

Combining ontologies with DSLs

Reasoning Web 2010 Summer School

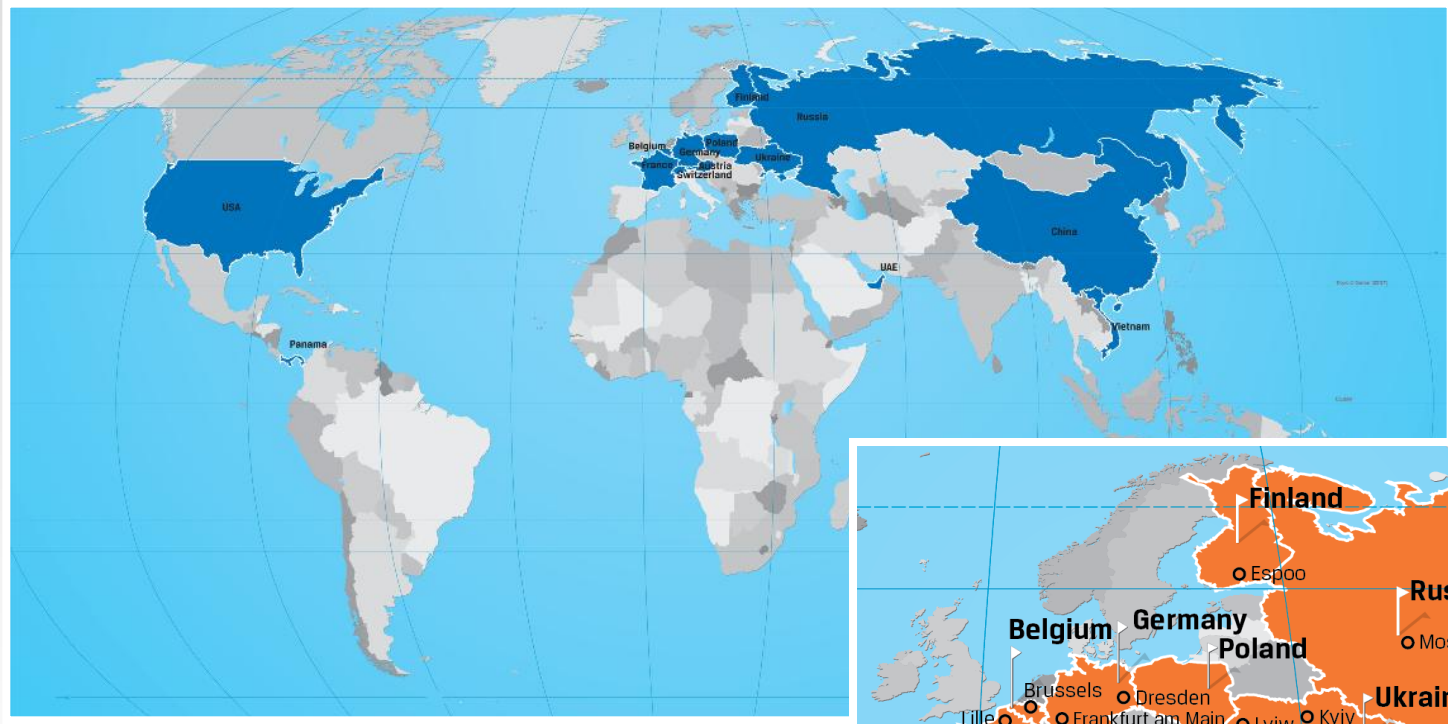
Krzysztof Miksa, Paweł Sabina, Marek Kasztelnik

Comarch SA

Dresden

2th September 2010

MOST - Marrying Ontology and Software Technology



Established in 1993
Headquarters – Krakow, Poland
Over 3000 employees worldwide

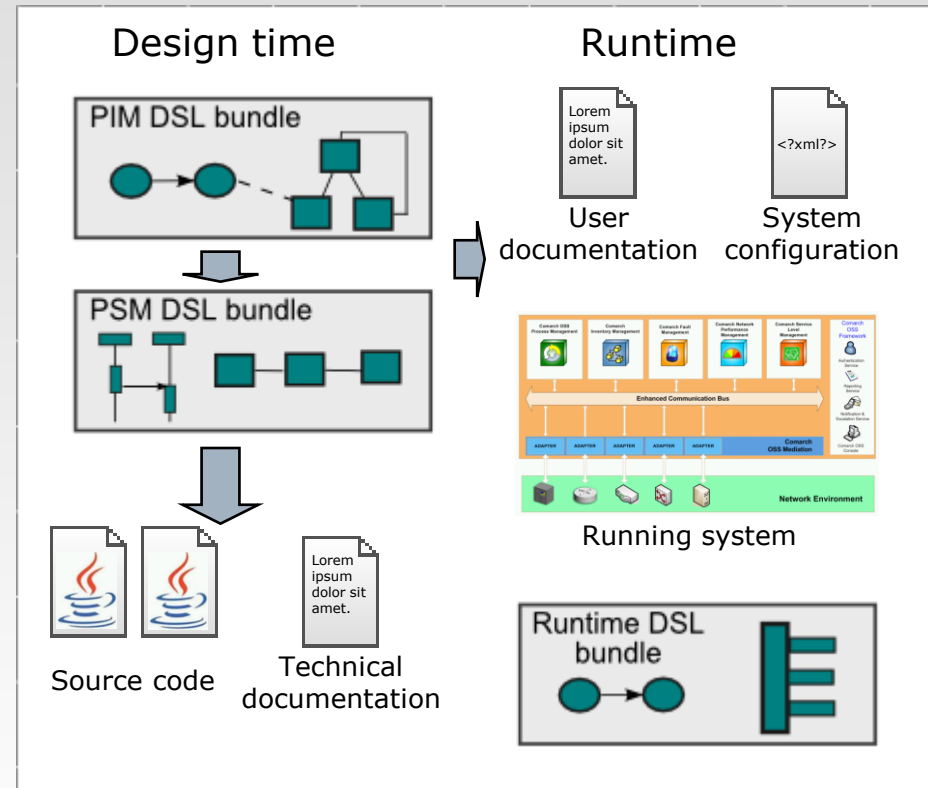
Agenda



- MDE & DSLs
- Case study
 - Motivation
 - 2D metamodelling
 - Physical Devices Ontology
 - DSL and OWL integration
 - Implementation of guidance services
- Evaluation

Why do we want model domain specifically?

- Precisely fit your needs
 - Productivity
 - Quality
- Having a DSL is a matter of hours
 - E.g. EMF
- Generate code
 - ...and other things
 - MDSO
- Interpreted at runtime



- Tools interoperating with DSL should have access to its unambiguous semantics
 - Otherwise: the interpretation of DSL constructs is hidden within the tools
 - ... or left out to the user
 - Important also for interoperability of DSLs
- Solution
 - *Provide formal semantics along language definition*

Problem description



- The domain of the case study:
Managing physical devices inventory within OSS system
 - OSS = *Operations Support Systems*
 - Computer systems used by telecommunications service providers
 - Aid the network operators in their daily work
 - Abstract from telecom business (e.g. billing) as opposed to Business Support Systems
 - Inventory Management
 - Core part of OSS
 - Management of telecom assets: sites, equipment, network links
 - Physical devices management
 - Important part of Inventory Management
 - Switches, routers, splitters, etc.
-

Why this is complicated?

- Equipment is modular
 - Devices have slots
 - Different cards can be plugged into the slots
 - Compatibility issues
 - Card constraints
- State-of-the-art in OSS
 - Easy access to extensive databases storing attributes of component types
 - No simultaneous support to the user by answering questions that involve sophisticated constraints
 - Custom solutions, hard coded constraints



- Questions:
 - Are the instances consistent with their types?
 - If not why?
 - Is the definition of the device type consistent?
 - What can I put into this slot?
 - Which cards?
 - Which types of cards?
 - Which device can I put this card in?
 - In which slot?
 - How can I repair my invalid device?

- *If the semantics is within the model*
 - *We can make use of semantic reasoning*
 - *These questions will remain simple*

