTT [**?**]
- formalize model transformation (with use of MOF metamodel and XMI, in the works)
- integrate the logical rule-based core (Model) with interfaces (View) with a hybrid Controller (the MVC pattern)

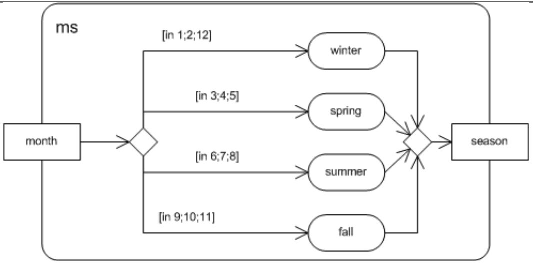
Representing ARD with component diagrams



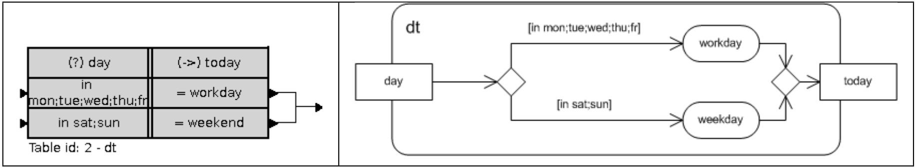
Activity diagram corresponding to XTT MS table

(?) month	(->) season
in 1;2;12	= winter
in 3;4;5	= spring
in 6;7;8	= summer
in 9;10;11	= fall

Table id: 1 - ms



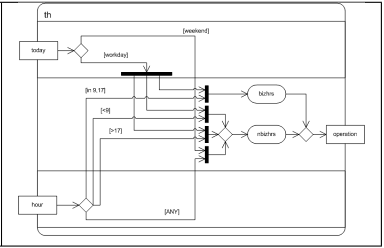
Activity diagram corresponding to XTT DT table



Activity diagram corresponding to XTT TH table

(?) today	(?) hour	(->) operation
= workday	> 17	= nbizhrs
= weekend	= ANY	= nbizhrs
= workday	< 9	= nbizhrs
= workday	in 9,17	= bizhrs

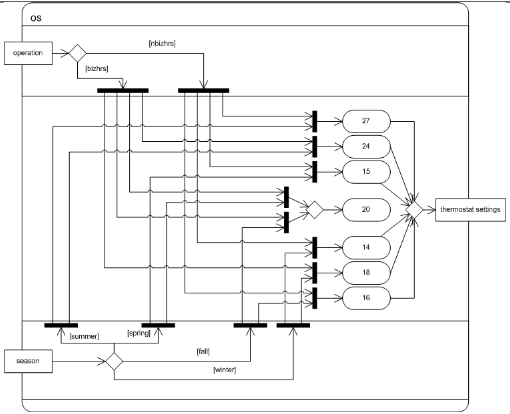
Table id: 3 - th



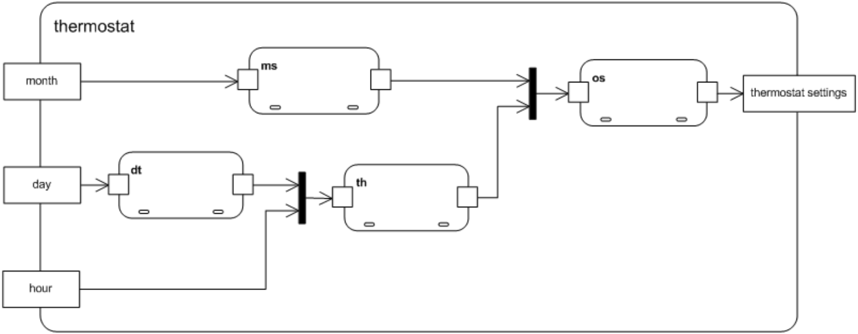
Activity diagram corresponding to XTT OS table

(?) operation	(?) season	(->) hermostat_setting
= nbizhrs	= summer	= 27
= bizhrs	= summer	= 24
= nbizhrs	= spring	= 15
= bizhrs	= spring	= 20
= nbizhrs	= winter	= 14
= bizhrs	= winter	= 18
= nbizhrs	= fall	= 16
= bizhrs	= fall	= 20

