Semantic CMS Community

PhD Course

Linköping, 2012

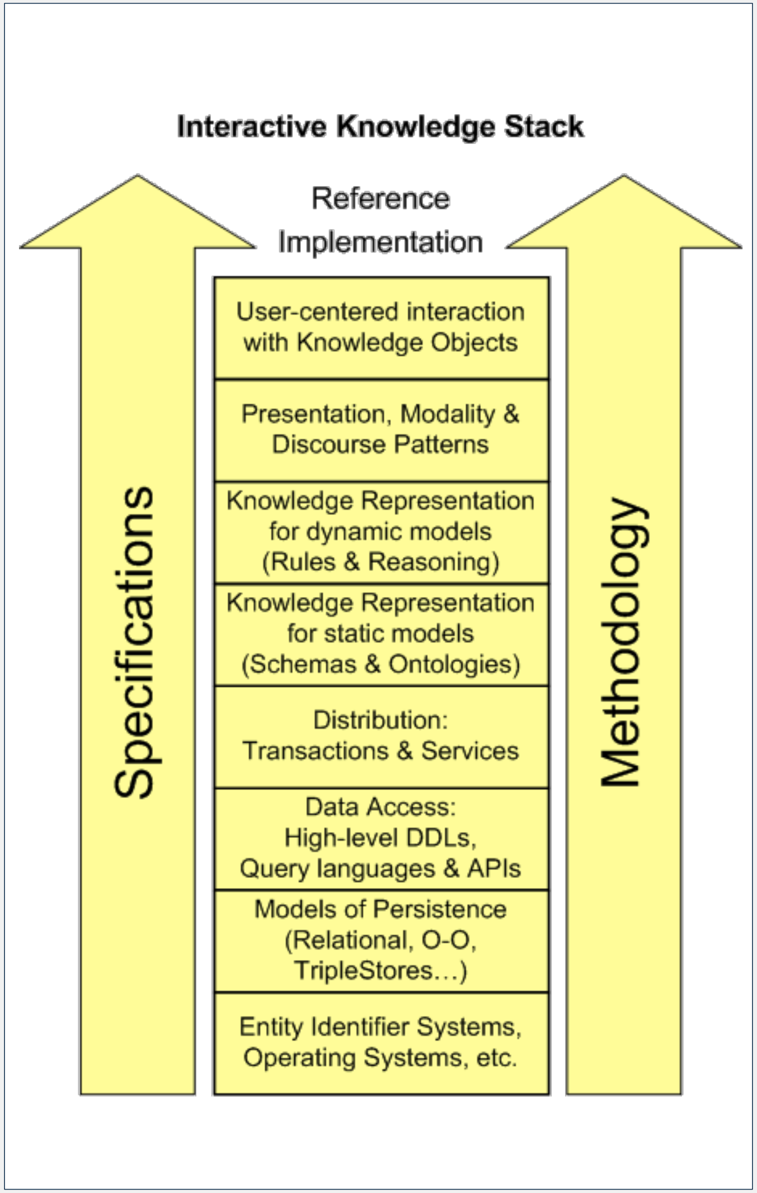
Semantic CMS: Introduction and Overview of IKS

October 31st, 2012

Valentina Presutti
STLab, ISTC-CNR (Italy)

valentina.presutti@cnr.it

Co-funded by the
European Union



Interactive Knowledge Stack

Reference Implementation

User-centered interaction with Knowledge Objects

Presentation, Modality & Discourse Patterns

Knowledge Representation for dynamic models (Rules & Reasoning)

Knowledge Representation for static models (Schemas & Ontologies)

Distribution: Transactions & Services

Data Access: High-level DDLs, Query languages & APIs

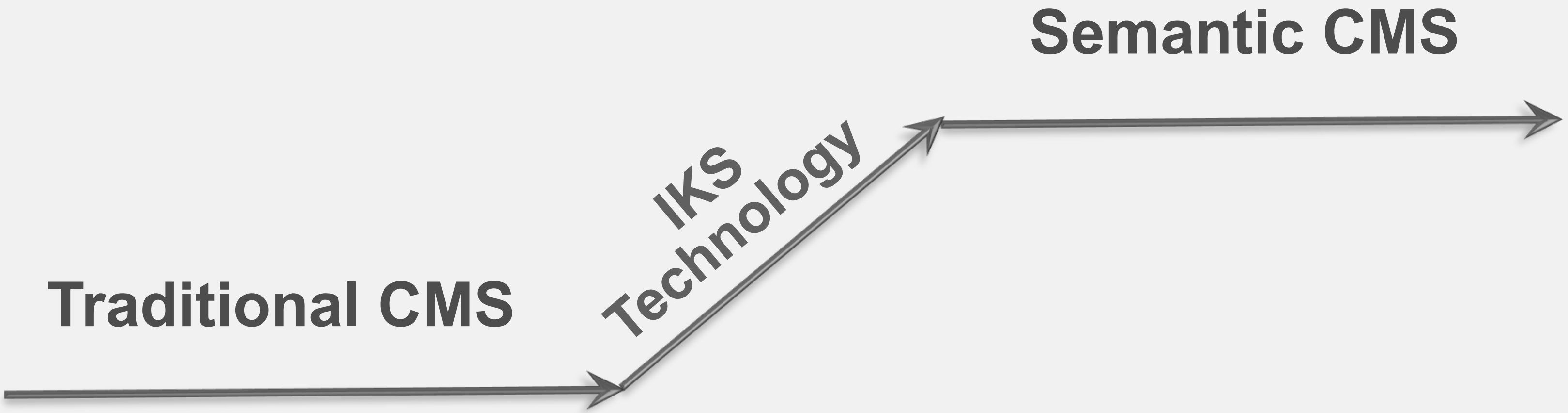
Models of Persistence (Relational, O-O, TripleStores…)

Entity Identifier Systems, Operating Systems, etc.

IKS Goal

A Reference Architecture for Semantically Enabled Content Management Systems

IKS Technology – a Path to the Semantic Level



Semantic CMS

IKS Technology

Traditional CMS

What is a Semantic CMS?

Traditional CMS vs. Semantic CMS

Traditional CMS

- Atomic unit: Document
- Properties as meta-data
  - e.g. author
  - tags, keywords
- Keyword search for
  - strings in docs
- Document Management
  - Document types
  - Document workflow

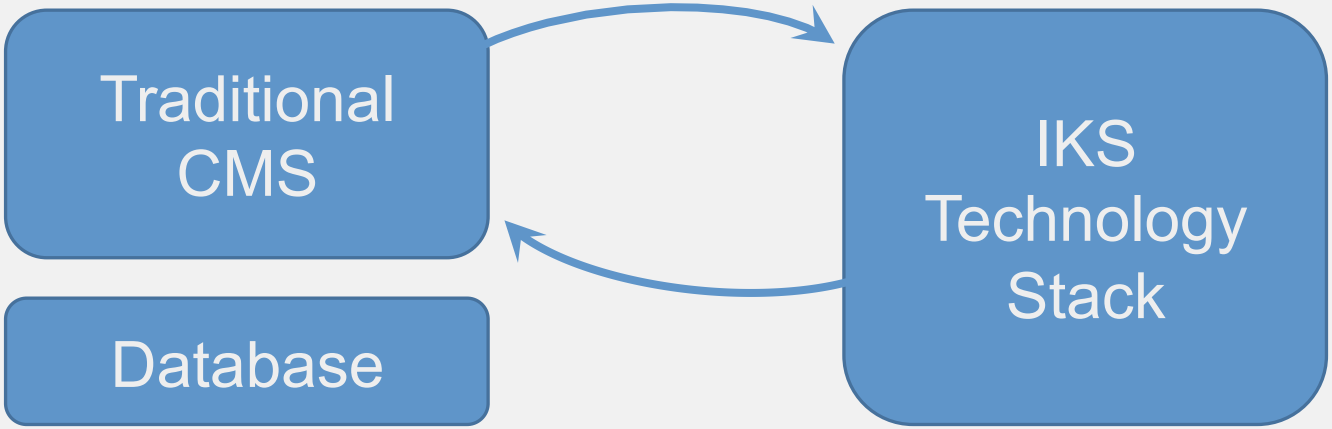
Semantic CMS

- Atomic unit: Entity
- Semantic meta-data
  - Defined entity types
  - Linked entities
- Semantic search for
  - entities and their relations
- Knowledge Management
  - Entity management
  - Ontologies

Do Not Replace – but Extend

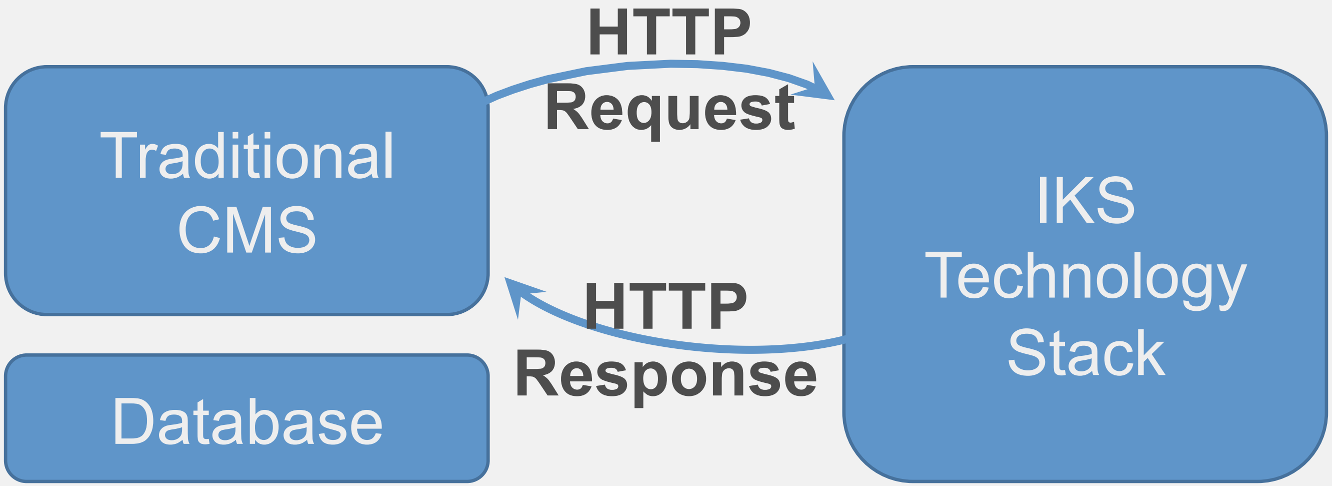
- No need to replace your existing technology.
- IKS components offer service oriented integration.