2D metamodelling

- Physical Devices modelling

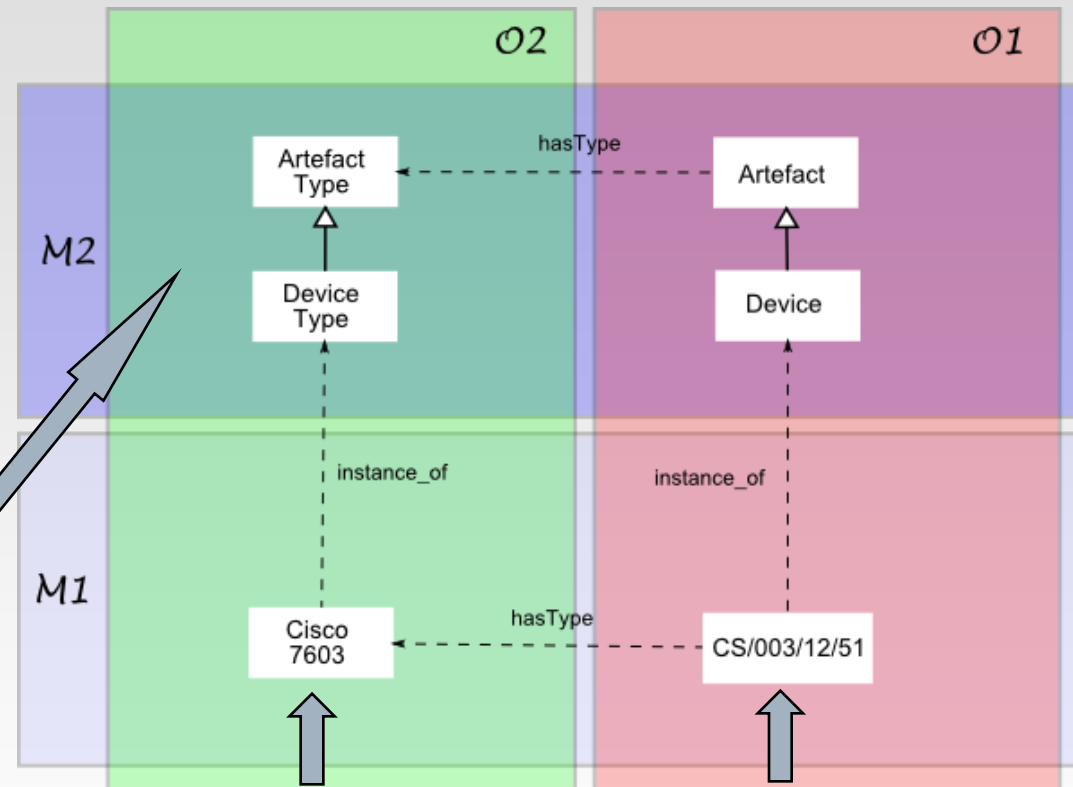
- Device types
- Instances of device types

- Two user roles

- Device Expert
- Device Modeller



Language Designer
(developer)

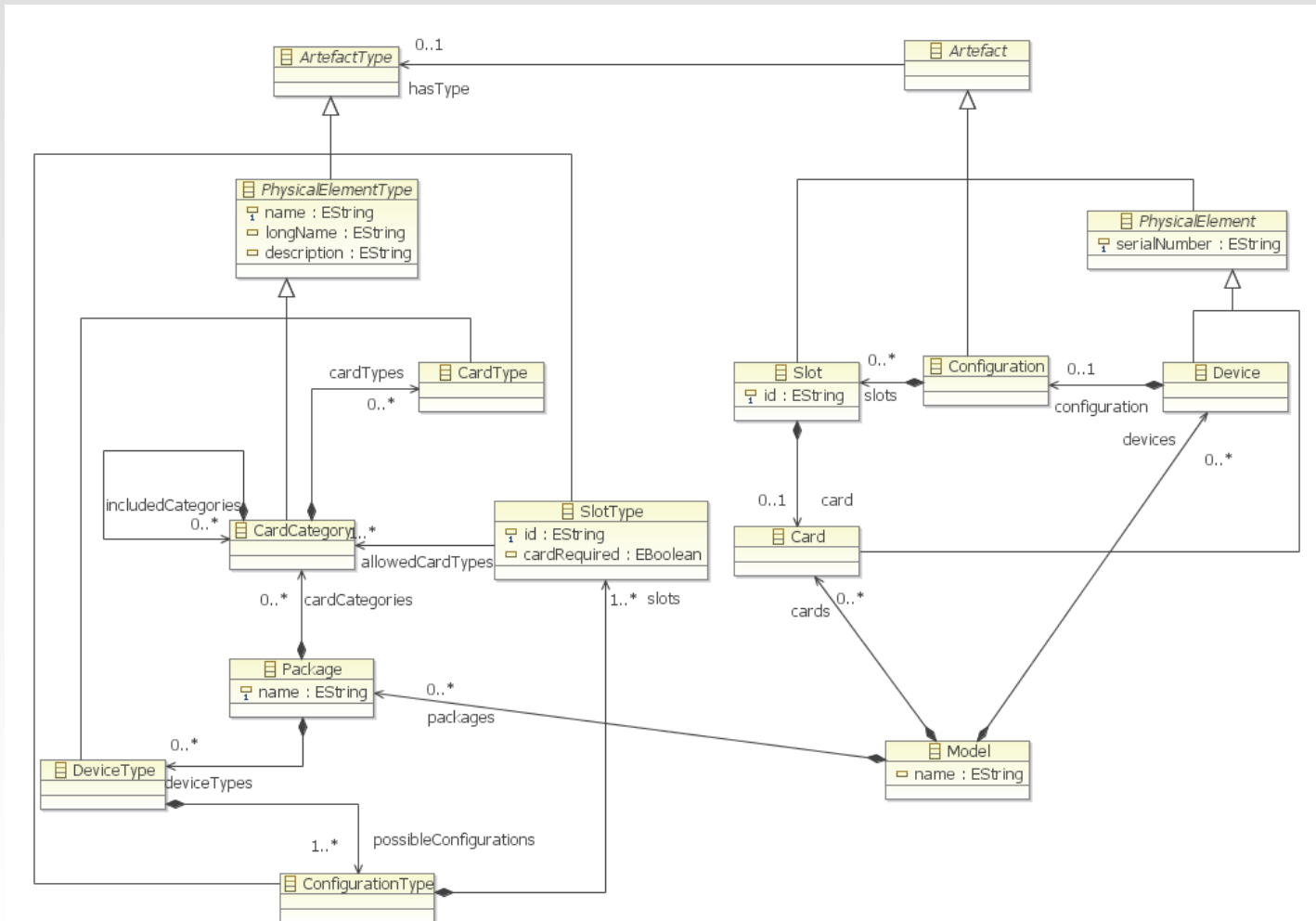


Device Expert



Device Modeller

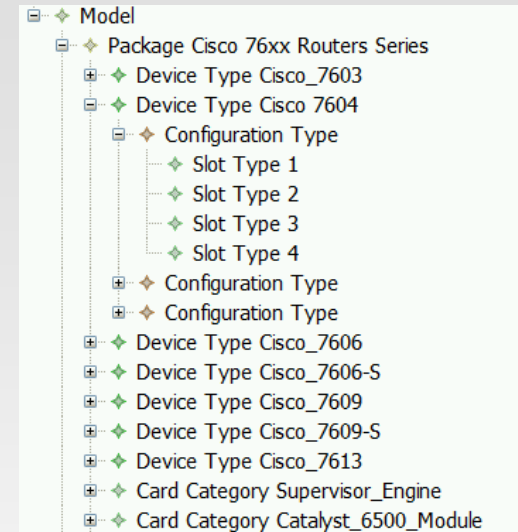
PDDSL abstract syntax



PDDSL concrete syntax

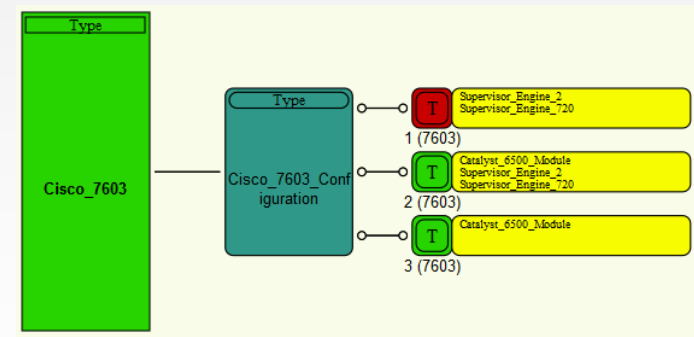
□ Demo

- Tree based editor
- Graphical syntax
- Textual syntax

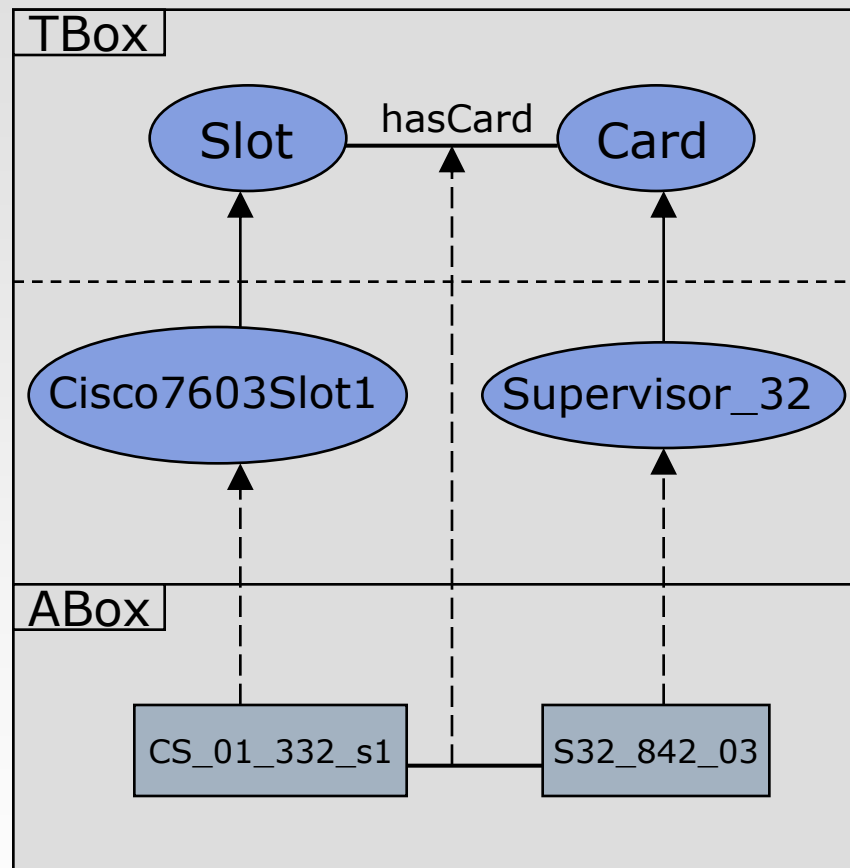


```

DeviceType "Cisco_7604" longName : "Cisco 7604 Chassis"
  allowed : {
    PossibleConfiguration "Cisco_7604_Configuration_1" {
      Slot "1" allowed : "Supervisor_Engine_32" required : true
      Slot "2" allowed : "Supervisor_Engine_32" "Catalyst_6500_Module" required : false
      Slot "3" allowed : "Catalyst_6500_Module" required : false
      Slot "4" allowed : "Catalyst_6500_Module" required : false
    }
    PossibleConfiguration "Cisco_7604_Configuration_2" {
      Slot "1" allowed : "Supervisor_Engine_720" required : true
      Slot "2" allowed : "Catalyst_6500_Module" "Supervisor_Engine_720" required : false
      Slot "3" allowed : "Catalyst_6500_Module" required : false
      Slot "4" allowed : "Catalyst_6500_Module" required : false
    }
    PossibleConfiguration "Cisco_7604_Configuration_3" {
      Slot "1" allowed : "Route_Switch_Processor_720" required : true
      Slot "2" allowed : "Catalyst_6500_Module" "Route_Switch_Processor_720" required : false
      Slot "3" allowed : "Catalyst_6500_Module" required : false
      Slot "4" allowed : "Catalyst_6500_Module" required : false
    }
  }
}
  
