National Vulnerability Database Product Ontology

Paul Cichonski
Booz Allen Hamilton / NIST

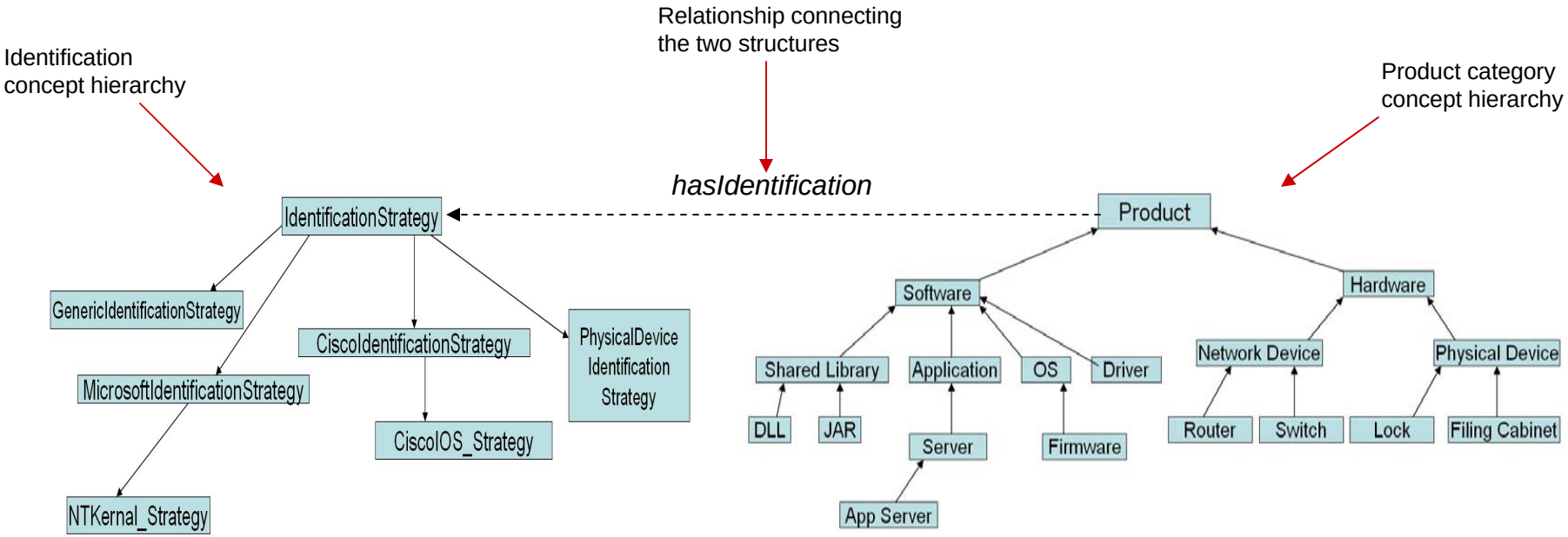
Agenda

- Overview of Ontology Goals
- Overview of NVD Product Ontology
- Overview of Inferencing Use Case for Querying Across Product Ranges.

NVD Product Ontology Goals

- Ontology must support NVD's primary use case involving making statements of applicability between IT concepts (e.g. CVEs, Checklists) and IT products.
- Ontology must support the ability to make statements of applicability at various levels of abstraction and across ranges of products (e.g. Microsoft Windows version 4.3 to 5.6).
- Ontology must support the ability to capture granular product identification data which may vary on a per product basis.
- Ontology must support the Common Platform Enumeration which is the standardized method for naming IT products.