Extend by Using Semantic Services



Rely on the Concepts of the Web

- Integration through a RESTful web service API
- Resources are identified by their URI



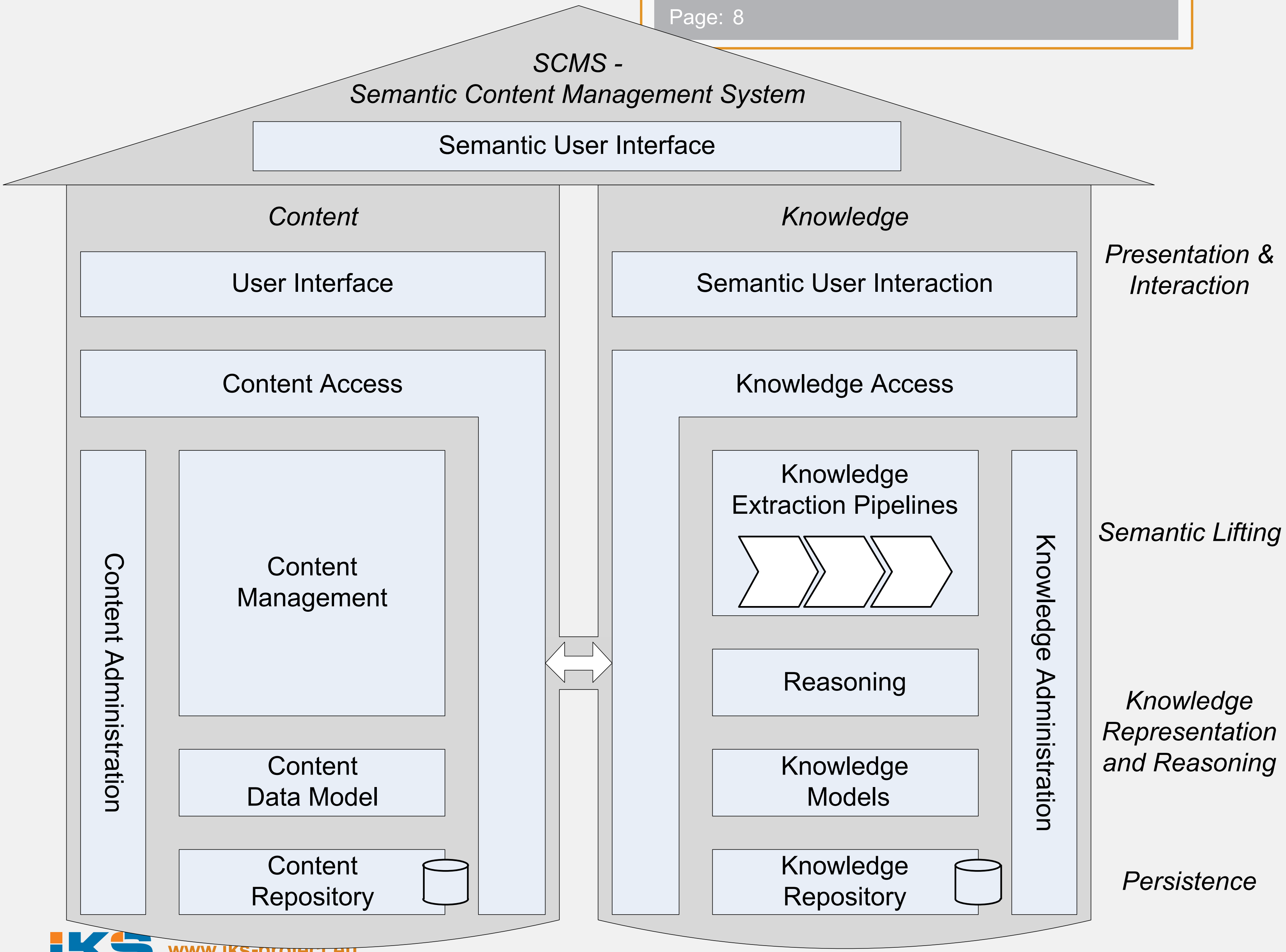
Hands on IKS 7.0

- You need: Java Runtime Environment (JRE) V1.6
- You download: http://dev.iks-project.eu/downloads/iks-stack-releases/IKS-RI-7.0.zip
- You execute:

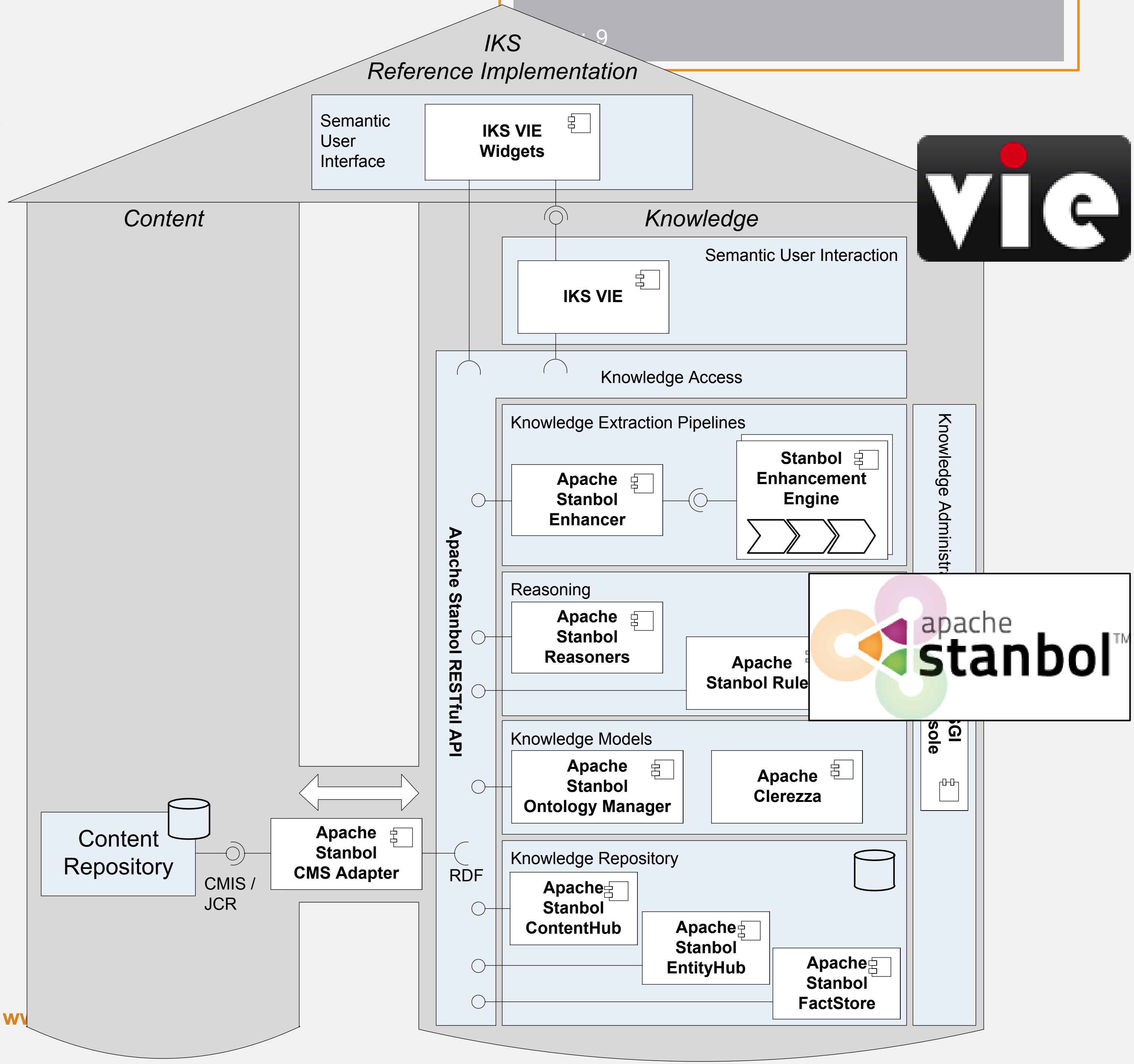
```
java -Xmx1024m –jar iks-7.0-launcher.jar
```

and open

`http://localhost:8080`


SCMS -
Semantic Content Management System
Semantic User Interface
Content
Knowledge
User Interface
Semantic User Interaction
Presentation & Interaction
Content Access
Knowledge Access
Content Administration
Content Management
Knowledge Extraction Pipelines
Semantic Lifting
Reasoning
Knowledge Administration
Content Data Model
Knowledge Models
Knowledge Representation and Reasoning
Content Repository
Knowledge Repository
Persistence