```



- TBox
 - Core concepts
 - Classes: Device, Slot, Card, ...
 - Properties: hasCard, hasSlot, ...
 - Device types
 - Cisco 7603, ...
- ABox
 - Instances
- Domain Closure
 - Depending on the use case
- Unique Names Assumption



□ Demo

```
Class: Cisco7603Configuration
  EquivalentTo: Configuration and
  # cardinality restriction on slots:
  hasSlot exactly 3 Slot and
  # required cards restriction:
  (hasSlot some (hasCard some Supervisors and id value 1)) and
  #optional card restriction:
  (hasSlot only (((hasCard some Supervisors and id value 1)) or
    ((hasCard some Supervisors and id value 2) or
      (hasCard some Hot_Swappable_OSM and id value 2)) or
    ((hasCard some Hot_Swappable_OSM and id value 3) or
      (hasCard some SPA_interface_processors and id value 3))))))
```

```
Namespace: pd <http://www.comarch.com/oss/pd.owl#>
Ontology: <http://www.comarch.com/oss/pd-ext.owl>
```

```
Class: pd:Cisco7603Configuration
  SubClassOf:
    ((pd:containsCard some pd:Hot_Swappable_OSM
    and (pd:containsCard some pd:Supervisor_engine_720))
    or (pd:containsCard only (not (pd:Hot_Swappable_OSM))))
```

Integrating DLS and OWL



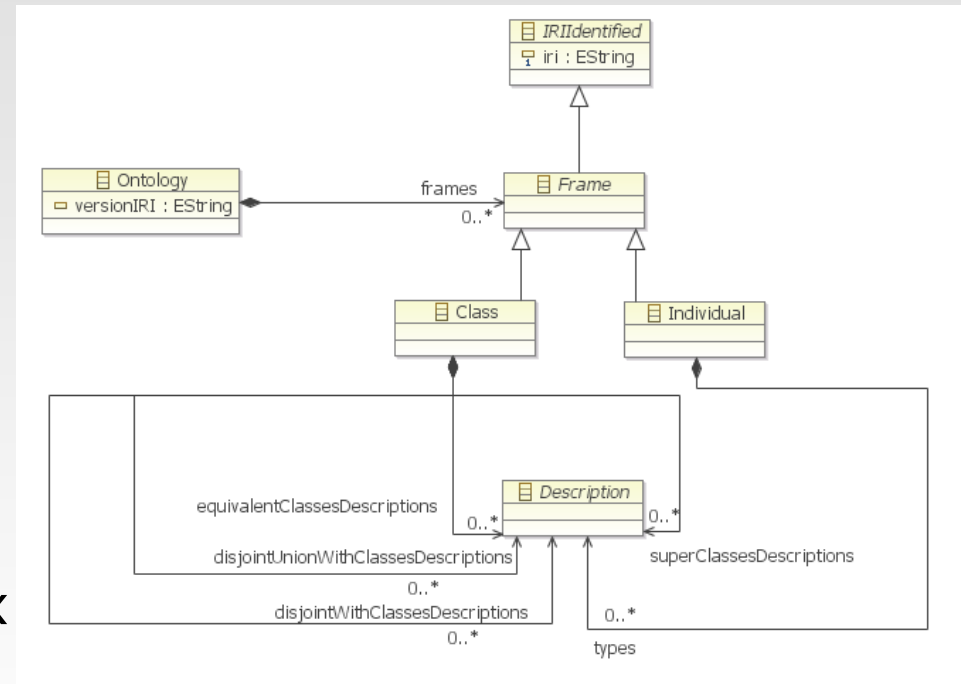
- Goal
 - Take profit from the **advantages of ontologies**, but...
 - still use the **productive DSL**
- Solution: integrate DSL and OWL
 - within modelling environment
- Integration process



Represent OWL2 in the metamodelling technical space

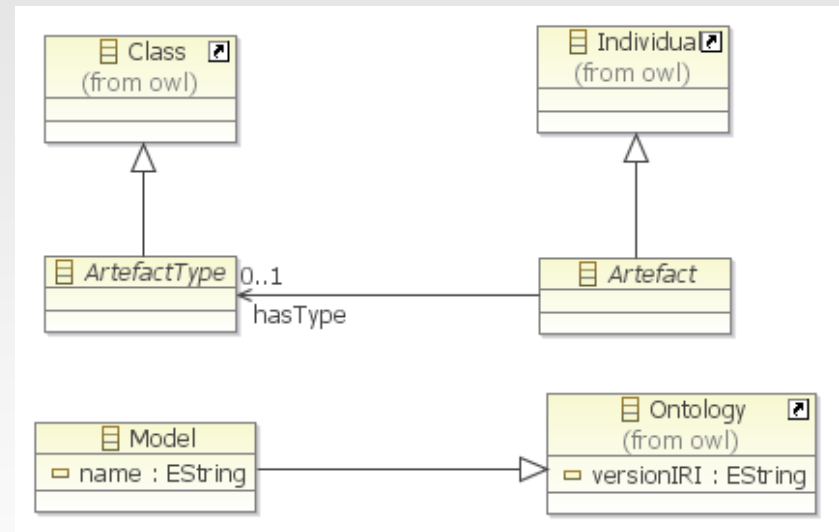


- OWL2 Manchester Syntax
 - Easy to use by the software developers
 - Used in Protégé
 - Already available from *emftext.org*
 - Metamodel
 - Concrete syntax
 - Concrete syntax can be directly parsed by OWL API



Integrate the abstract syntaxes

- ❑ PDDSL ArtefactType *is an* OWL Class
- ❑ PDDSL Artefact *is an* OWL Individual
- ❑ PDDSL Model *is an* OWL Ontology



Integrate the concrete syntaxes

```
CardCategory ::= !"CardCategory" iri[","]
(
  (annotations !1)
  | ("SubClassOf:" superClassesDescriptions
    ("," superClassesDescriptions)* !1)
  | ("EquivalentTo:" equivalentClassesDescriptions
    ("," equivalentClassesDescriptions)* !1)
  | ("DisjointWith:" disjointWithClassesDescriptions
    ("," disjointWithClassesDescriptions)* !1)
  | ("DisjointUnionOf:" disjointUnionWithClassesDescriptions
    ("," disjointUnionWithClassesDescriptions)* !1)
)*
("longName" ":" longName[","])? ("description" ":" description[","])?
(!"cardTypes" ":" "{" cardTypes* !0"}")?
(!"includedCategories" ":" "{" includedCategories* !0"}")? ;
```

Integrated Language



- PhysicalDevice DSL (PDDSL)
 - Structural modelling
- OWL2-DL
 - Semantic constraints which reference PDDSL elements

DeviceType "Cisco_7603"

SubClassOf: pd_hasConfiguration **some** (pd_hasSlot **some** (pd_hasCard **some** Cisco_7600_SIP))

longName : "CISCO 7603 CHASSIS" **description** : "The Cisco® 7603 Router is a high-performance ..." **allowed** : {

PossibleConfiguration "Cisco_7603_Configuration" {

Slot "1" allowed : "Supervisor_Engine_2" "Supervisor_Engine_720" **required** : true

Slot "2" allowed : "Supervisor_Engine_2" "Supervisor_Engine_720" "Catalyst_6500_Module"

required : false

Slot "3" allowed : "Catalyst_6500_Module" **required** : false

Device serialNumber : "cisco_7603" **hasType** : "Cisco_7603"

configuration :

```
{  
  Slot id : "1" : Card serialNumber : "supervisor_2_5" hasType : "WS-X6K-S2U-MSFC2"  
  Slot id : "2" :  
  Slot id : "3" :  
}
```