High-level NVD Ontology Overview



= <owl:Class>

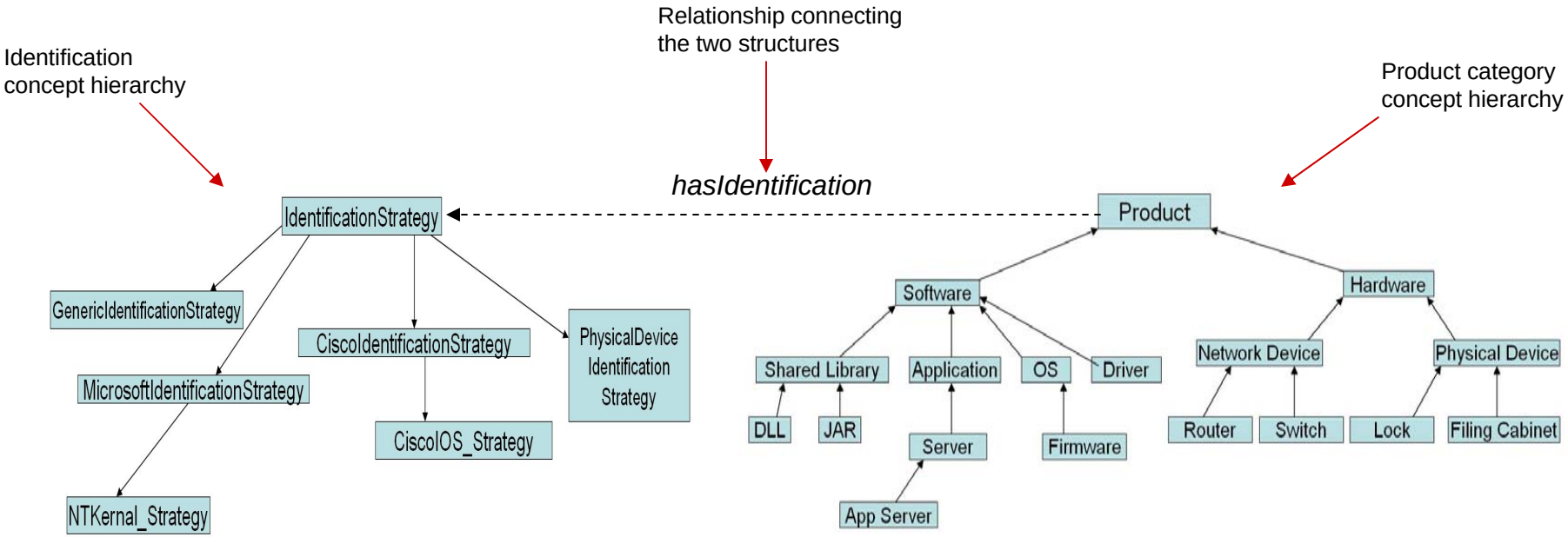
= <rdfs:subClassOf>

ABC = <rdf:Property>

Structure of the Ontology

- NVD Ontology models two separate concept structures as formal “is-a” hierarchies.
  - Category concept hierarchy
  - Identification concept hierarchy
- NVD Ontology also includes other types of semantic relationships.
  - Relationships between applications and codebases (“made up of” relationships)
  - Explicit differences between sets of products created by defining disjoint sets (e.g. hardware vs. software products)

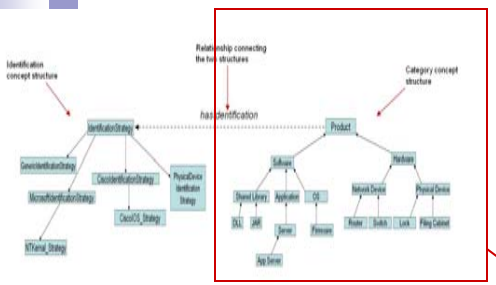
High-level NVD Ontology Overview



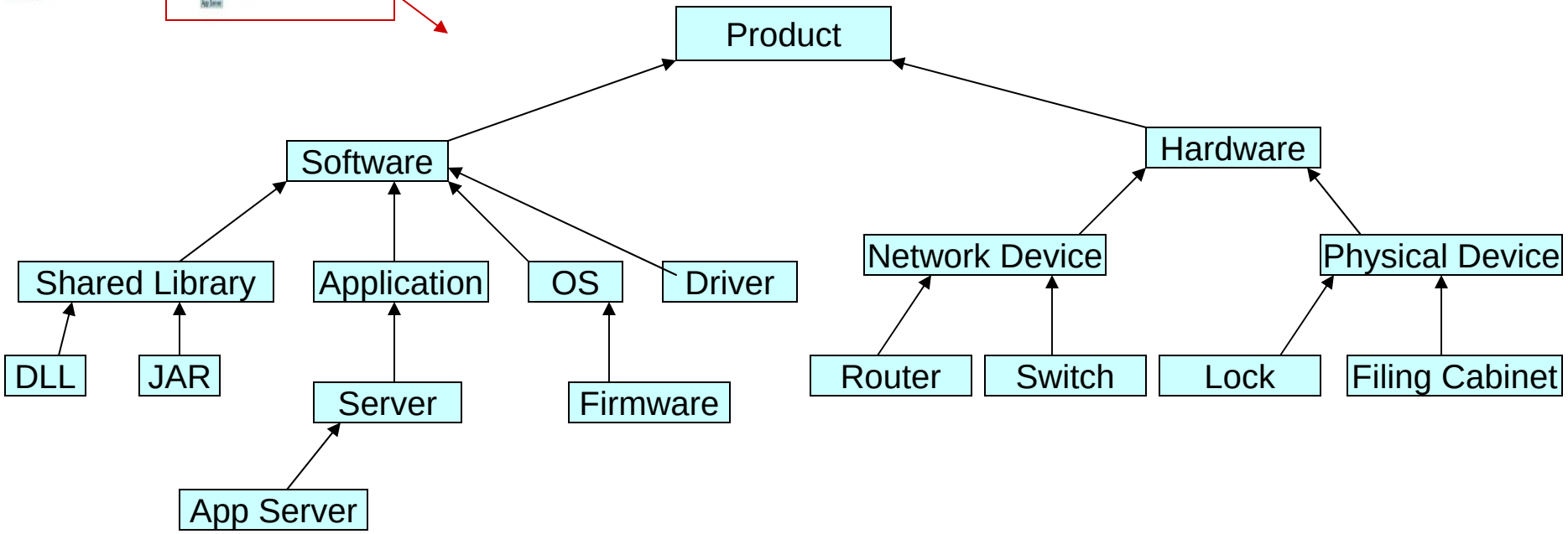
= <owl:Class>

= <rdfs:subClassOf>

ABC = <rdf:Property>



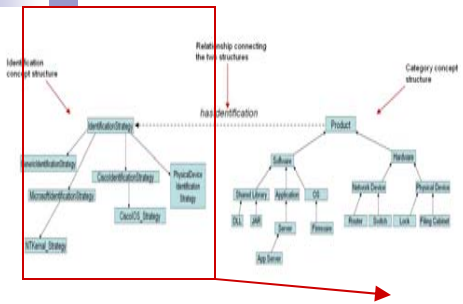
Product Category Hierarchy



Possible Predicates

- *hasIdentification*, domain of Product, range of IdentificationStrategy, <owl:inverseFunctionalProperty>
- *hasReleaseDate*, domain of Product
- *hasCpeName*, domain of Product
- *usesSharedLibrary*, domain of Application, range of SharedLibrary
- *contains*, domain of Product, range of Product, inverseOf *containedIn*
- *hasOwner*, domain of Product, range of Foaf:Agent, inverseOf *ownedBy*
- *hasAutomationTest*, domain of Product
- Many other possibilities exist, very granular predicates can be defined further down tree.