IKS 7.0


IKS
Reference Implementation
Semantic User Interface
IKS VIE Widgets
Content
Knowledge
Semantic User Interaction
IKS VIE
Knowledge Access
Apache Stanbol RESTful API
Knowledge Extraction Pipelines
Apache Stanbol Enhancer
Stanbol Enhancement Engine
Reasoning
Apache Stanbol Reasoners
Apache Stanbol Rule
Knowledge Models
Apache Stanbol Ontology Manager
Apache Clerezza
Knowledge Repository
Apache Stanbol ContentHub
Apache Stanbol EntityHub
Apache Stanbol FactStore
Knowledge Administra
GI
sole
Content Repository
CMIS / JCR
Apache Stanbol CMS Adapter
RDF
vie
apache stanbol™

VIE Quick Facts

- **VIE** is a utility library for semantic maintenance in JavaScript
- Offers semantic web developers a DSL to ease recurring tasks
  - Easy access to embedded semantic annotations in HTML (RDFa)
  - Easy loading of properties for entities from external services
  - Easy saving of knowledge about entities
  - Easy querying of semantic services
- **VIE Widgets** are web user interface components based on VIE.

Apache Stanbol Quick Facts

- Modular (OSGi) components implemented in Java

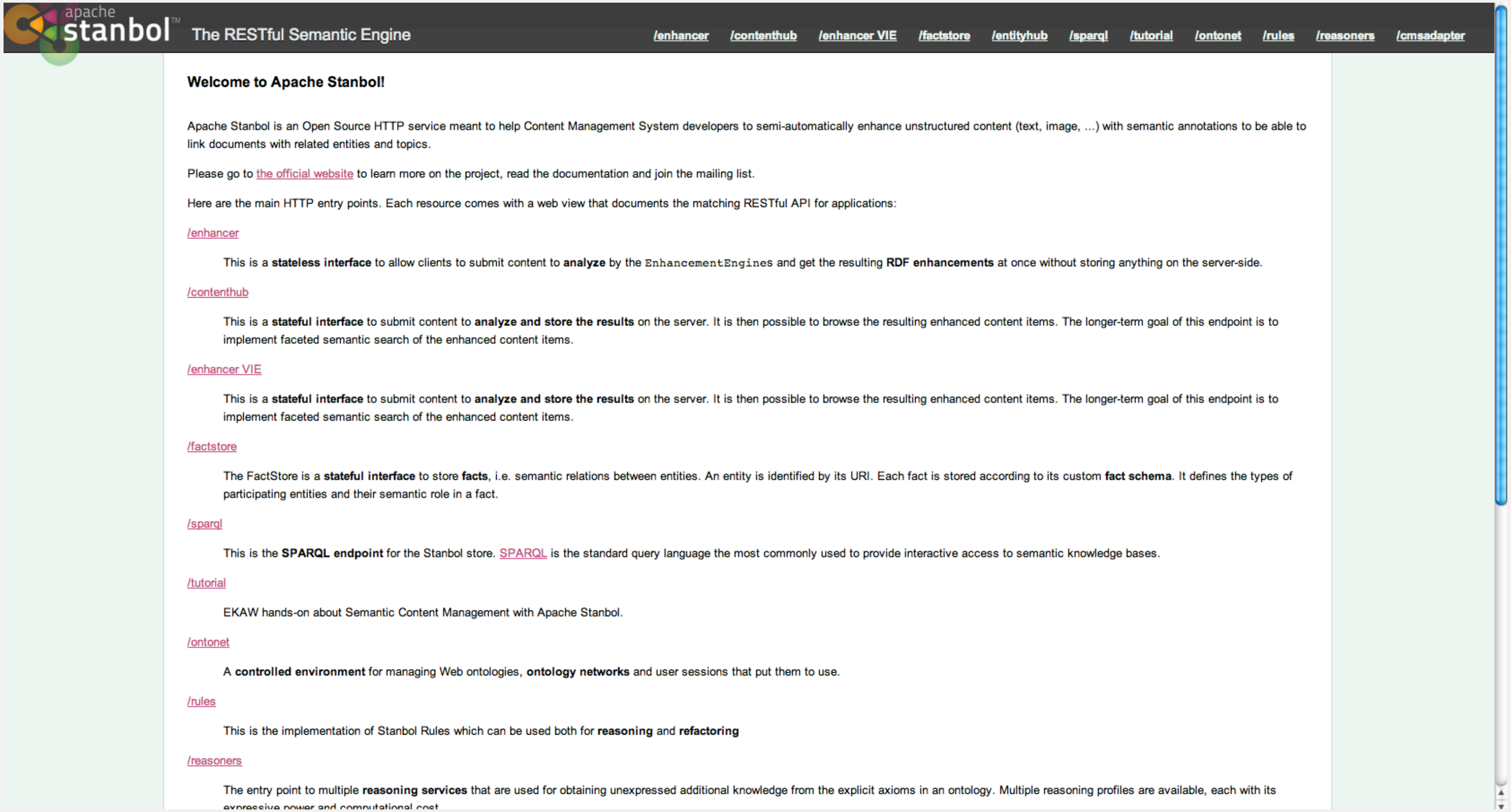
Semantic Lifting

- Enhance content
- Link to Linked Open Data (LOD) sources
- Store and index enhanced content for search

Knowledge Representation & Reasoning

- Manage ontologies
- Apply rules to ontologies
- Reasoning over managed ontologies

Stanbol GUI



Welcome to Apache Stanbol!

Apache Stanbol is an Open Source HTTP service meant to help Content Management System developers to semi-automatically enhance unstructured content (text, image, ...) with semantic annotations to be able to link documents with related entities and topics.

Please go to the official website to learn more on the project, read the documentation and join the mailing list.

Here are the main HTTP entry points. Each resource comes with a web view that documents the matching RESTful API for applications:

/enhancer

This is a **stateless interface** to allow clients to submit content to **analyze** by the `EnhancementEngines` and get the resulting **RDF enhancements** at once without storing anything on the server-side.

/contenthub

This is a **stateful interface** to submit content to **analyze and store the results** on the server. It is then possible to browse the resulting enhanced content items. The longer-term goal of this endpoint is to implement faceted semantic search of the enhanced content items.

/enhancer VIE

This is a **stateful interface** to submit content to **analyze and store the results** on the server. It is then possible to browse the resulting enhanced content items. The longer-term goal of this endpoint is to implement faceted semantic search of the enhanced content items.

/factstore

The FactStore is a **stateful interface** to store **facts**, i.e. semantic relations between entities. An entity is identified by its URI. Each fact is stored according to its custom **fact schema**. It defines the types of participating entities and their semantic role in a fact.

/sparql

This is the **SPARQL endpoint** for the Stanbol store. SPARQL is the standard query language the most commonly used to provide interactive access to semantic knowledge bases.

/tutorial

EKAW hands-on about Semantic Content Management with Apache Stanbol.

/ontonet

A **controlled environment** for managing Web ontologies, **ontology networks** and user sessions that put them to use.