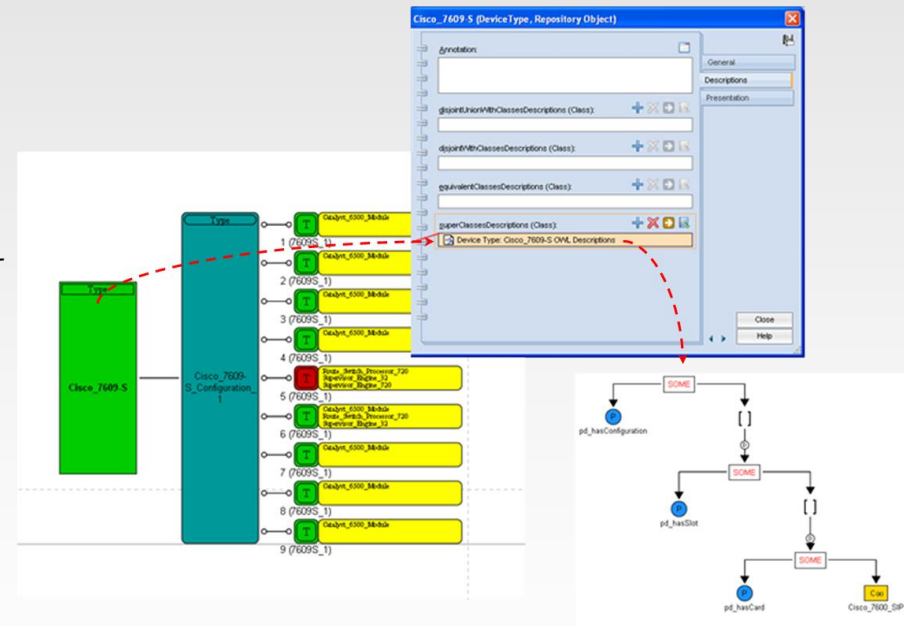
Concrete syntaxes

- Demo
 - Integrated textual syntax
 - Integrated graphical syntax

```
SubClassOf: pd_hasConfiguration some
  ( pd_hasSlot some
    ( pd_hasCard some Cisco_7600_SIP ) )
```

```
longName : "CISCO 7609-S CHASSIS", allowed : {
  PossibleConfiguration "Cisco_7609-S_Configuration_1" {
    Slot "1" allowed : "Catalyst_6500_Module"
    required : false
    Slot "2" allowed : "Catalyst_6500_Module"
    required : false
```

```
...
}
}
```



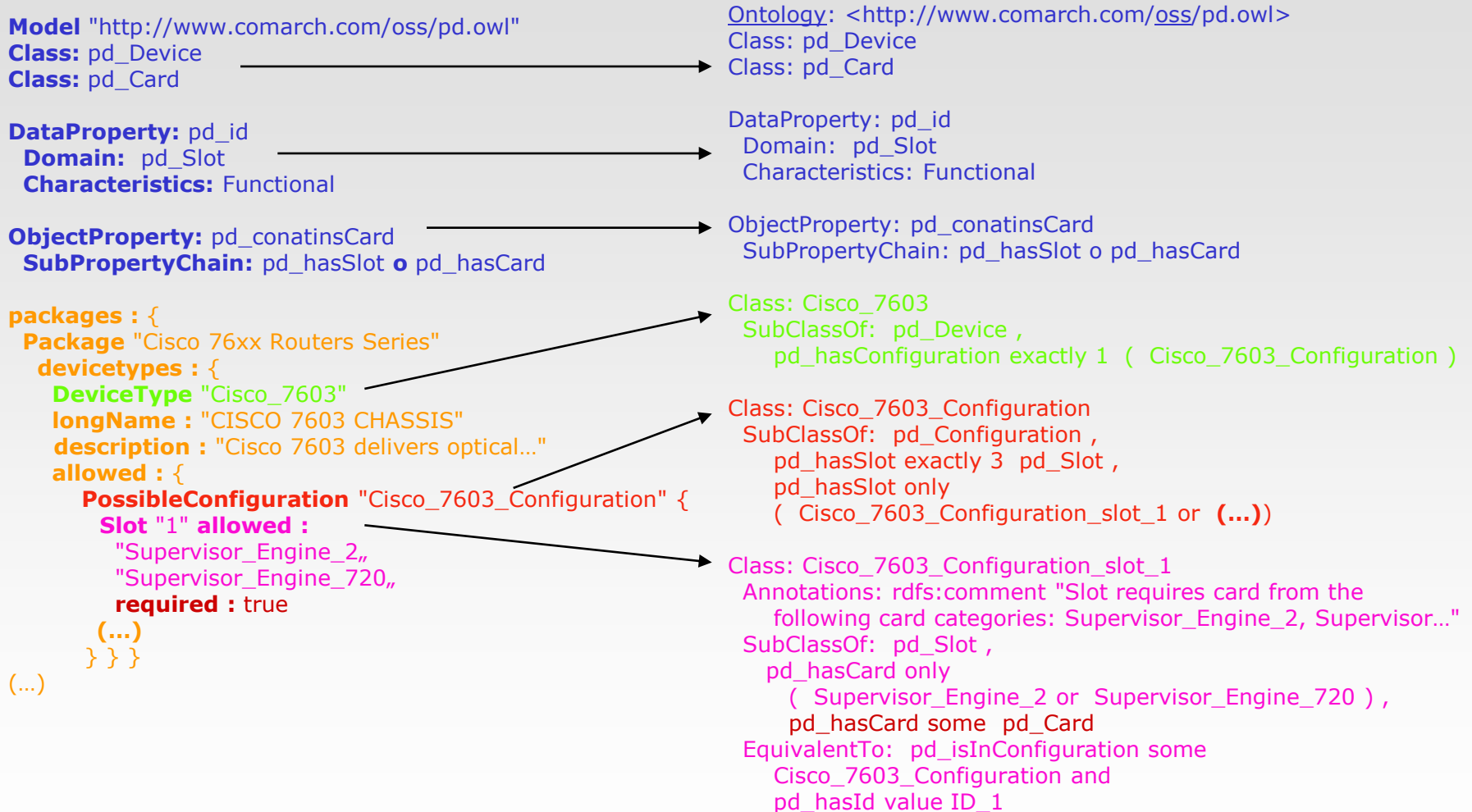
Transform to OWL2



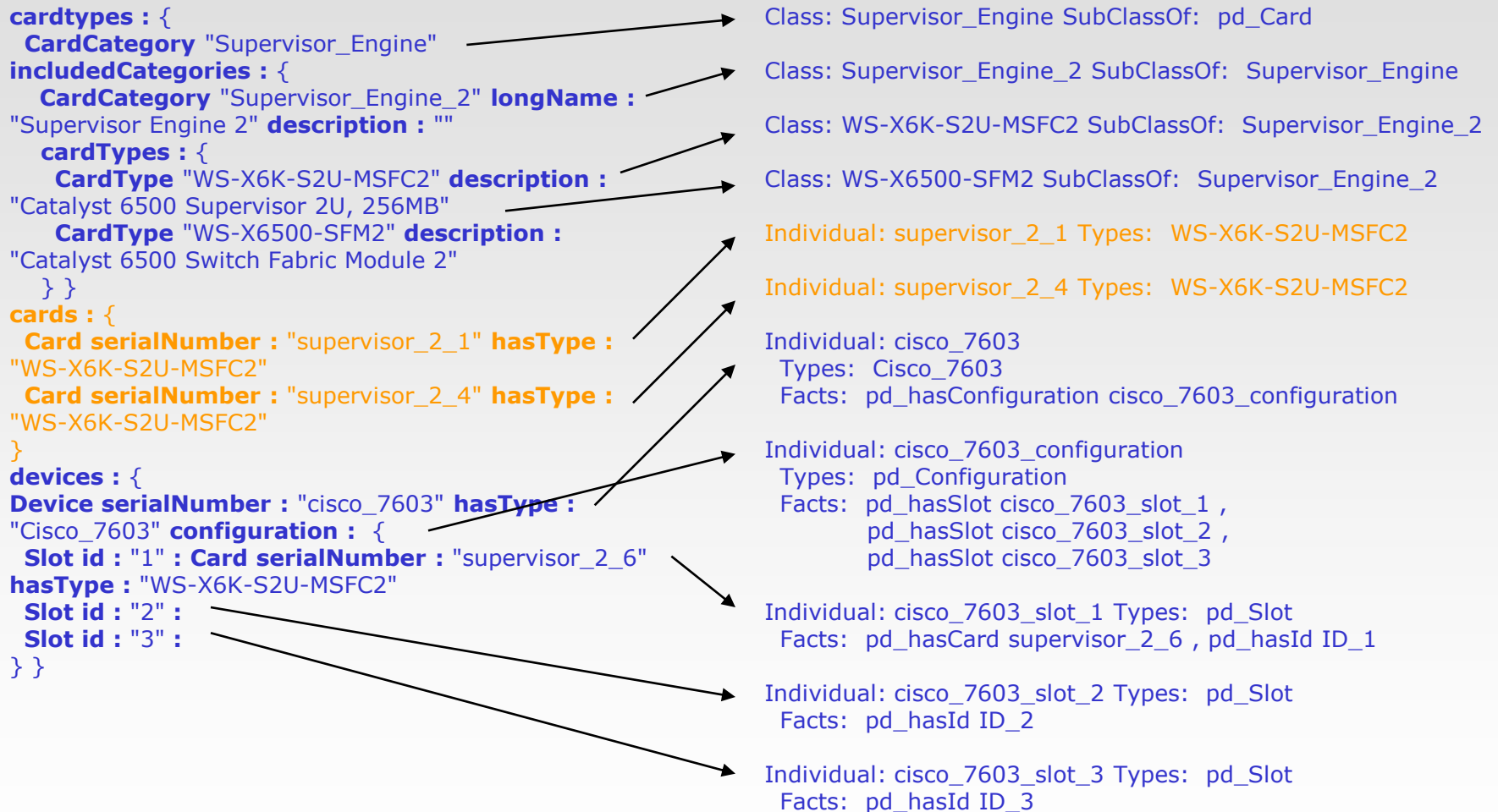
- Goal: define semantics of PDDSL constructs
- E.g. what is the meaning of *allowedCardTypes* association
- Informally: only the cards listed in *allowedCardTypes* can be connected into the slot
- Implemented as QVT Operational transformation
 - Model-to-model
- Another goal
 - Transform custom containment tree into flat frame representation
 - Map identifiers

```
mapping inout SlotType::updateSlotType() {  
    // Define a specific Slot subclass containing  
    // only allowed CardTypes, like in example:  
    //  
    // pd.hasCard only (Supervisor_Engine.2  
    //                   or Supervisor_Engine.720)  
    self.superClassesDescriptions += object ObjectPropertyOnly {  
        featureReference := object FeatureReference {  
            feature := hasCardProperty;  
        };  
        primary := object NestedDescription {  
            description := object Disjunction {  
                self.allowedCardTypes -> forEach (i) {  
                    conjunctions += object ClassAtomic{  
                        clazz := i;  
                    }  
                }  
            }  
        }  
    };  
}  
  
// Move all OWL::Class objects  
model.frames += model.allSubobjects() [OWL::Class] -> sortedBy(iri);
```