= <owl:Class>
= <rdfs:subClassOf>
ABC = <rdf:Property>



Identification Concept Hierarchy

IdentificationStrategy

Product instance data related to an IdentificationStrategy through the predicate hasIdentification

GenericIdentificationStrategy

CiscoIdentificationStrategy

PhysicalDevice Identification Strategy

MicrosoftIdentificationStrategy

CiscoIOS_Strategy

NTKernal_Strategy

Possible Predicates

- *hasName*, domain of IdentificationStrategy
- *hasModelNumber*, domain of PhysicalDeviceIdentificationStrategy
- *hasCiscoTrainIdentifier*, domain of CiscoIOS_Strategy
- *hasCiscoInterimBuildNumber*, domain of CiscoIOS_Strategy
- *hasMicrosoftMajorVersion*, domain of NTKernal_Strategy
- *hasVersion*, domain of GenericIdentificationStrategy
- *hasUpdate*, domain of GenericIdentificationStrategy

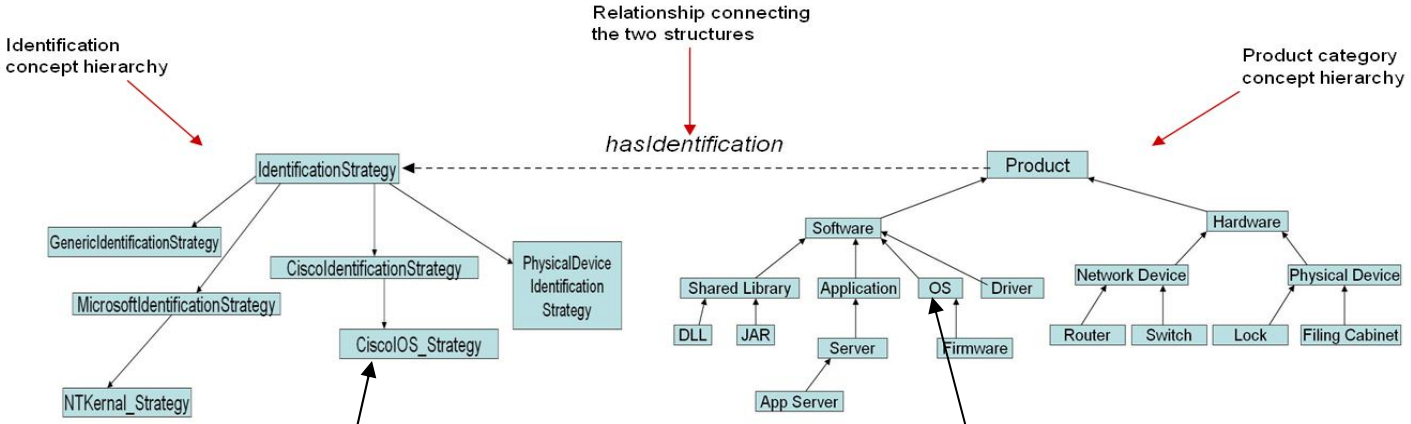
= <owl:Class>

= <rdfs:subClassOf>

ABC = <rdf:Property>

Product Instance Data Instantiated from Model Classes

Model



Instance Data

Blank Node

blank_32320

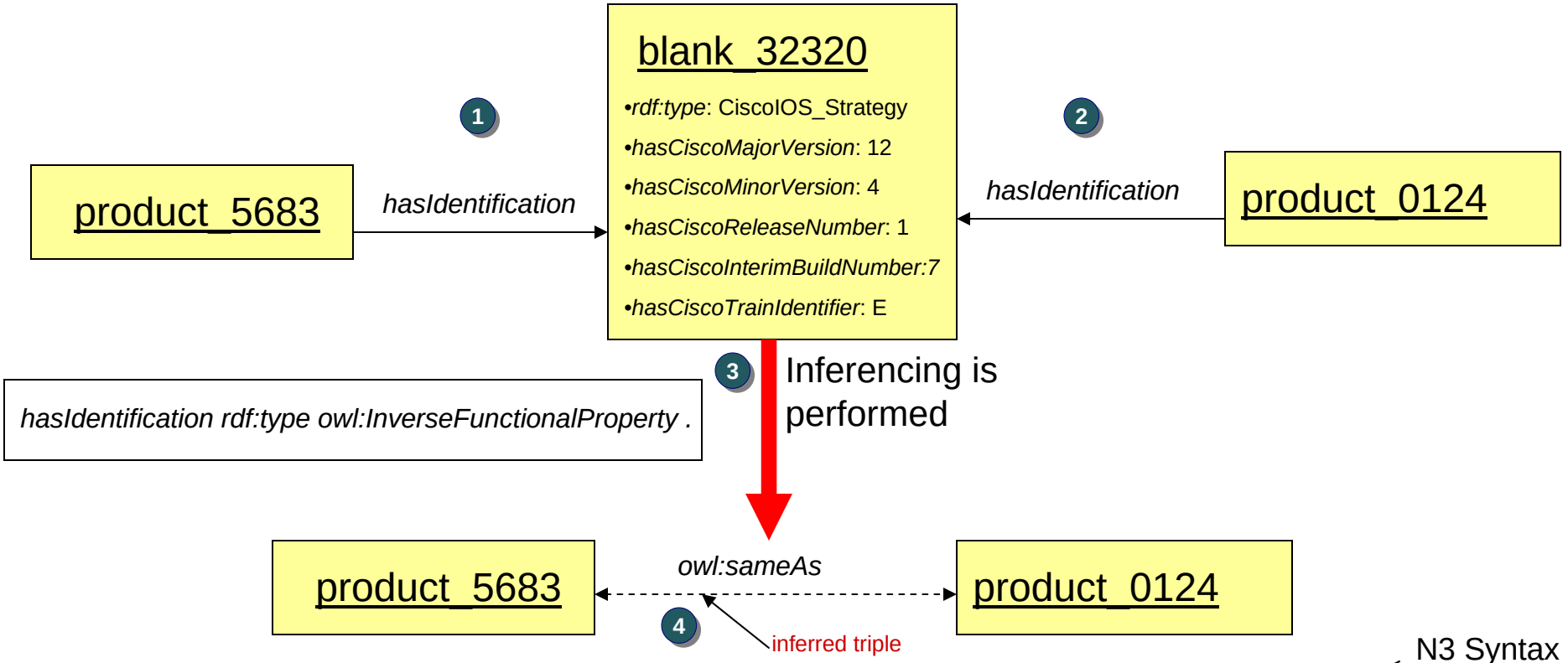
- *rdf:type*: CiscoIOS_Strategy
- *hasCiscoMajorVersion*: 12
- *hasCiscoMinorVersion*: 4
- *hasCiscoReleaseNumber*: 1
- *hasCiscoInterimBuildNumber:7*
- *hasCiscoTrainIdentifier*: E