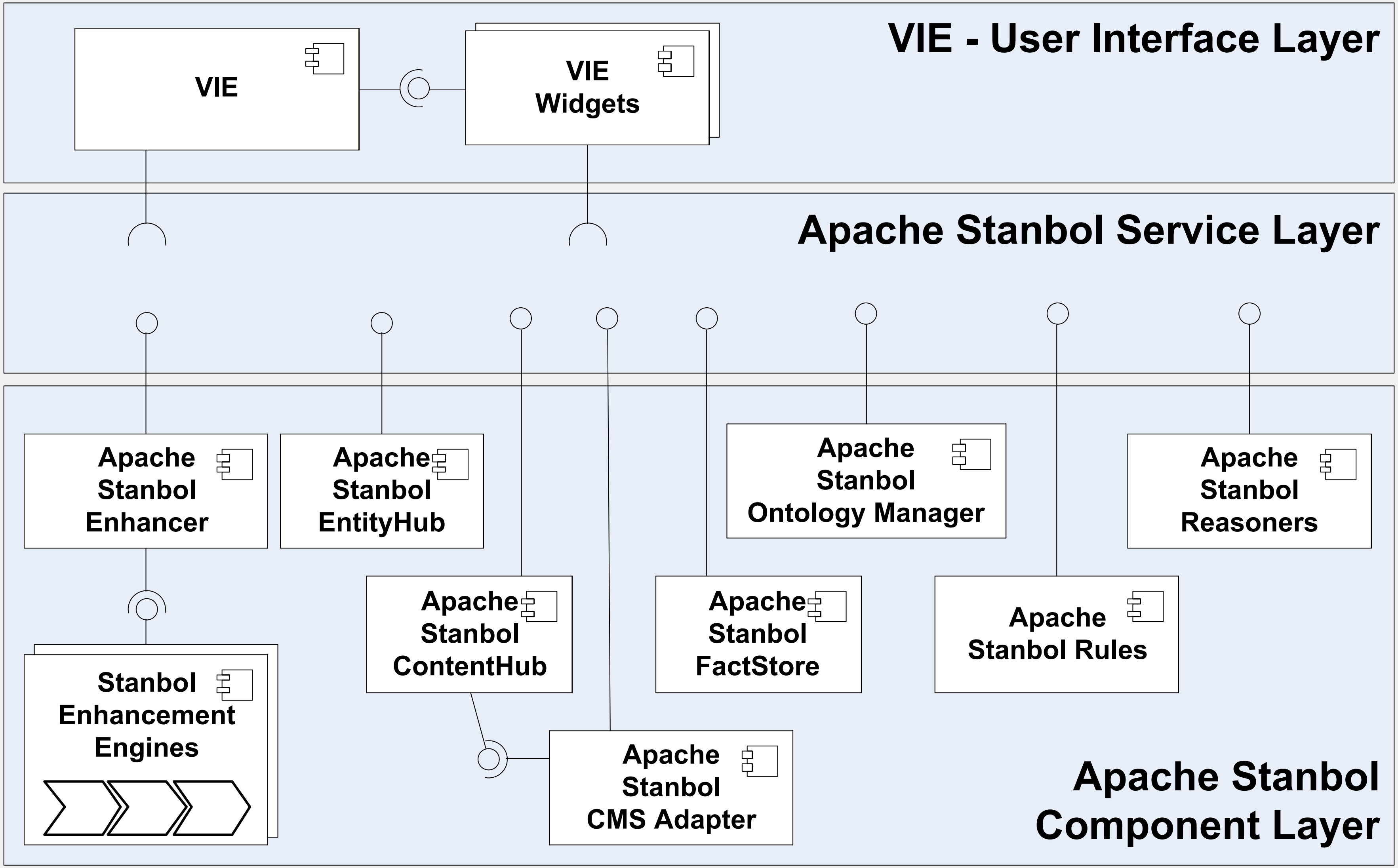
/rules

This is the implementation of Stanbol Rules which can be used both for **reasoning** and **refactoring**

/reasoners

The entry point to multiple **reasoning services** that are used for obtaining unexpressed additional knowledge from the explicit axioms in an ontology. Multiple reasoning profiles are available, each with its expressive power and computational cost.

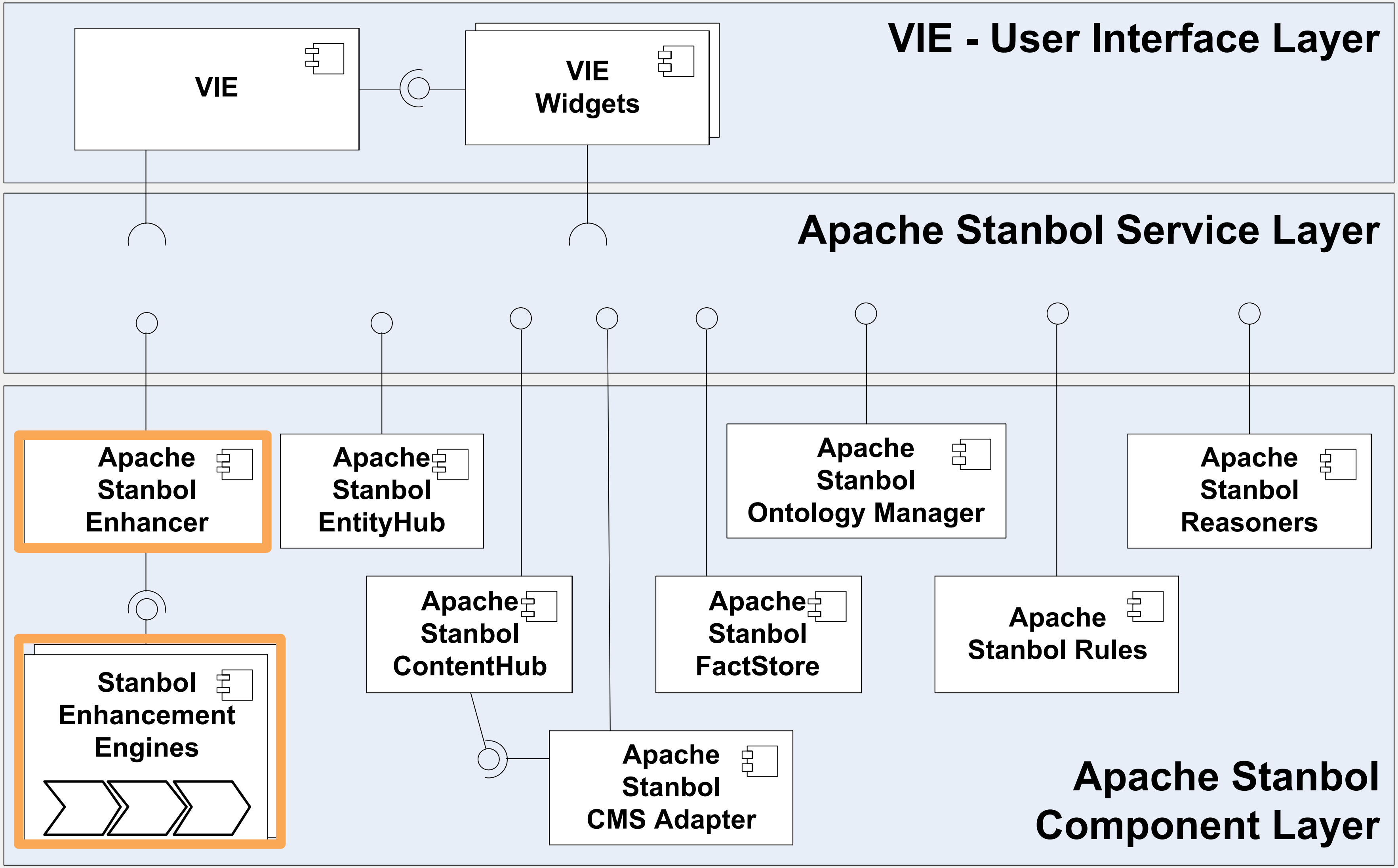
Service-Oriented View


VIE - User Interface Layer
VIE
VIE Widgets
Apache Stanbol Service Layer
Apache Stanbol Enhancer
Apache Stanbol EntityHub
Apache Stanbol Ontology Manager
Apache Stanbol Reasoners
Stanbol Enhancement Engines
Apache Stanbol ContentHub
Apache Stanbol FactStore
Apache Stanbol Rules
Apache Stanbol CMS Adapter
Apache Stanbol Component Layer

Semantic Lifting

Knowledge Representation & Reasoning

Enhancer & Enhancement Engines


VIE - User Interface Layer
VIE
VIE Widgets
Apache Stanbol Service Layer
Apache Stanbol Enhancer
Apache Stanbol EntityHub
Apache Stanbol Ontology Manager
Apache Stanbol Reasoners
Stanbol Enhancement Engines
Apache Stanbol ContentHub
Apache Stanbol FactStore
Apache Stanbol Rules
Apache Stanbol CMS Adapter
Apache Stanbol Component Layer

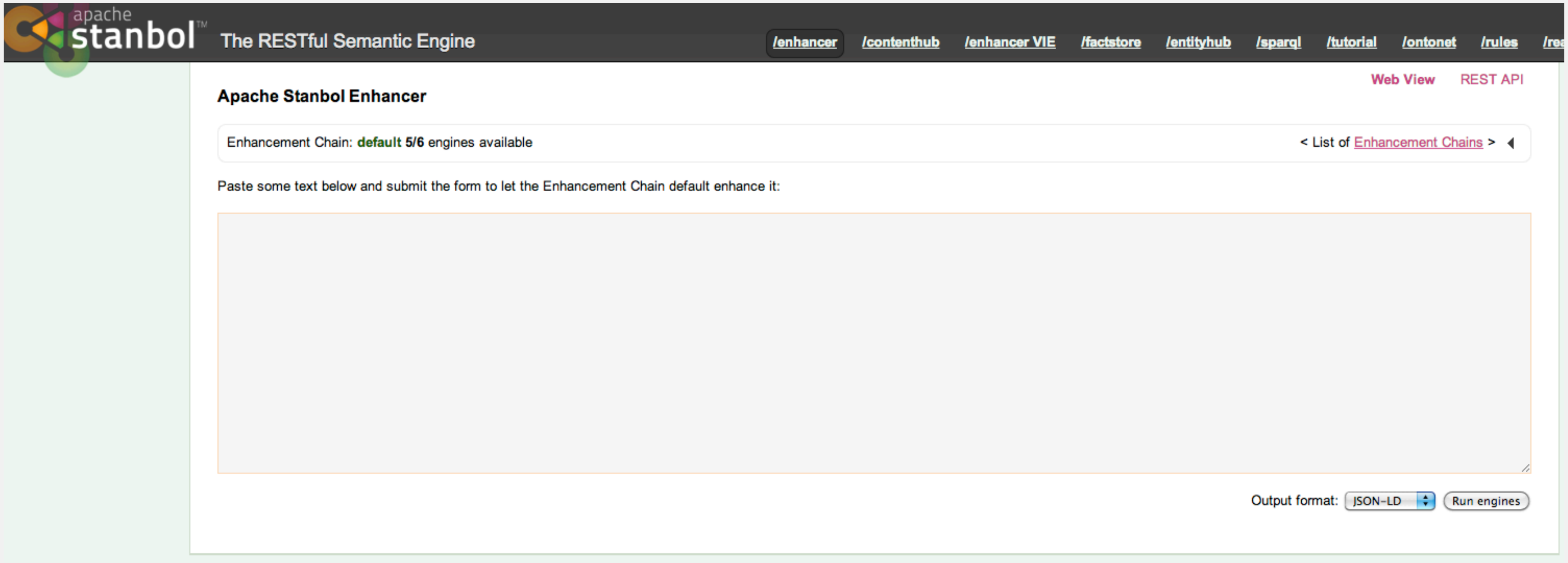
Semantic Lifting

Knowledge Representation & Reasoning

Enhancer & Engines Features

- Semantic lifting by automatically extracting entities from textual content
- Different enhancement engines for specific tasks
- Engines are arranged in customizable enhancement chains where one engine may rely on the output of another engine
- Examples
  - Language Identification Engine
  - Named Entity Extraction Engine
  - Geonames Engine to annotate places with additional information from geonames.org