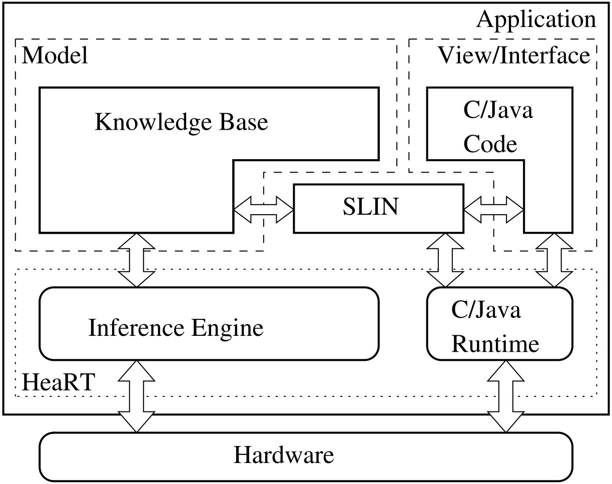
Table id: 4 - os



Activity diagram for the whole thermostat



Application integration in HeKatE



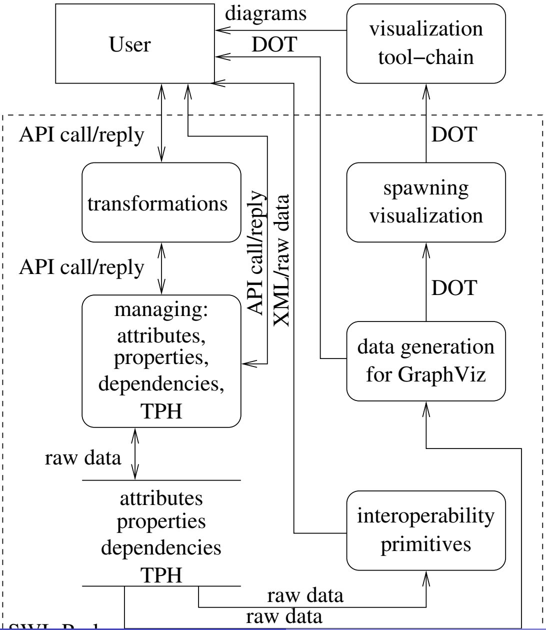
Overview

- *ARD*+ design tool → VARDA
- XTT^2 editor → HQEd
- XML-based knowledge exchange
- Prolog-based inference engine (HeaRT) (in the works)

VARDA intro

- a design tool for ARD+ [?]
- FIXME

VARDA architecture

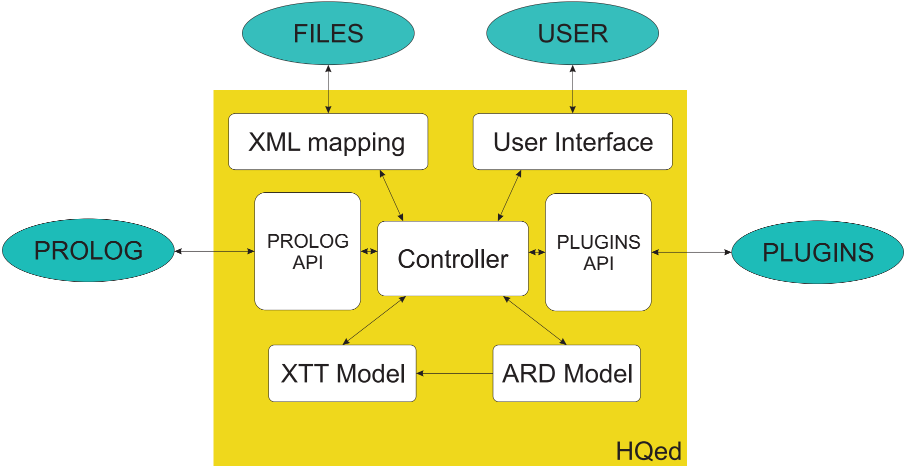


HQEd intro

[?]

- a complex XTT^2 editor
- FIXME

HQEd architecture



Knowledge Markup

- HML – Hekate Markup Language
- XSLT-based translators
- FIXME

Description

ARD model

XTT model

Description

ARD model

XTT model

Description

ARD model

Conclusions

Future developments

-

nxt
embedded
bizrules

Relevant papers

The End

Thank you for your attention!
Any questions?

HeKatE Web Page: http://hekate.ia.agh.edu.pl

Powered by LaTeX