hasIdentification

product_0124

- *rdf:type*: OS
- *hasCpeName*: cpe:/o:cisco:ios:12.4(1.7)E
- *hasOwner*: Foaf_resource:Cisco

hasIdentification Property Uniquely Identifies a Product



N3 Syntax

Definition of InverseFunctionalProperty:

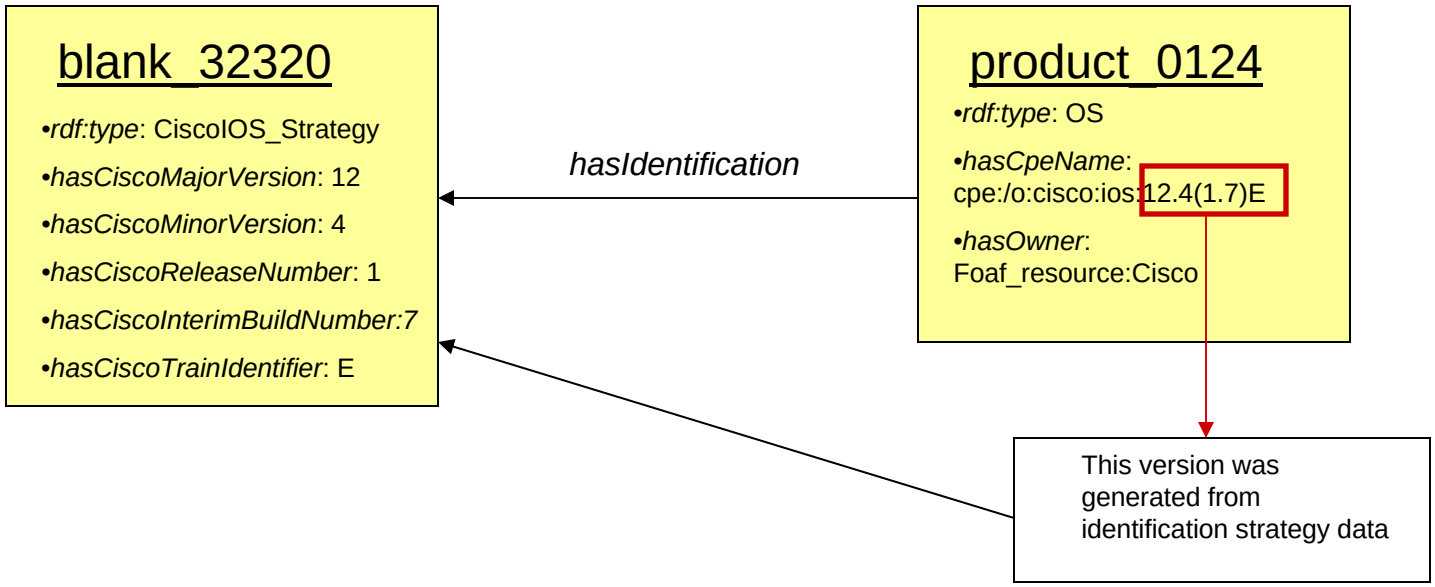
If a property, *P*, is tagged as InverseFunctional then for all *x*, *y*, *z:*
P(y, x) and *P(z, x)* implies *y = z*

OR

P rdf:Type owl:InverseFunctionalProperty .
Y P X .
Z P X.
Infer that:
Y owl:sameAs Z .