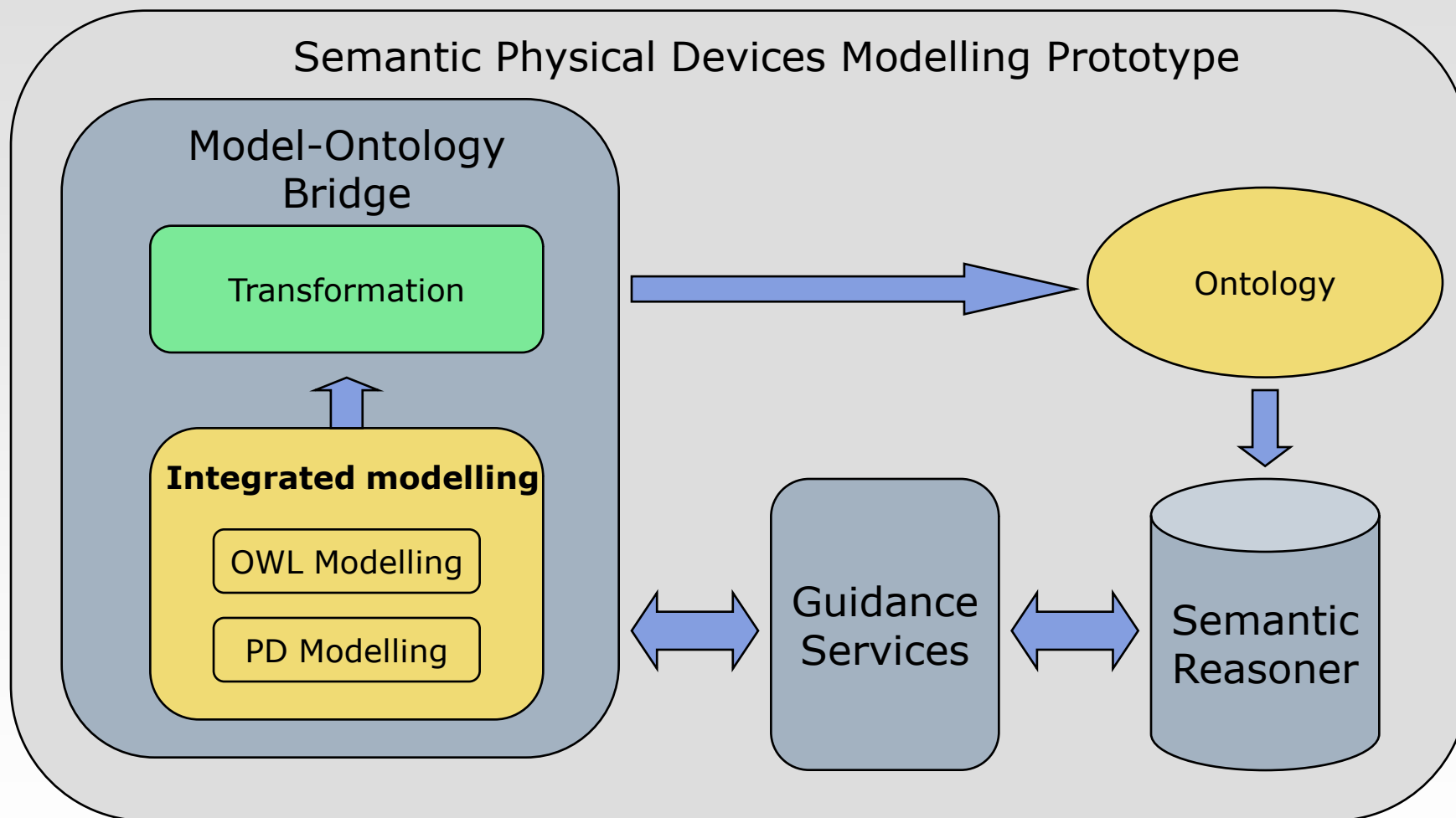
Transformation example



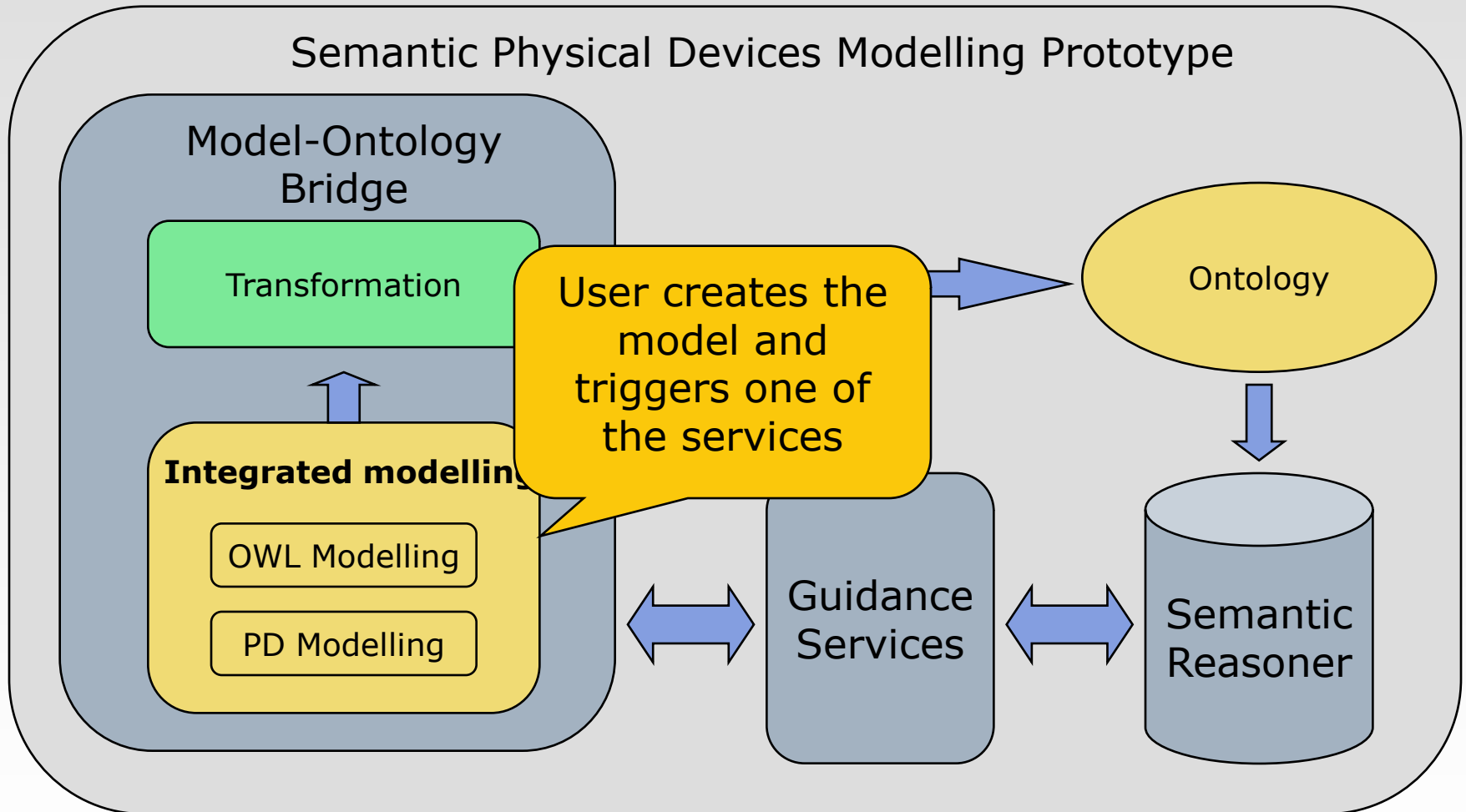
Transformation example cont.