Stanbol Enhancer GUI



Apache Stanbol Enhancer

Enhancement Chain: **default** **5/6** engines available

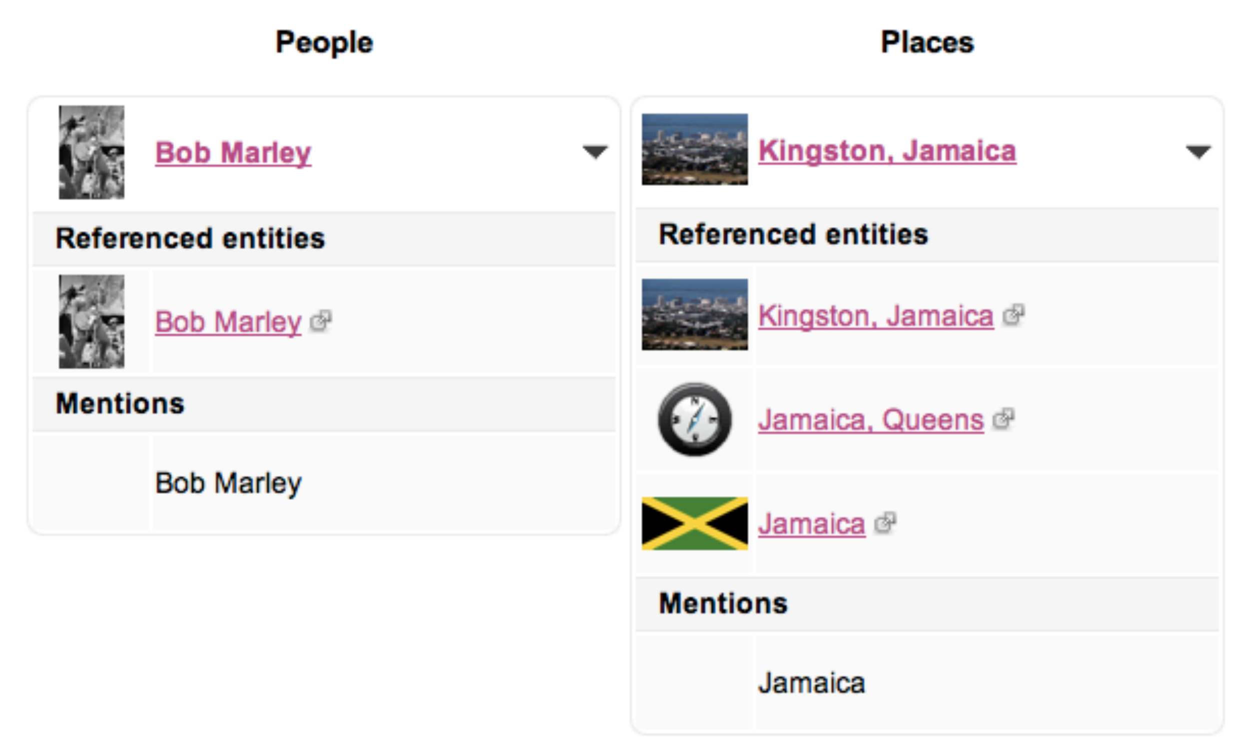
< List of Enhancement Chains >

Paste some text below and submit the form to let the Enhancement Chain default enhance it:

Output format: JSON-LD

Stanbol Enhancer

- Input
  - "Bob Marley was a famous musician from Jamaica."
- Output



People

Bob Marley

Referenced entities

Bob Marley

Mentions

Bob Marley

Places

Kingston, Jamaica

Referenced entities

Kingston, Jamaica

Jamaica, Queens

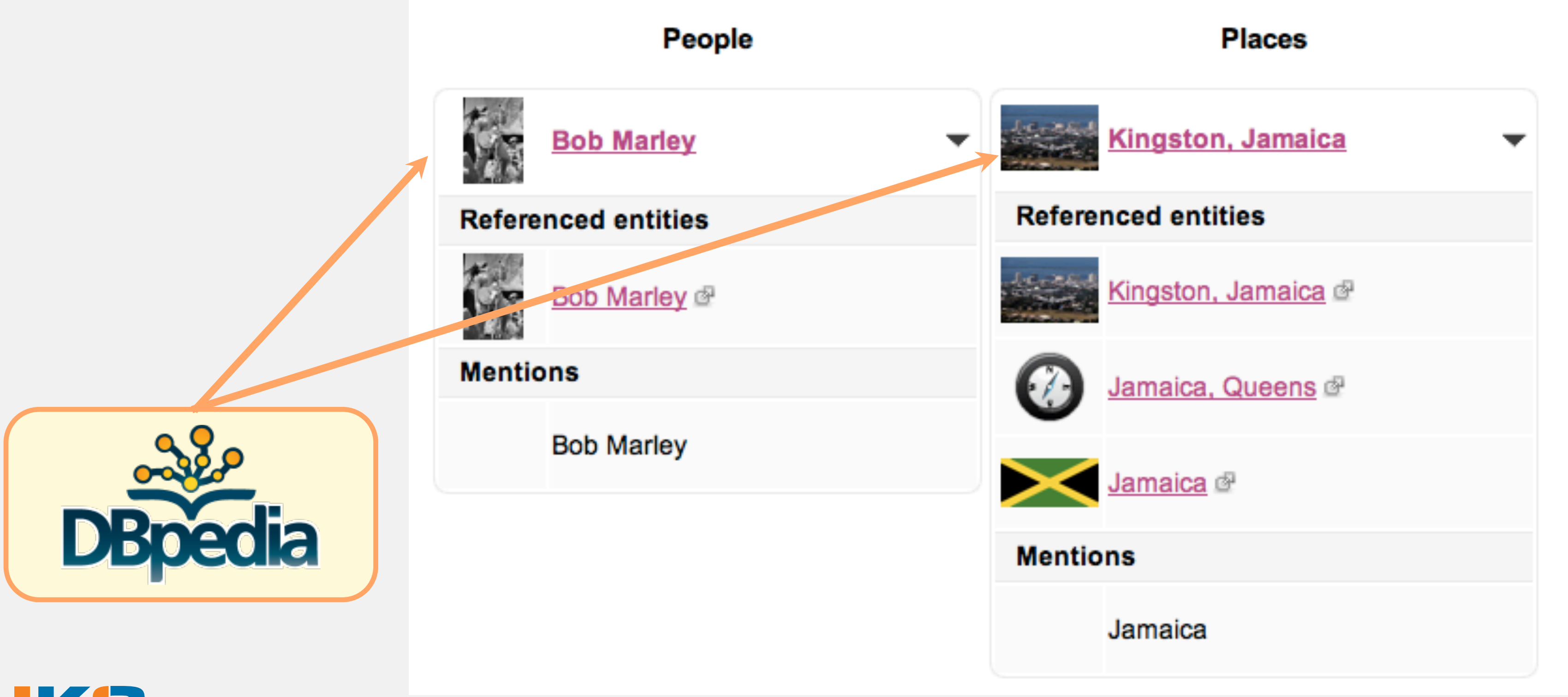
Jamaica

Mentions

Jamaica

Stanbol Enhancer

- Input
  - "Bob Marley was a famous musician from Jamaica."
- Output



People

Bob Marley

Referenced entities

Bob Marley

Mentions

Bob Marley

Places

Kingston, Jamaica

Referenced entities

Kingston, Jamaica

Jamaica, Queens

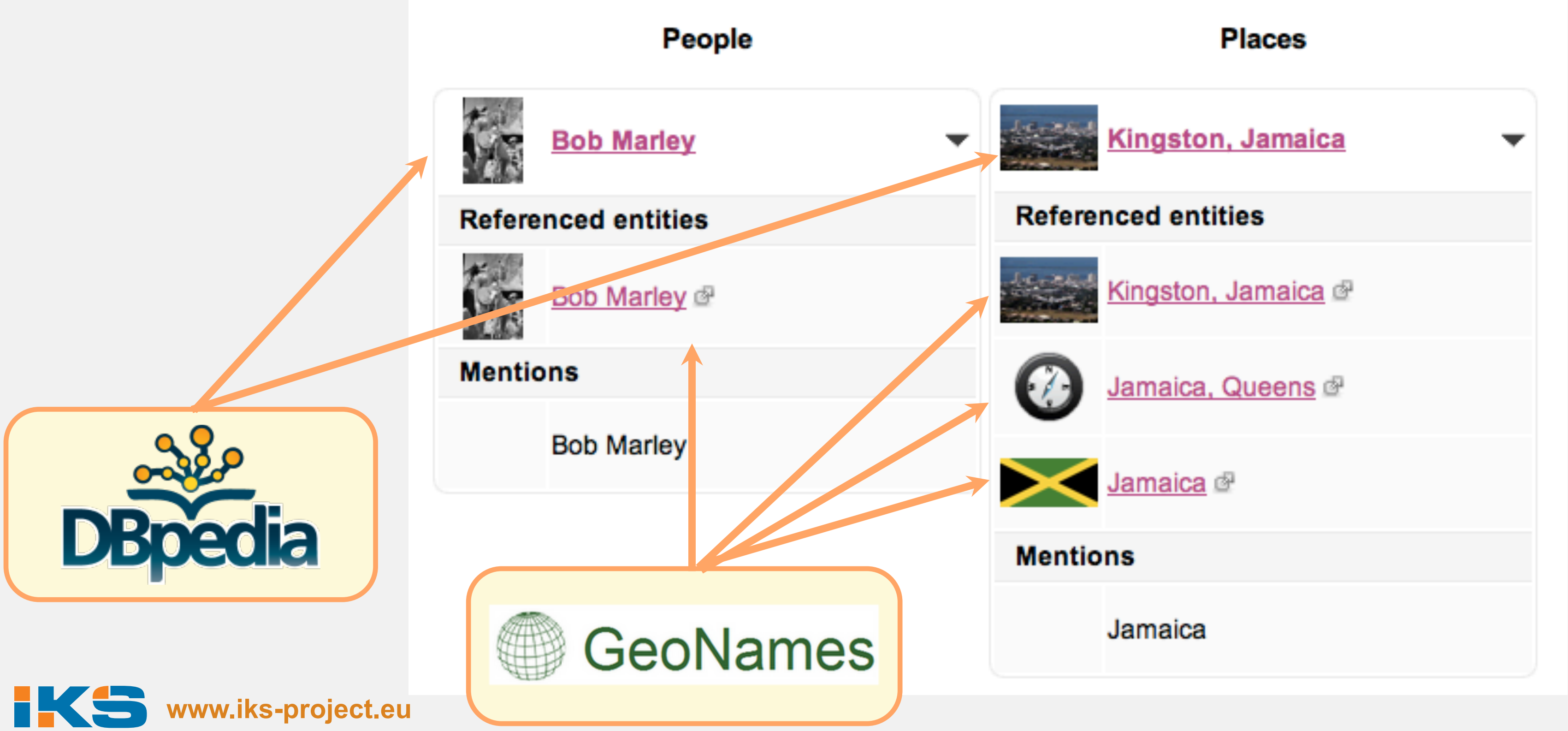
Jamaica

Mentions

Jamaica

Stanbol Enhancer

- Input
  - "Bob Marley was a famous musician from Jamaica."
- Output



People

Bob Marley

Referenced entities

Bob Marley

Mentions

Bob Marley

Places

Kingston, Jamaica

Referenced entities

Kingston, Jamaica

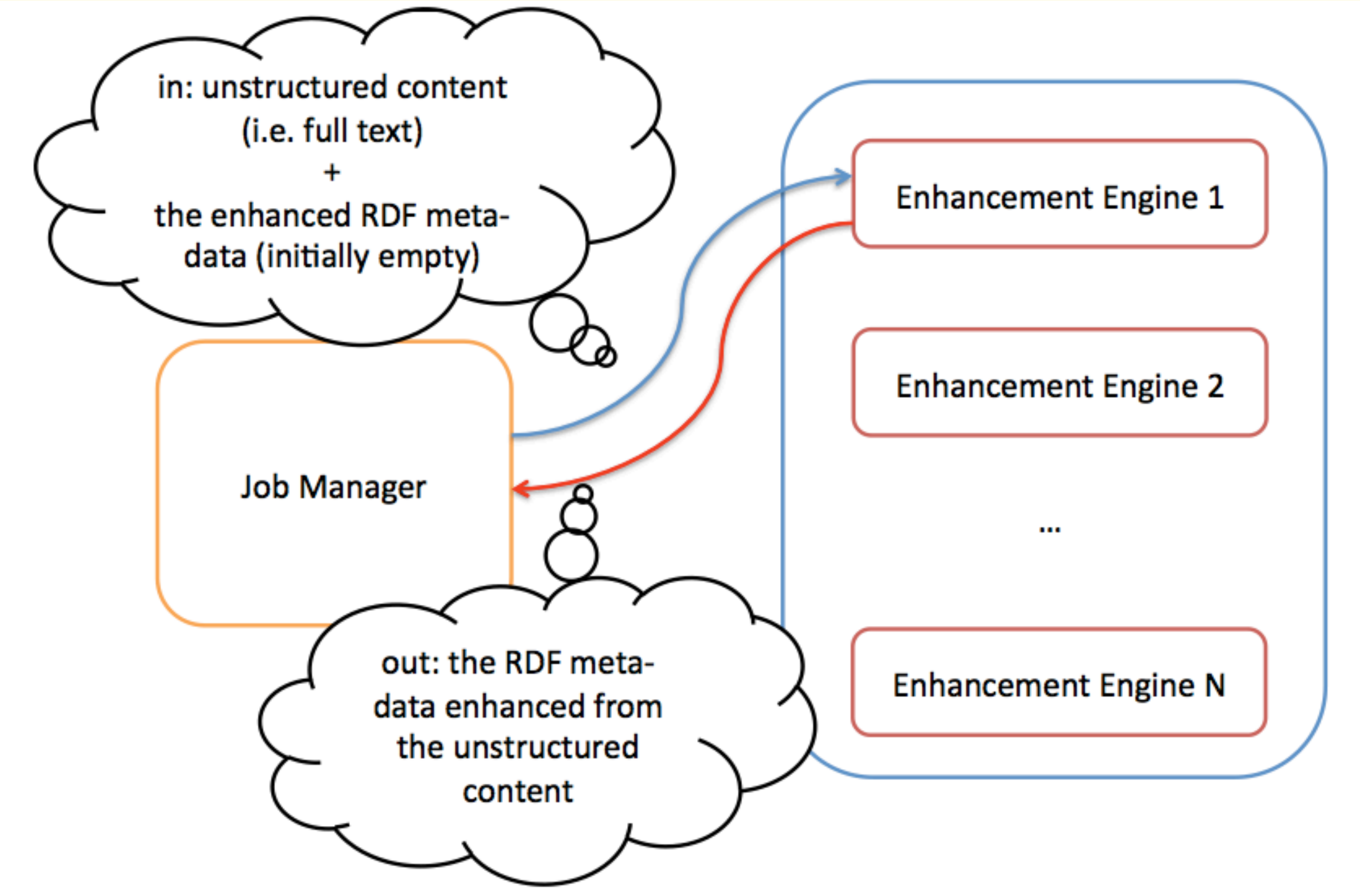
Jamaica, Queens

Jamaica

Mentions

Jamaica

How the Enhancer works



in: unstructured content (i.e. full text) + the enhanced RDF meta-data (initially empty)

Job Manager

Enhancement Engine 1

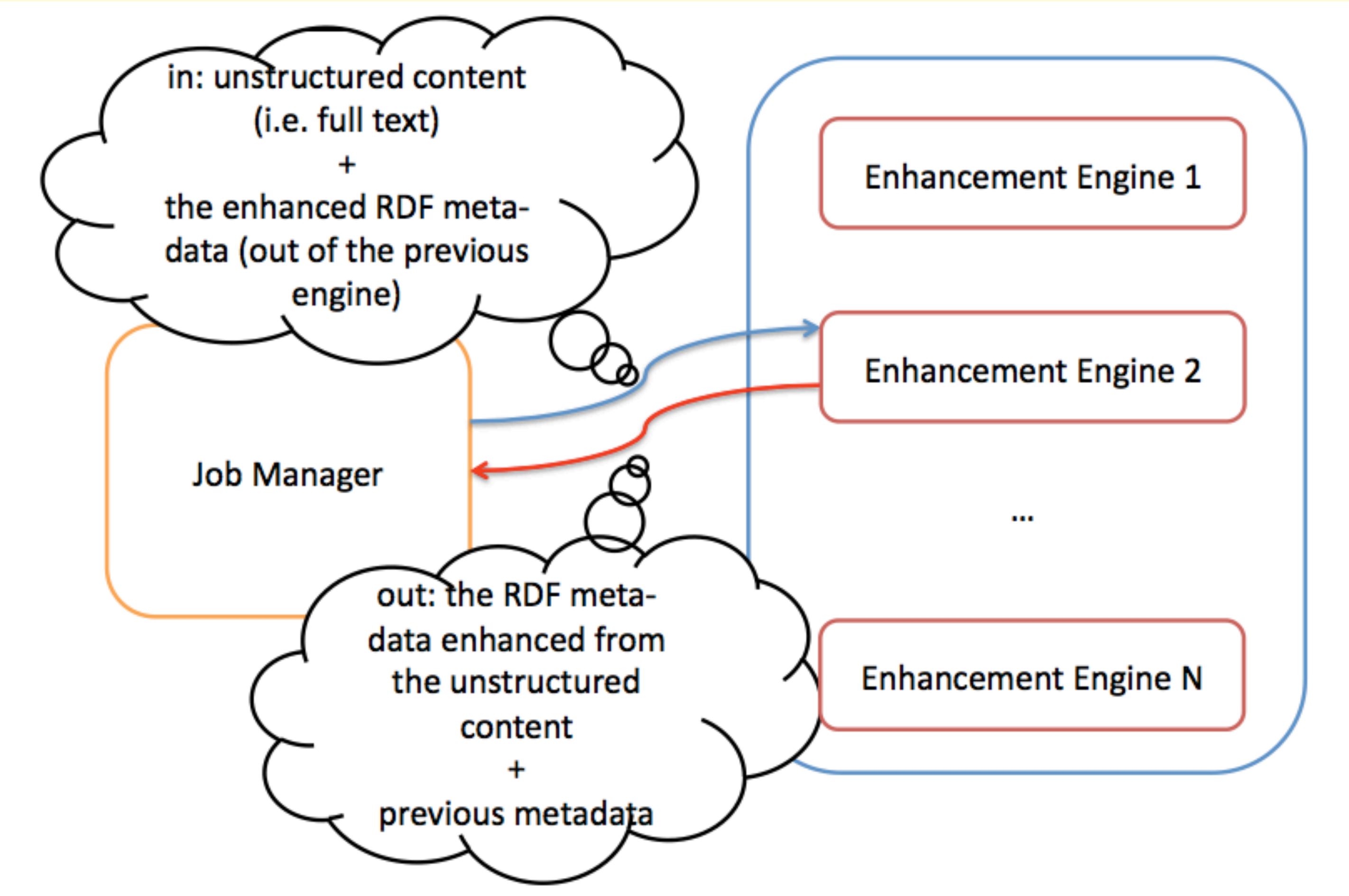
Enhancement Engine 2

...