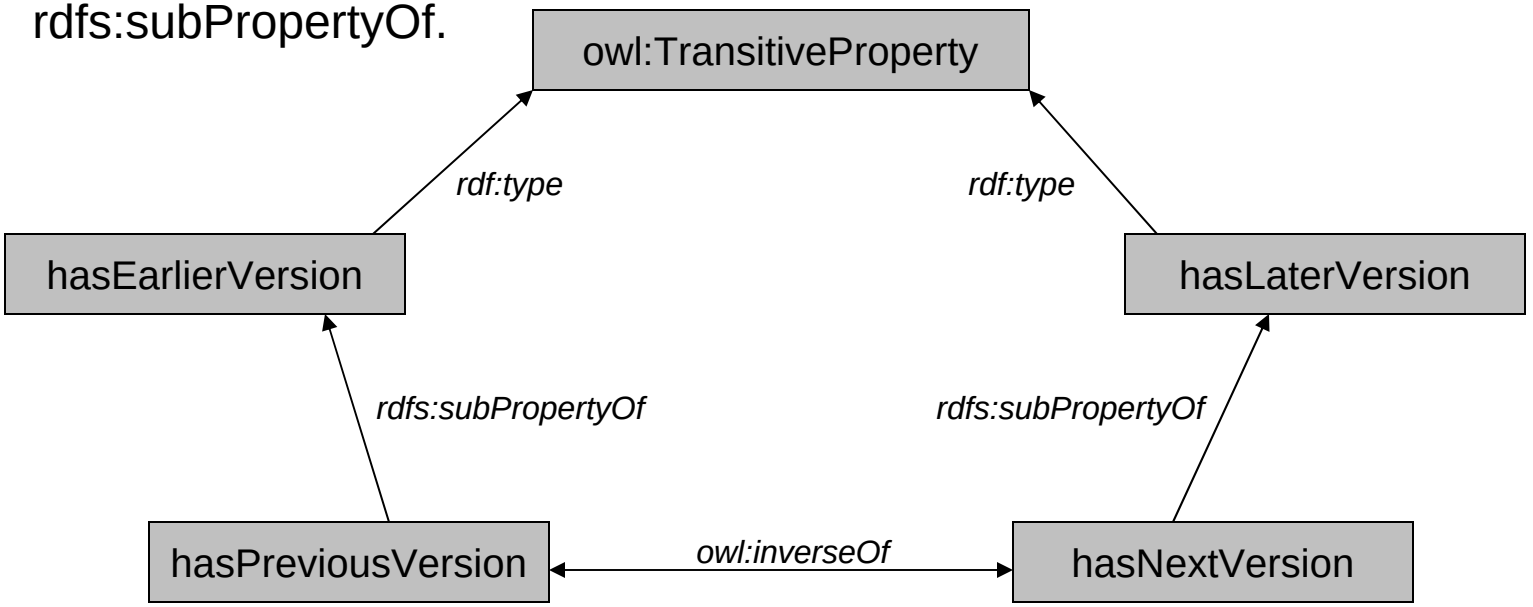
Ontology will provide backwards compatibility with CPE 2.x

- CPE names can be generated from product instance data in a formalized way due to the granular way in which IdentificationStrategies are modeled.
- If modeled with SWRL rules, this backwards compatibility logic will live in the model.

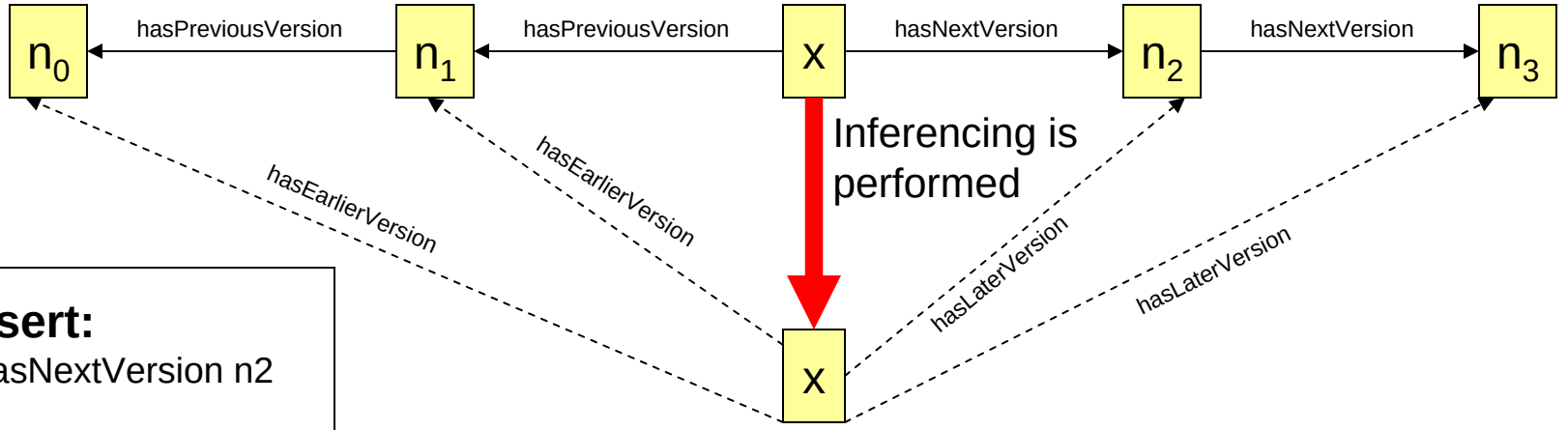


The Ontology Provides the Capability for Modeling Ranges of Products

- This is accomplished with four predicates
  - *hasNextVersion, hasPreviousVersion*
  - *hasLaterVersion (transitive), hasEarlierVersion* (transitive)
- These four predicates are modeled using a predicate hierarchy such that the non-transitive predicates are related to the transitive predicates through rdfs:subPropertyOf.



Inferencing for Product Range Data



Assert:
X hasNextVersion n2

Infer:

X hasLaterVersion n_2

n_2 hasPreviousVersion X

n_2 hasEarlierVersion X

= product instance data
= asserted triple
= inferred triple

- The reasoner creates inferred triples which allow an observer to see all products in a version chain earlier and later than x. Inferred triples are also captured for n_0, n_1, n_2, and n_3.
- The version chain DOES have to be captured by a human since a version chain order is ambiguous
  - In the future if IdentificationStrategies are modeled fully it may be possible to encode version chain order into the model and let the reasoner figure it out.