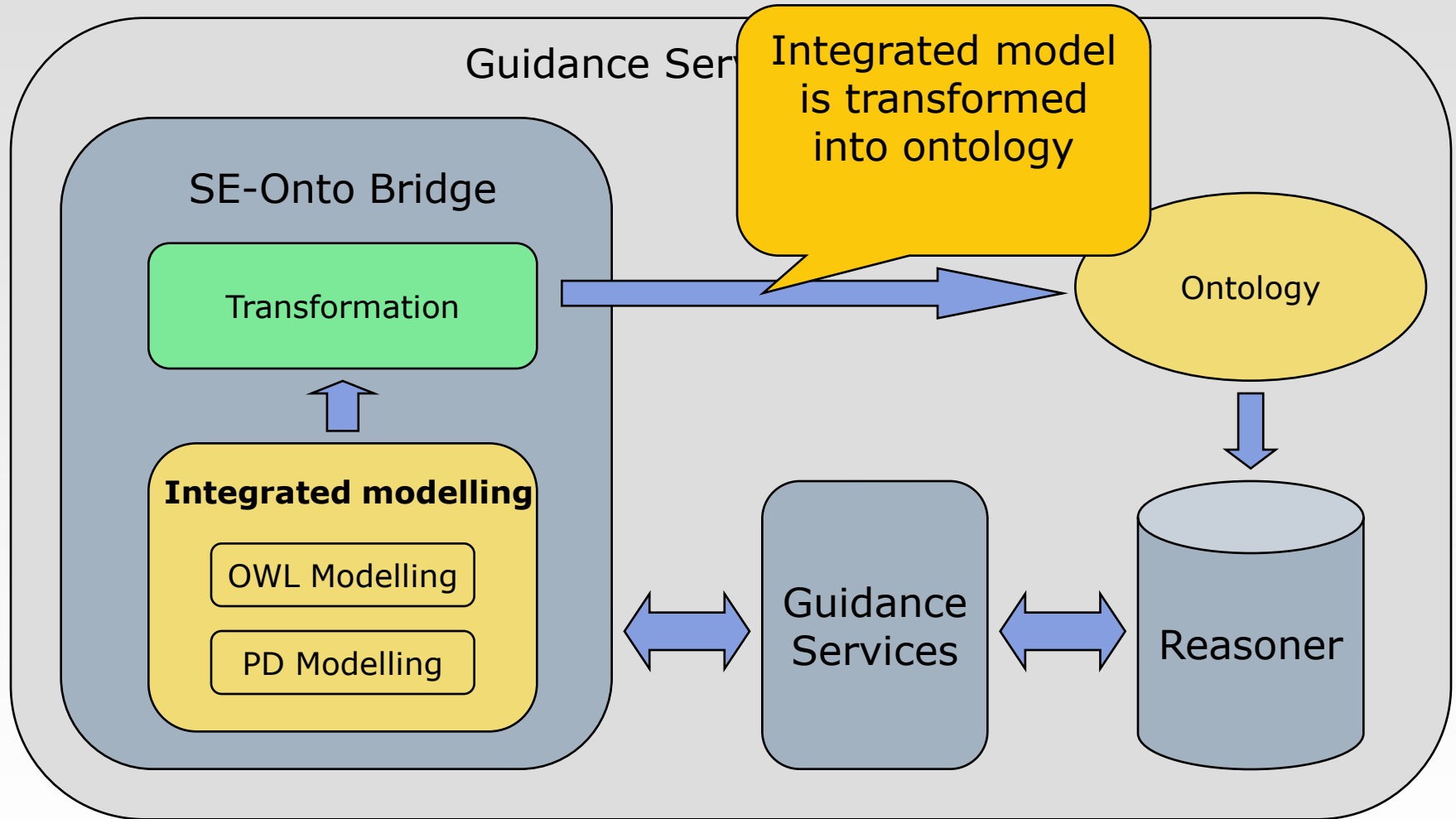
Integrated modelling environment with consistency guidance