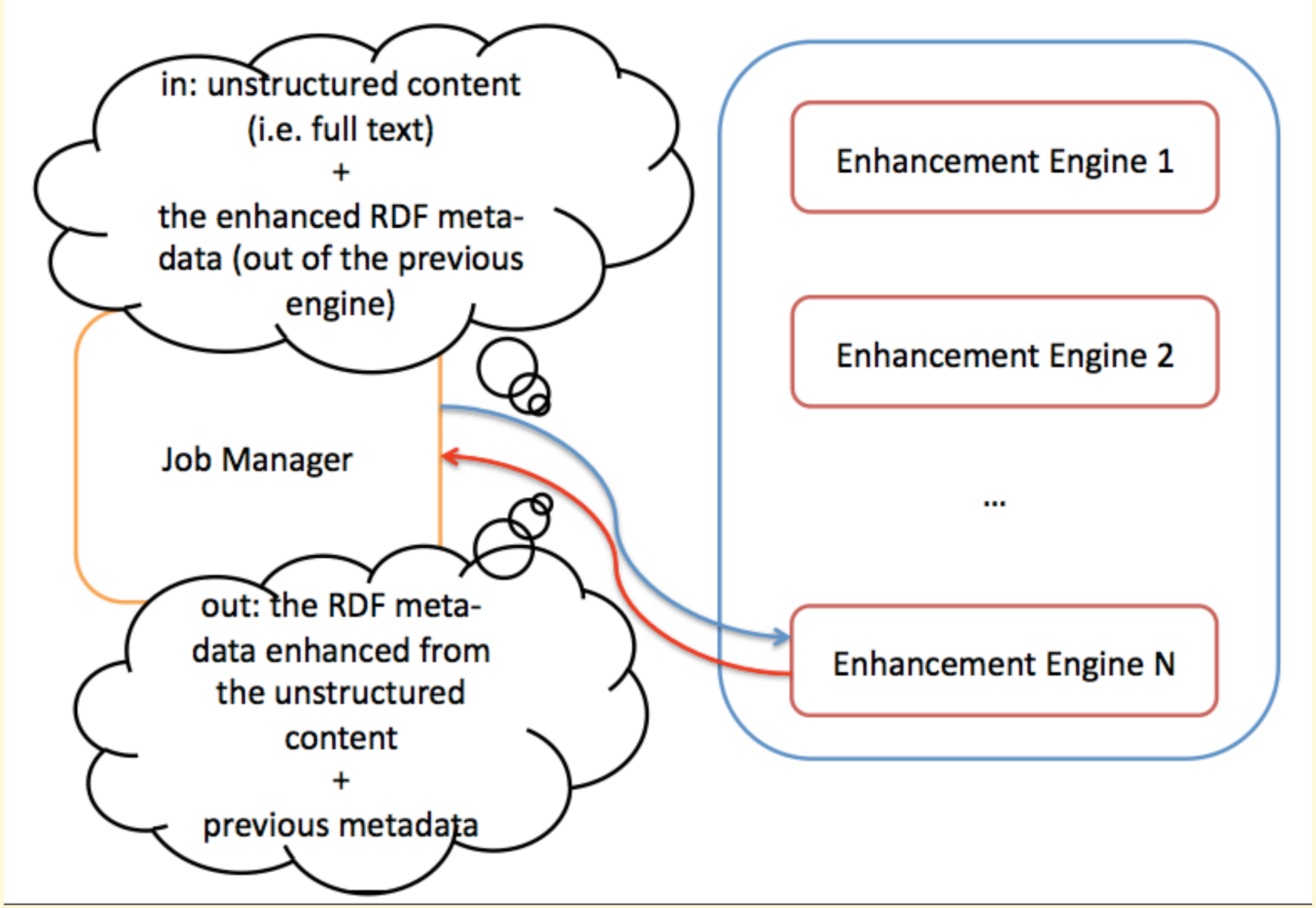
Enhancement Engine N

out: the RDF meta-data enhanced from the unstructured content

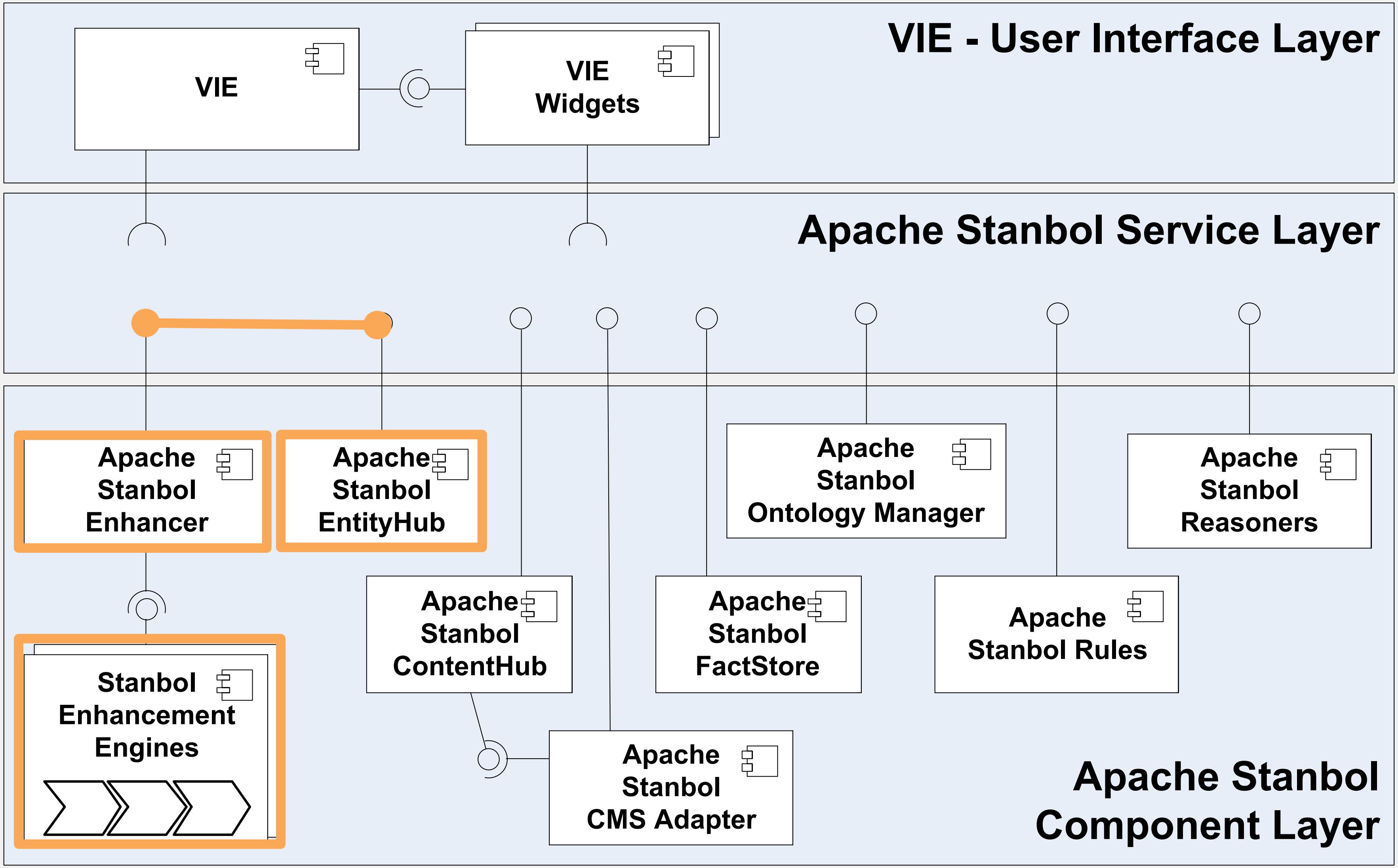
How the Enhancer works


in: unstructured content (i.e. full text) + the enhanced RDF meta-data (out of the previous engine)
Job Manager
Enhancement Engine 1
Enhancement Engine 2
...
Enhancement Engine N
out: the RDF meta-data enhanced from the unstructured content + previous metadata