Querying for Product Range Data

- Analysts populate version chain using non-transitive predicates (hasNextVersion and hasPerviousVersion)
- A SPARQL query could then be written against the transitive predicates which the reasoner has inferred.
- Querying against the transitive predicates allow system to determine all “earlier” and all “later” versions (i.e. a product range).

```
SELECT ?product
WHERE {
        ?product a nvd:product
        ?product nvd:hasEarlierVersion 3.2
        ?product nvd:hasLaterVersion 5.4
}
```

- Keeps all application logic for range relationships in model
- This DOES require instance data to be fully populated
- Could potentially explode triples

Contact Information

Paul Cichonski

paul.cichonski@nist.gov

http://nvd.nist.gov

Extra

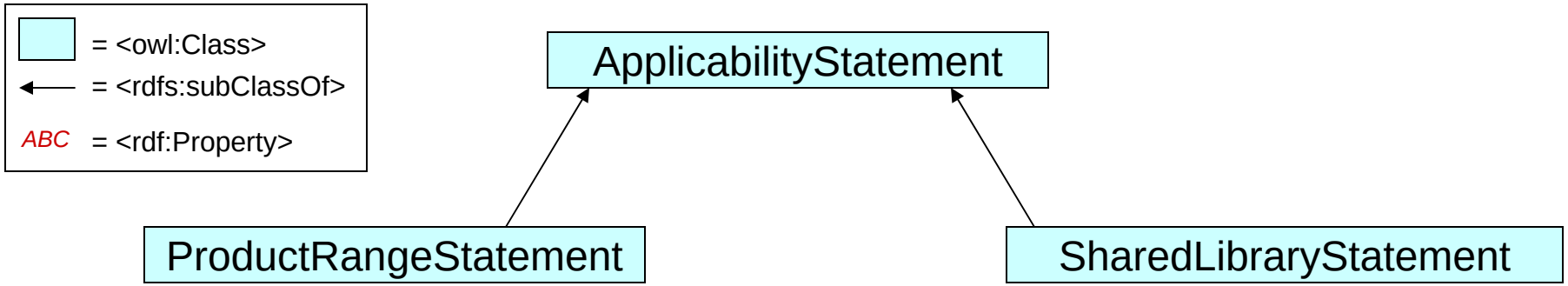
Ontology Provides the Means to Make More Granular Statements of Applicability

- Shared Library (e.g. DLL, JAR) instance data can be captured and related to typical product instance data.
  - Through predicates such as *usesSharedLibary*.
  - Analysts can then associate vulnerabilities with shared libraries and simple queries can be used to determine all products which use the shared library.
- Classes can be added to the Model to capture codebases.
  - Relationships can then be asserted on instance data to relate products to the codebase from which they originate (e.g. *isBasedOnCodeFrom*).
  - Analysts can assign vulnerability to a specific codebase, and the system can generate the list of all applicable products.
- These predicates could become the standard way for all product ontologies to declare these relationships.
  - This would provide a shared understanding across a wide set of data.

Statements of Applicability can be Modeled as First Class Individuals

- Applicability statements are a way of relating a grouping of products to a particular IT concept (e.g. CVE, CCE, Checklist).
- If modeled as actual classes in an ontology applicability statements will provide the ability to create groupings of products at various levels of abstraction depending on the needs of a use case.
  - Possible to represent all products in a certain range
  - Possible to represent all products that use a certain shared library
- Predicates can be defined to capture relationships between applicability statements
  - Possible to express that one applicability statement is a prerequisite for another statement to be possible on a network
  - Possible to express that one applicability statement subsumes another statement.