Integrated modelling environment with consistency guidance

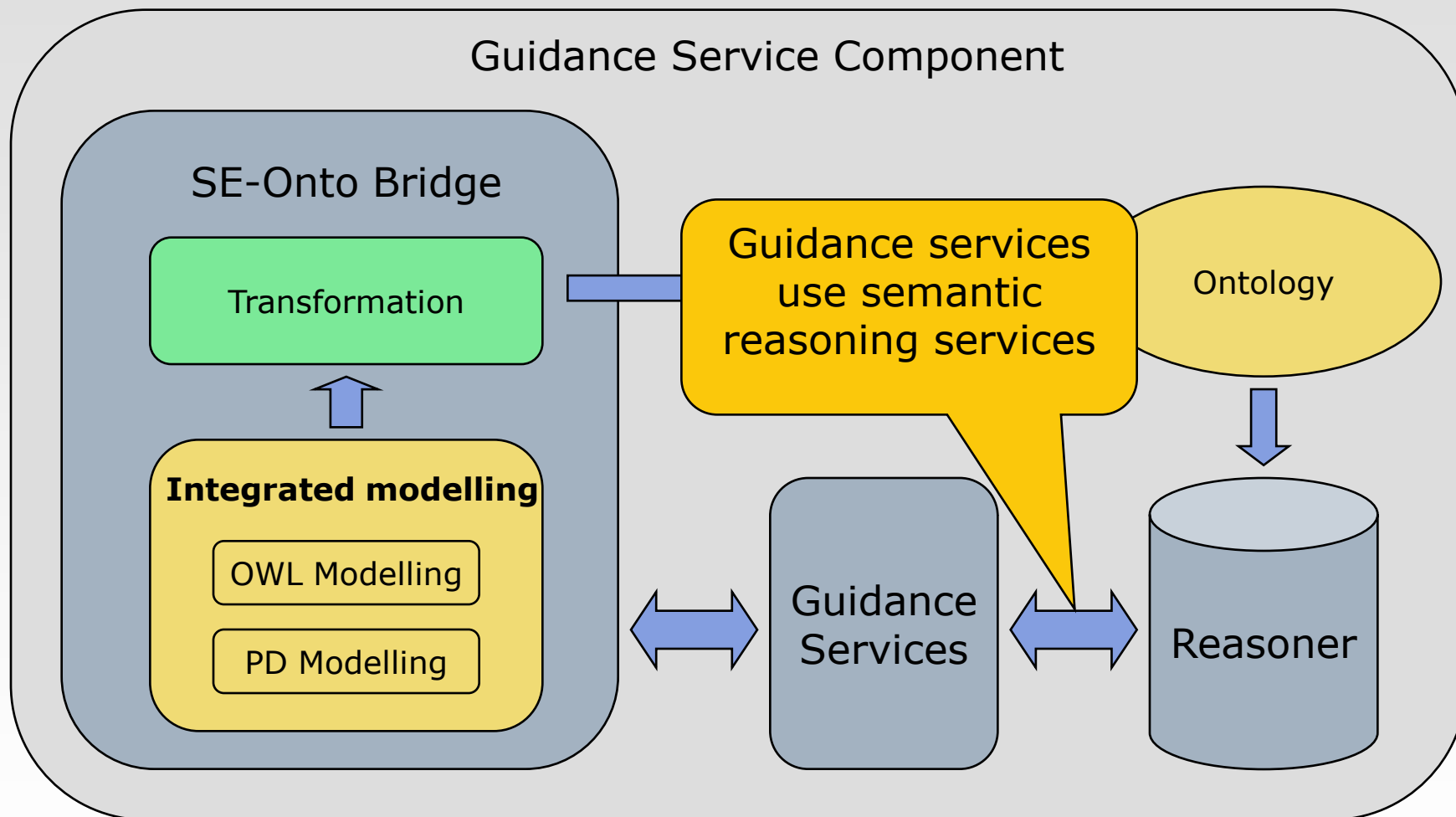


Integrated modelling environment with consistency guidance

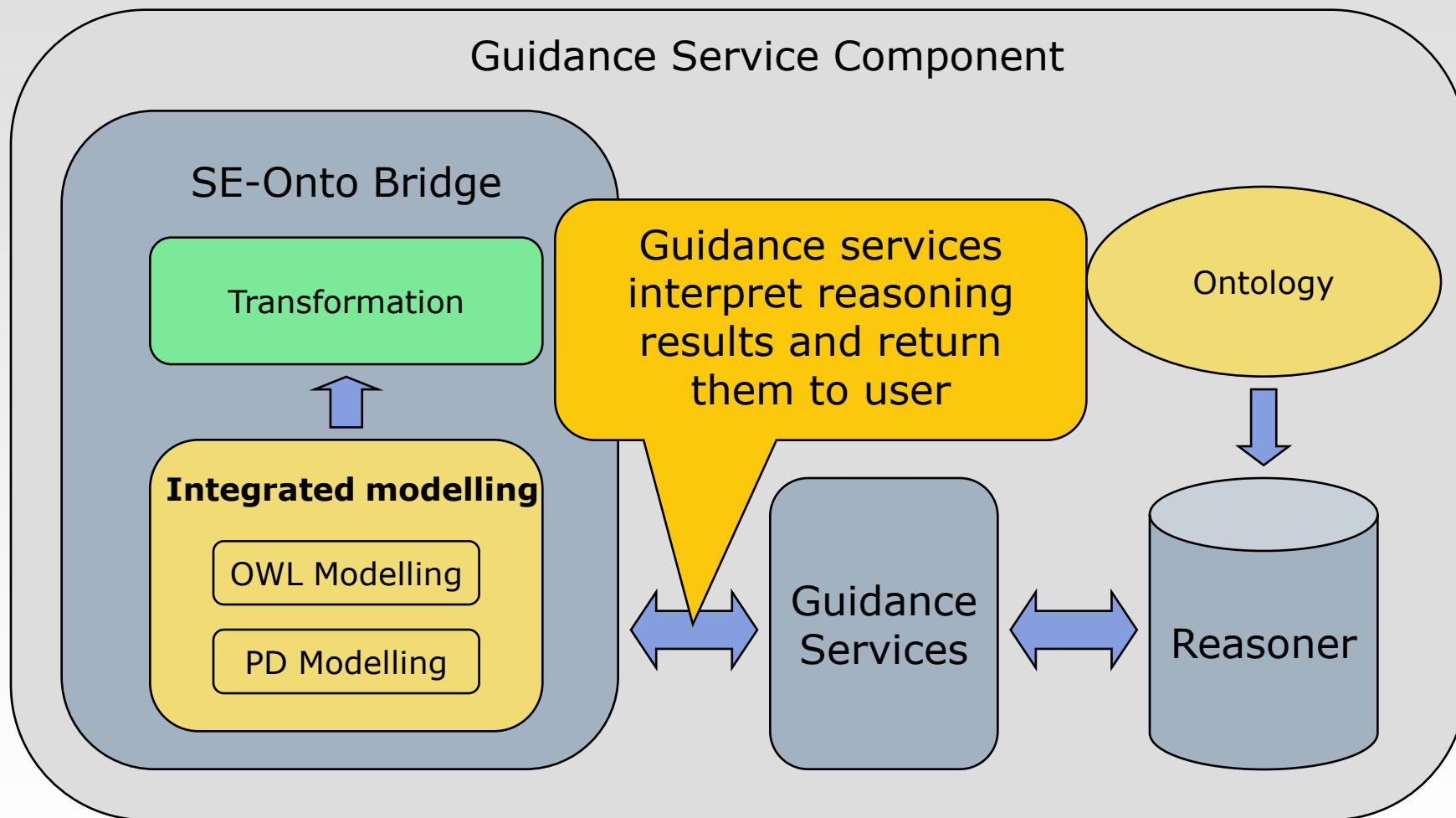


Integrated modelling environment with consistency guidance

Guidance Service Component



Integrated modelling environment with consistency guidance



- **UC-1:** Detect errors in physical device type definition
- **UC-2:** Find wrongly configured instances of devices and explain errors
- **UC-3:** Suggest card categories which are allowed in a slot

Implementation of the use cases



Use case	Description	Reasoning services
UC-1	Detect errors in physical device type definition	Satisfiability checking
UC-2	Find wrongly configured instances of devices and explain errors	Consistency checking, Explanations
UC-3	Suggest card categories which are allowed in a slot	Subsumption reasoning

UC-1: user model



DeviceType "Cisco_7603"

longName : "CISCO 7603 CHASSIS" description : "Cisco 7603 delivers optical LAN, WAN..."
allowed : {
 PossibleConfiguration "Cisco_7603_Configuration" {
 Slot "1" allowed : "Supervisor_Engine_2" "Supervisor_Engine_720" required : true
 Slot "2" allowed : "Supervisor_Engine_2" "Supervisor_Engine_720" "Catalyst_6500_Module"
 required : false
 Slot "3" allowed : "Catalyst_6500_Module" required : false
 }
}

UC-1: error



DeviceType "Cisco_7603"

SubClassOf: pd:hasConfiguration some (pd:hasSlot some (pd:hasCard some Cisco_7600_SIP))

longName : "CISCO 7603 CHASSIS" description : "Cisco 7603 delivers optical LAN, WAN..."

allowed : {

PossibleConfiguration "Cisco_7603_Configuration" {

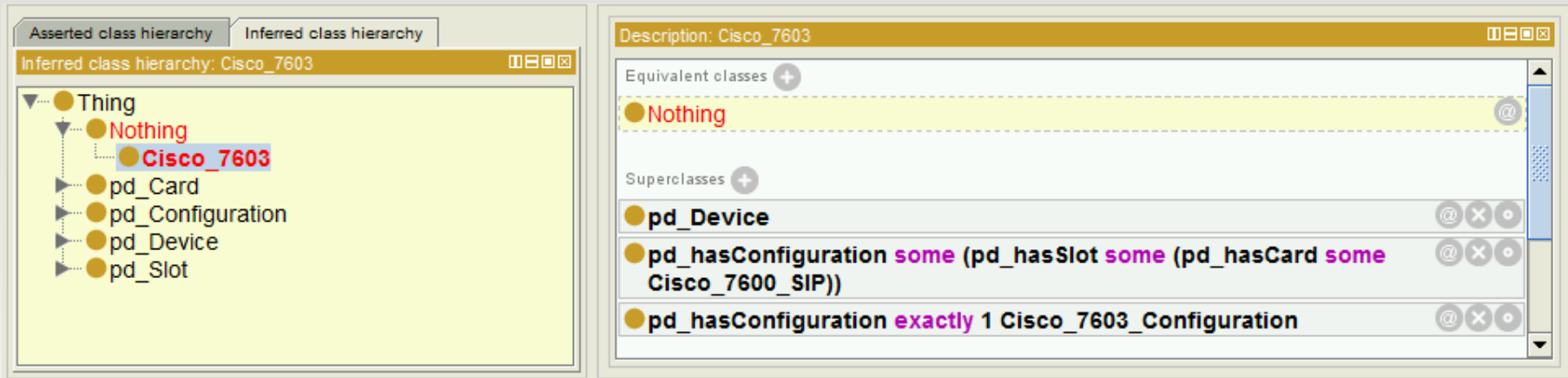
Slot "1" allowed : "Supervisor_Engine_2" "Supervisor_Engine_720" required : true

Slot "2" allowed : "Supervisor_Engine_2" "Supervisor_Engine_720" "Catalyst_6500_Module" required : false

Slot "3" allowed : "Catalyst_6500_Module" required : false

**}
}**

UC-1: implementation



The screenshot displays two panels from a software application. The left panel, titled 'Inferred class hierarchy: Cisco_7603', shows a tree structure starting with 'Thing' at the root. Under 'Thing', there are 'Nothing' and 'Cisco_7603'. Below 'Nothing', there are 'pd_Card', 'pd_Configuration', 'pd_Device', and 'pd_Slot'. The right panel, titled 'Description: Cisco_7603', shows a list of 'Equivalent classes' and 'Superclasses'. The 'Equivalent classes' list includes 'Nothing'. The 'Superclasses' list includes 'pd_Device', 'pd_hasConfiguration some (pd_hasSlot some (pd_hasCard some Cisco_7600_SIP))', and 'pd_hasConfiguration exactly 1 Cisco_7603_Configuration'.

1. Find unsatisfiable classes in PD Ontology
2. Find corresponding set of *Artefacts* in PDDSL model
3. Report the *Artefacts* to the user

UC-2: user model



```
DeviceType "Cisco_7603"  
  longName : "CISCO 7603 CHASSIS" description : "Cisco 7603 delivers optical LAN, WAN..."  
  allowed : {  
    PossibleConfiguration "Cisco_7603_Configuration" {  
      Slot "1" allowed : "Supervisor_Engine_2" "Supervisor_Engine_720" required : true  
      Slot "2" allowed : "Supervisor_Engine_2" "Supervisor_Engine_720" "Catalyst_6500_Module",,  
        required : false  
      Slot "3" allowed : "Catalyst_6500_Module" required : false  
    }  
  }  
}  
  
Device serialNumber : "cisco_7603-INCORRECT" hasType : "Cisco_7603"  
configuration :  
{  
  Slot id : "1" :  
  Slot id : "2" :  
  Slot id : "3" :  
}  
  
CardCategory "Catalyst_6500_Module" includedCategories : {  
  CardCategory "Cisco_7600_Ethernet_Card_Line"  
  longName : "Cisco 7600 Series Ethernet Services Plus 20- and 40-Gbps Line Cards,,  
  cardTypes : {  
    CardType "7600-ES-2TG3C"  
    CardType "7600-ES-2TG3CXL,,  
  }  
}
```

UC-2: error



```
DeviceType "Cisco_7603"  
  longName : "CISCO 7603 CHASSIS" description : "Cisco 7603 delivers optical LAN, WAN..."  
  allowed : {  
    PossibleConfiguration "Cisco_7603_Configuration" {  
      Slot "1" allowed : "Supervisor_Engine_2" "Supervisor_Engine_720" required : true  
      Slot "2" allowed : "Supervisor_Engine_2" "Supervisor_Engine_720" "Catalyst_6500_Module",,  
        required : false  
      Slot "3" allowed : "Catalyst_6500_Module" required : false  
    }  
  }  
}  
  
Device serialNumber : "cisco_7603-INCORRECT" hasType : "Cisco_7603"  
configuration :  
{  
  Slot id : "1" : Card serialNumber : "ethernet_card" hasType : "7600-ES-2TG3C"  
  Slot id : "2" :  
  Slot id : "3" :  
}  
  
CardCategory "Catalyst_6500_Module" includedCategories : {  
  CardCategory "Cisco_7600_Ethernet_Card_Line"  
  longName : "Cisco 7600 Series Ethernet Services Plus 20- and 40-Gbps Line Cards,,  
  cardTypes : {  
    CardType "7600-ES-2TG3C"  
    CardType "7600-ES-2TG3CXL"  
  }  
}
```

UC-2: MIPS - Inconsistency explanation



```
7600-ES-2TG3C subClassOf Cisco_7600_Ethernet_Card_Line
Supervisor_Engine_2 subClassOf Supervisor_Engine
cisco_7603-INCORRECT type Cisco_7603
cisco_7603-INCORRECT_slot_1 pd_id 1
Cisco_7600_Ethernet_Card_Line subClassOf Catalyst_6500_Module
cisco_7603-INCORRECT pd_hasConfiguration cisco_7603-INCORRECT_configuration
Supervisor_Engine_720 subClassOf Supervisor_Engine
ethernet_card type 7600-ES-2TG3C
Cisco_7603_Configuration_slot_1 subClassOf pd_hasCard only
    Supervisor_Engine_2 or Supervisor_Engine_720
Functional pd_hasConfiguration
Cisco_7603 subClassOf pd_hasConfiguration exactly 1Cisco_7603_Configuration
cisco_7603-INCORRECT_slot_1 pd_hasCard ethernet_card
DisjointClasses(Catalyst_6500_Module
    Cisco_7600_SIP
    Supervisor_Engine)
pd_hasSlot inverseOf pd_isInConfiguration
Cisco_7603_Configuration_slot_1 equivalentTo pd_id value 1
    and pd_isInConfiguration some Cisco_7603_Configuration
cisco_7603-INCORRECT_configuration pd_hasSlot cisco_7603-INCORRECT_slot_1
```

UC-2: User-friendly explanations

- Inconsistent individuals:
 - cisco_7603-INCORRECT
 - cisco_7603-INCORRECT_configuration
 - cisco_7603-INCORRECT_slot_1
- Textual explanation from axiom annotations:
 - "Slot requires card from the following card categories: Supervisor_Engine_2, Supervisor_Engine_720"
 - Annotation of Cisco_7603_slot_1

Class: Cisco_7603_Configuration_slot_1

Annotations: rdfs:comment "Slot requires card from the following card categories: Supervisor_Engine_2, Supervisor_Engine_720"

SubClassOf: pd_Slot ,

pd_hasCard only

(Supervisor_Engine_2 or Supervisor_Engine_720) ,

pd_hasCard some pd_Card

Device validation explanation interpretation



7600-ES-2TG3C subClassOf Cisco_7600_Ethernet_Card_Line
Supervisor_Engine_2 subClassOf Supervisor_Engine
cisco_7603-INCORRECT type Cisco_7603
cisco_7603-INCORRECT_slot_1 pd_id 1
Cisco_7600_Ethernet_Card_Line subClassOf Catalyst_6500_Module
cisco_7603-INCORRECT pd_hasConfiguration cisco_7603-INCORRECT_configuration
Supervisor_Engine_720 subClassOf Supervisor_Engine
ethernet_card type 7600-ES-2TG3C
**Cisco_7603_Configuration_slot_1 subClassOf pd_hasCard only
Supervisor_Engine_2 or Supervisor_Engine_720**
Functional pd_hasConfiguration
Cisco_7603 subClassOf pd_hasConfiguration exactly 1Cisco_7603_Configuration
cisco_7603-INCORRECT_slot_1 pd_hasCard ethernet_card
DisjointClasses(Catalyst_6500_Module
Cisco_7600_SIP
Supervisor_Engine)
pd_hasSlot inverseOf pd_isInConfiguration
Cisco_7603_Configuration_slot_1 equivalentTo pd_id value 1
and pd_isInConfiguration some Cisco_7603_Configuration
cisco_7603-INCORRECT_configuration pd_hasSlot cisco_7603-INCORRECT_slot_1

UC-3: user model



```
DeviceType "Cisco_7603"  
  longName : "CISCO 7603 CHASSIS" description : "Cisco 7603 delivers optical LAN, WAN..."  
  allowed : {  
    PossibleConfiguration "Cisco_7603_Configuration" {  
      Slot "1" allowed : "Supervisor_Engine_2" "Supervisor_Engine_720" required : true  
      Slot "2" allowed : "Supervisor_Engine_2" "Supervisor_Engine_720" "Catalyst_6500_Module",,  
        required : false  
      Slot "3" allowed : "Catalyst_6500_Module" required : false  
    }  
  }  
}  
  
Device serialNumber : "cisco_7603-INCORRECT" hasType : "Cisco_7603"  
configuration :  
{  
  Slot id : "1" : Card serialNumber : "ethernet_card" hasType : "7600-ES-2TG3C"  
  Slot id : "2" :  
  Slot id : "3" :  
}  
  
CardCategory "Catalyst_6500_Module" includedCategories : {  
  CardCategory "Cisco_7600_Ethernet_Card_Line"  
  longName : "Cisco 7600 Series Ethernet Services Plus 20- and 40-Gbps Line Cards,,  
  cardTypes : {  
    CardType "7600-ES-2TG3C"  
    CardType "7600-ES-2TG3CXL"  
  }  
}
```

UC-3: Computing suggestions



□ Idea

1. What cannot be inserted to a slot given the current knowledge base?
2. Compute the complement
 - 1. is an OWA question
 - 2. requires local CWA
 - Answer both in terms of classes and individuals

UC-3: Computing suggestions - example



- Subclasses
 - *CardCategories* which are not allowed in the slot

- Instances
 - *Cards* which are not allowed in the slot

Query (class expression)

```
pd_Card and not (inv(pd_hasCard) value cisco_7603_slot_1)
```

Execute Add to ontology

Query results

Sub classes (4)

- Catalyst_6500_Module
- Cisco_7600_SIP
- Route_Switch_Processor_720
- Supervisor_Engine_32

Instances (8)

- ◆ card_c2_1
- ◆ card_c2_2
- ◆ card_es20_2
- ◆ card_es20_1
- ◆ card_es20_3
- ◆ supervisor_32_1

Super classes
 Ancestor classes
 Equivalent classes
 Subclasses
 Descendant classes
 Individuals

UC-3: Computing suggestions - complement



- Easier to perform in the modelling technical space
- OCL:
Model ->
allCardCategories() ->
excluding(disallowed)
- In OWL, requires CWA

Query (class expression)

```
pd_Card and not {card_c2_1, card_c2_2, card_es20_1, card_es20_2, card_es20_3, supervisor_32_1}
```

Execute Add to ontology

Query results

Sub classes (3)

- ☰ Cisco_7600_SIP
- Supervisor_Engine_2
- Supervisor_Engine_720

Instances (7)

- ◆ supervisor_2_6
- ◆ supervisor_720_3
- ◆ supervisor_2_3
- ◆ supervisor_2_1
- ◆ supervisor_720_2
- ◆ supervisor_720_1
- ◆ supervisor_2_4

Super classes
 Ancestor classes
 Equivalent classes
 Subclasses
 Descendant classes
 Individuals

UC-3: other considerations

- Cards allowed in a slot, given the current configuration
 - `pd_Card and not (inv(pd_hasCard) value cisco_7603_slot_1)`
- Cards allowed in a slot, in general
 - `pd_Card and not (inv(pd_hasCard) some Cisco_7603_Configuration_slot_1)`
- Cards allowed in a device, given the current configuration
 - `pd_Card and not (inv(pd_hasCard) some (inv(pd_hasSlot) some (inv(pd_hasConfiguration) cisco_7603)))`
- Cards allowed in a device, in general
 - `pd_Card and not (inv(pd_hasCard) some (inv(pd_hasSlot) some (inv(pd_hasConfiguration) Cisco7603)))`
- Which slots I can put a given car in
 - `pd_Slot and not (pd_hasCard value HS_OSM_1)`
- Which card types are compatible with a given card type
 - `pd_Card and not (inv(pd_hasCard) some (inv(pd_hasSlot) some (pd_hasSlot some (pd_hasCard some (Supervisor_32))))`
- What are the possible types of a device
 - `Device and not {cisco_11}`
- ...

Initial findings: positives



- Creation of DSL tools is easier
 - Purely declarative approach
 - Various questions can be asked
 - No need to hard code semantics in tools
- Support for two-dimensional metamodelling
 - SE metamodelling contributes support for *linguistic instantiation*
 - Ontologies contribute support for *ontological (domain) instantiation*
 - Needed in other areas: service modelling, connectivity modelling
- Simple but expressive DSL
 - PDDSL part contributes simplicity, productivity
 - OWL part contributes rich expressiveness
- Other usage scenarios exist

Initial findings: challenges



- High cost of implementation
 - Lots of manual work
 - Costly to reuse for other languages
 - Possible solution:
 - Generalize the approach
 - Product-line of Ontology Supported Two Dimensional DSL
- Multiple explanations and scalability
 - Single explanation result in early compiler style of work
 - Not possible with presently available tools
 - Expensive service: scalable implementation needed
 - Possible solution:
 - Approximate to OWL EL

Thank you

Further information:

Krzysztof.Miksa@comarch.com

Pawel.Sabina@comarch.com

Marek.Kasztelnik@comarch.com