Applicability Statements Modeled as First Class Individuals

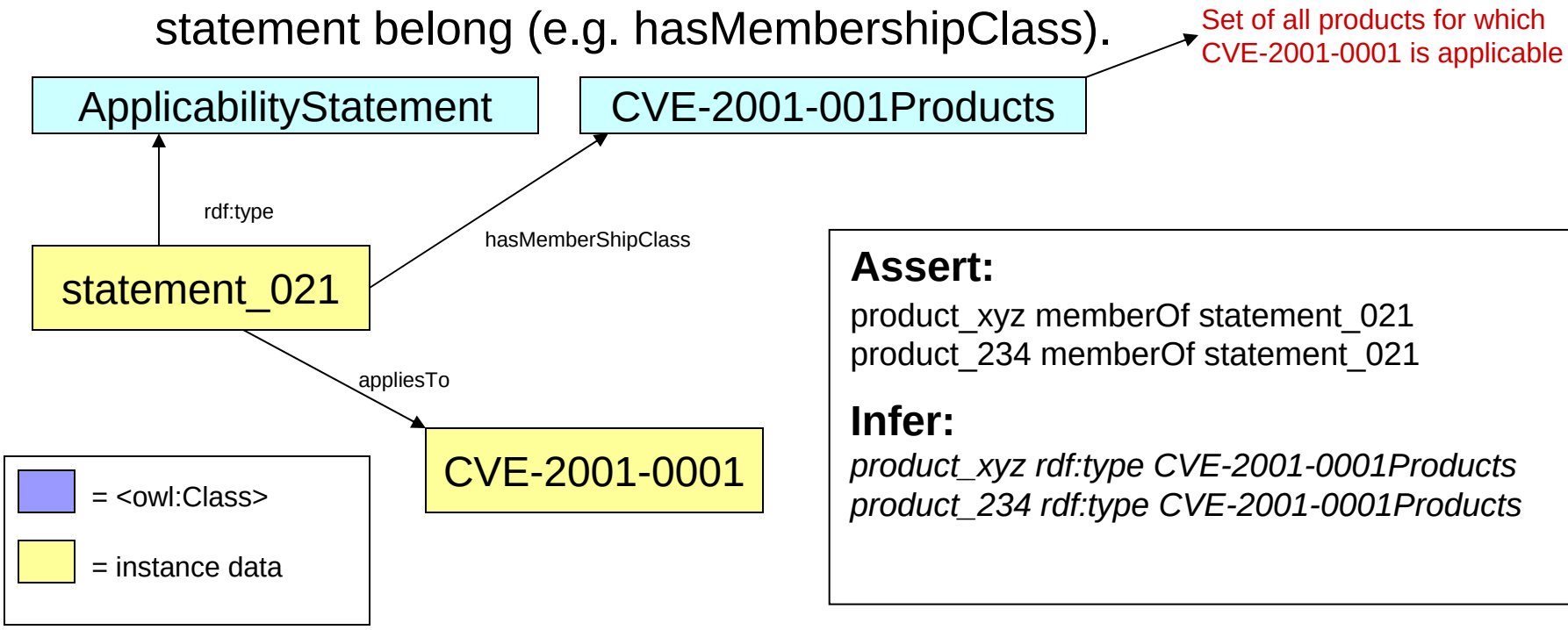


Possible Predicates

- Predicates capturing information relating to products encompassed by a applicability statement
  - *includesProduct, domain of ApplicabilityStatement, range of Product, inverseOf memberOf*
  - *memberOf, domain of Product, range of ApplicabilityStatement, inverseOf includesProduct*
  - *minimumProduct*, domain of ProductRangeStatement, range of Product
  - *maximumProduct*, domain of ProductRangeStatement, range of Product
  - *sharedLibrary*, domain of SharedLibraryStatement, range of SharedLibrary
- Predicates capturing relationships between applicability statements and IT concepts *(e.g. CVE, CCE, Checklist)*
  - *hasApplicabilityStatement, domain of some IT concept, range of ApplicabilityStatement, inverseOf appliesTo*
  - *appliesTo, domain of ApplicabilityStatement, range of some IT concept, inverseOf hasApplicabilityStatement*
- Predicates capturing relationships between applicability statements and other applicability statements
  - *hasPrerequisite, domain of ApplicabilityStatement, range of ApplicabilityStatement, inverseOf prerequisiteFor*
  - *prerequisiteFor, domain of ApplicabilityStatement, range of ApplicabilityStatement, inverseOf hasPrerequisite*
  - *subsumes, domain of ApplicabilityStatement, range of ApplicabilityStatement, inverseOf subsumedBy*

Defining Product Class Membership through Applicability Statements

- Model may want to include a class defining the set of all products for which a certain CVE is applicable.
- This can be done by defining a relationship between an applicability statement and the class to which all products included in the statement belong (e.g. hasMembershipClass).



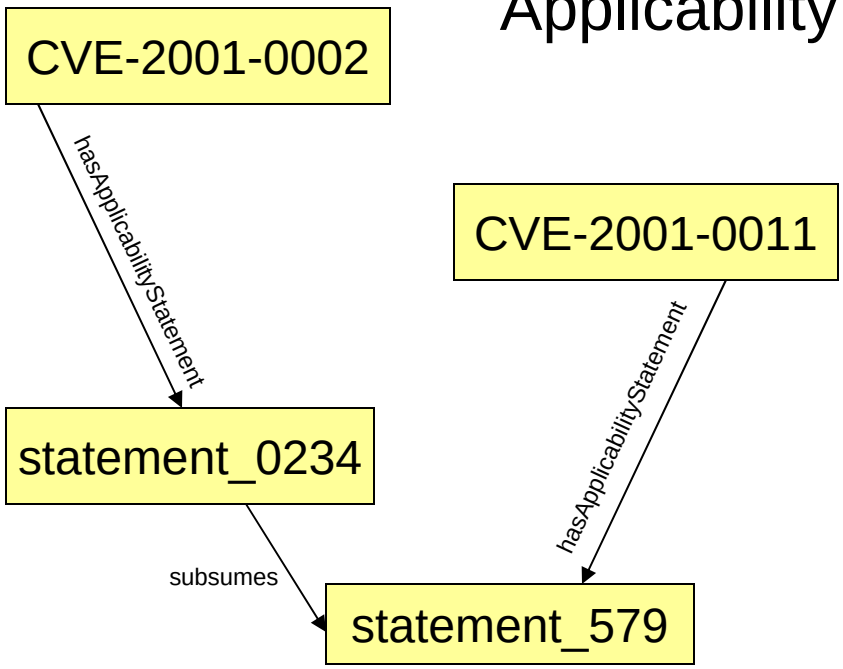
Assert:

product_xyz memberOf statement_021
product_234 memberOf statement_021

Infer:

product_xyz rdf:type CVE-2001-0001Products
product_234 rdf:type CVE-2001-0001Products

Possible to Capture Relationships Where Statements of Applicability Subsume Others



Set of all applicability statements that match conditions on a Organization A's network

orgA:FoundOnNetwork

orgA:FoundOnNetwork owl:subClassOf
[a owl:Restriction;
owl:onProperty subsumes;
owl:allValuesFrom orgA:FoundOnNetwork].

orgA:ExistingVulnerability

orgA:ExistingVulnerability owl:equivalentClass
[a owl:Restriction;
owl:onProperty hasApplicabilityStatement;
owl:someValuesFrom orgA:FoundOnNetwork].

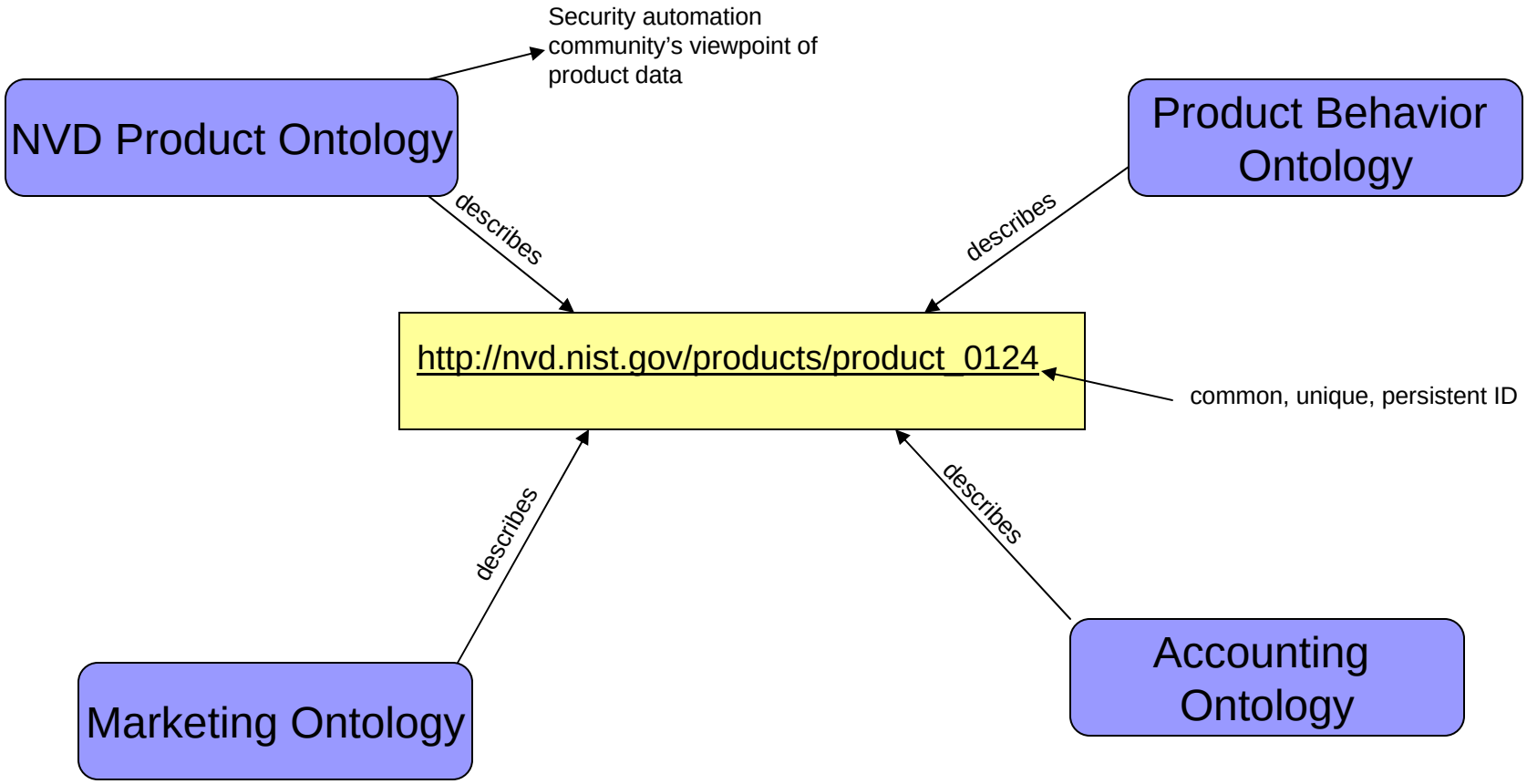
Assert:

statement_0234 rdf:type orgA:FoundOnNetwork

Infer:

statement_579 rdf:type orgA:FoundOnNetwork
CVE-2001-0002 rdf:type orgA:ExistingVulnerability
CVE-2001-0011 rdf:type orgA:ExistingVulnerability

The Same Product Resource can be Described by Heterogeneous Viewpoints using disparate Ontologies



A Formalized Model will Allow for a Shared Understanding of How To Capture Normalized Product Data

- The IdentificationStrategy hierarchy provides a method to define vendor's versioning strategies.
  - The granularity of the model is up to the community.
  - The model itself will show users the types of relationships that must be captured to identify a product.
- In the future it may even be possible to create a complementary ontology which tells user's HOW to find the data
  - Ex) where to look, commands, API calls.
  - This will really allow us to put most of the logic in the ontology itself and provide a high level of confidence for users creating product instance data.

Inferencing for Broad Statements of Applicability

- Possible to define classes to identify all individuals which meet a desired criteria.
- For example, a class could be defined to capture all CiscoIOS Products

```
<owl:Class rdf:ID="OperatingSystem_1">
  <rdfs:subClassOf rdf:resource="#OperatingSystem"/>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
   >CiscoIOSProduct</rdfs:label>
  <owl:equivalentClass>
   <owl:Restriction>
    <owl:onProperty rdf:resource="#hasIdentification"/>
    <owl:someValuesFrom rdf:resource="#CiscoIOS_Strategy"/>
   </owl:Restriction>
  </owl:equivalentClass>
 </owl:Class>
```