How the Enhancer works


in: unstructured content (i.e. full text) + the enhanced RDF meta-data (out of the previous engine)
Job Manager
out: the RDF meta-data enhanced from the unstructured content + previous metadata
Enhancement Engine 1
Enhancement Engine 2
...
Enhancement Engine N

Entity Hub



VIE - User Interface Layer

Apache Stanbol Service Layer

Apache Stanbol Component Layer

Semantic Lifting

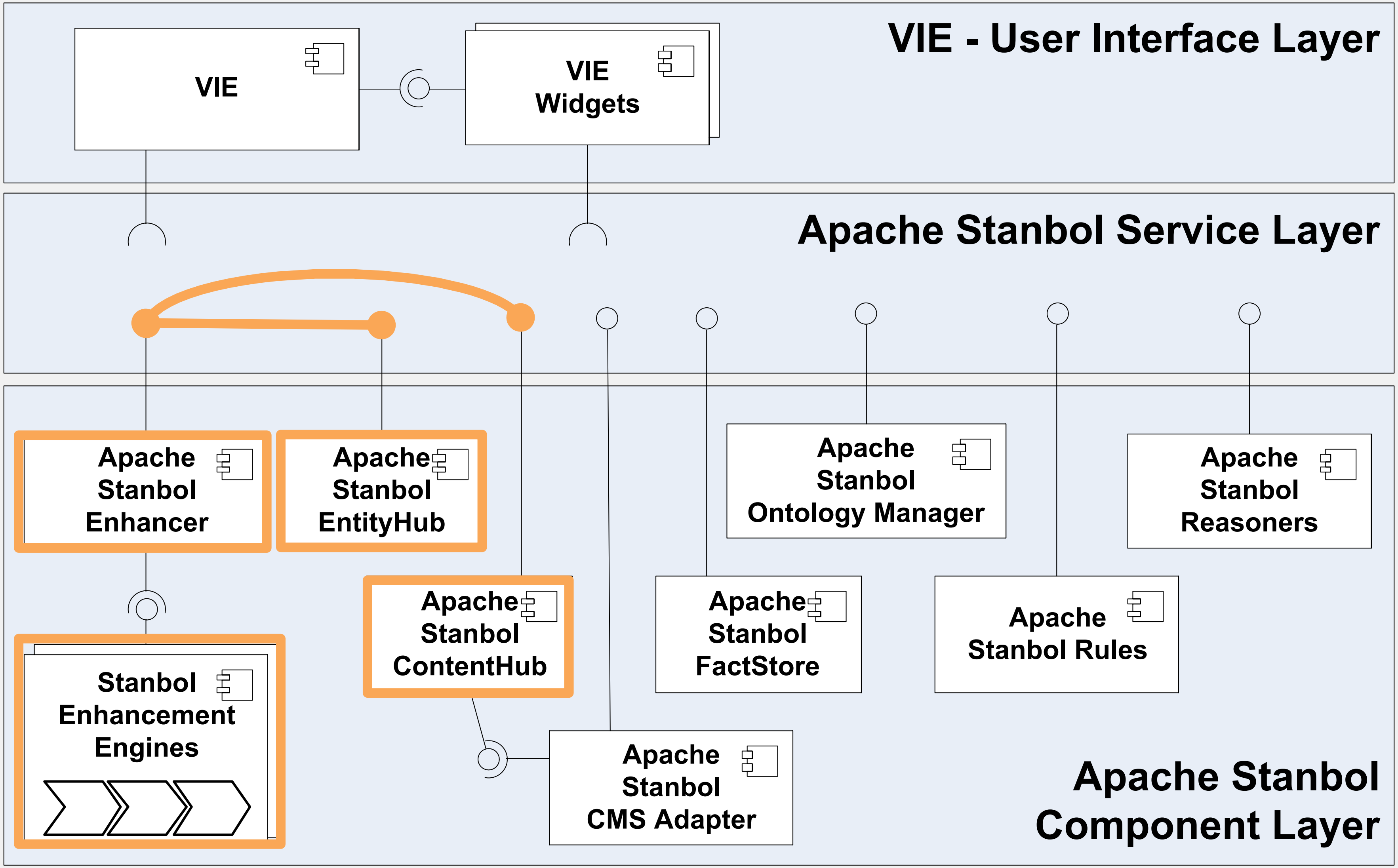
Knowledge Representation & Reasoning

Entityhub Features

- Manage a network of remote sites for fast entity lookup
- Caching of externally retrieved entity information
- CRUD management of local entities

- Examples
  - Use DBPedia linked open data source to retrieve additional information for entities
  - Use a customized vocabulary for local entities

Content Hub


VIE - User Interface Layer
VIE
VIE Widgets
Apache Stanbol Service Layer
Apache Stanbol Enhancer
Apache Stanbol EntityHub
Apache Stanbol Ontology Manager
Apache Stanbol Reasoners
Apache Stanbol ContentHub
Apache Stanbol FactStore
Apache Stanbol Rules
Stanbol Enhancement Engines
Apache Stanbol CMS Adapter
Apache Stanbol Component Layer

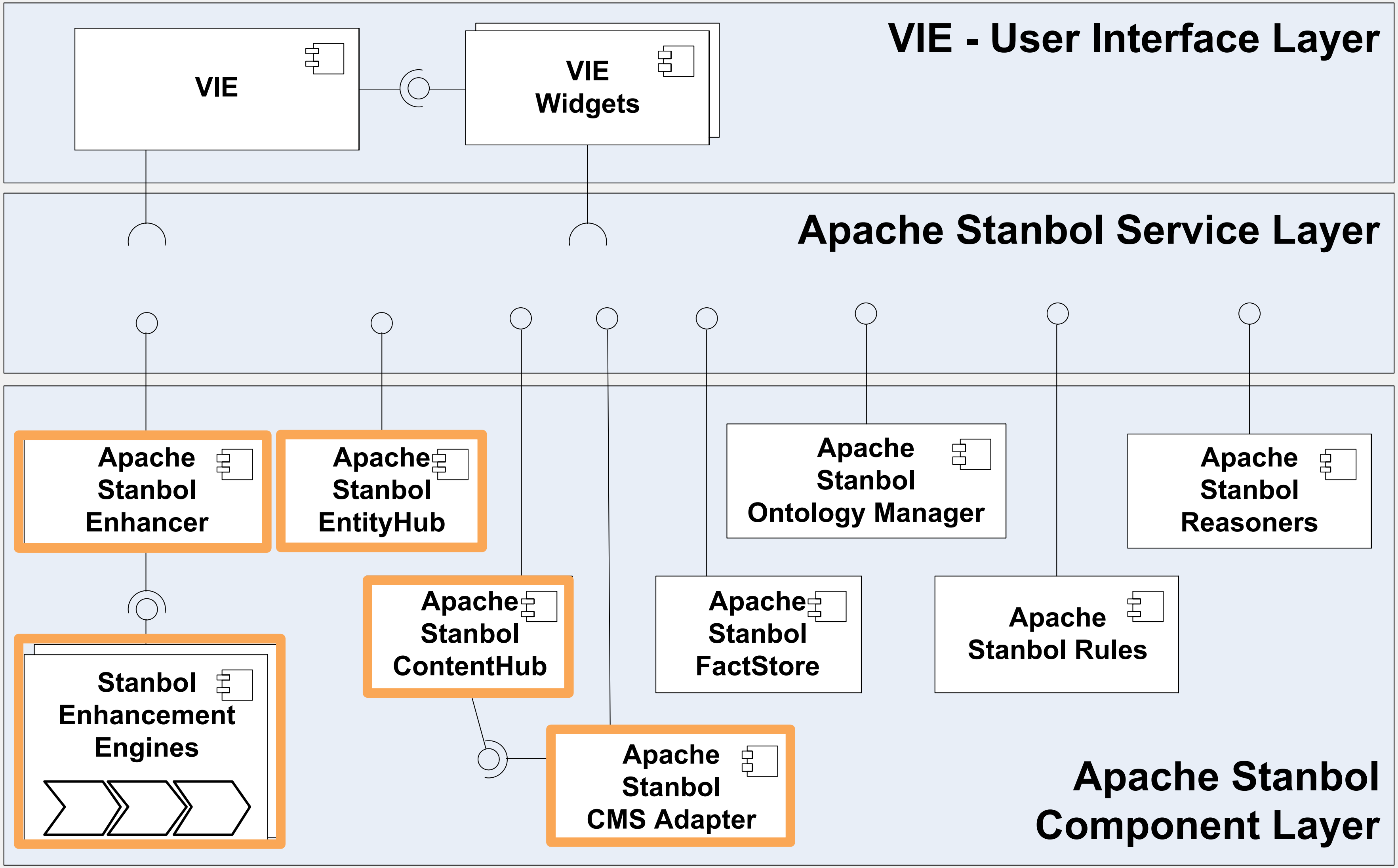
Semantic Lifting

Knowledge Representation & Reasoning

Contenthub Features

- Document repository by indexing retrieved documents
- Supports indexing of additional semantic metadata provided along the content
- Search facilities
  - Keyword Search
  - Faceted Search based on available semantic metadata

CMS Adapter



VIE - User Interface Layer
VIE
VIE Widgets
Apache Stanbol Service Layer
Apache Stanbol Enhancer
Apache Stanbol EntityHub
Apache Stanbol Ontology Manager
Apache Stanbol Reasoners
Stanbol Enhancement Engines
Apache Stanbol ContentHub
Apache Stanbol FactStore
Apache Stanbol Rules
Apache Stanbol CMS Adapter
Apache Stanbol Component Layer

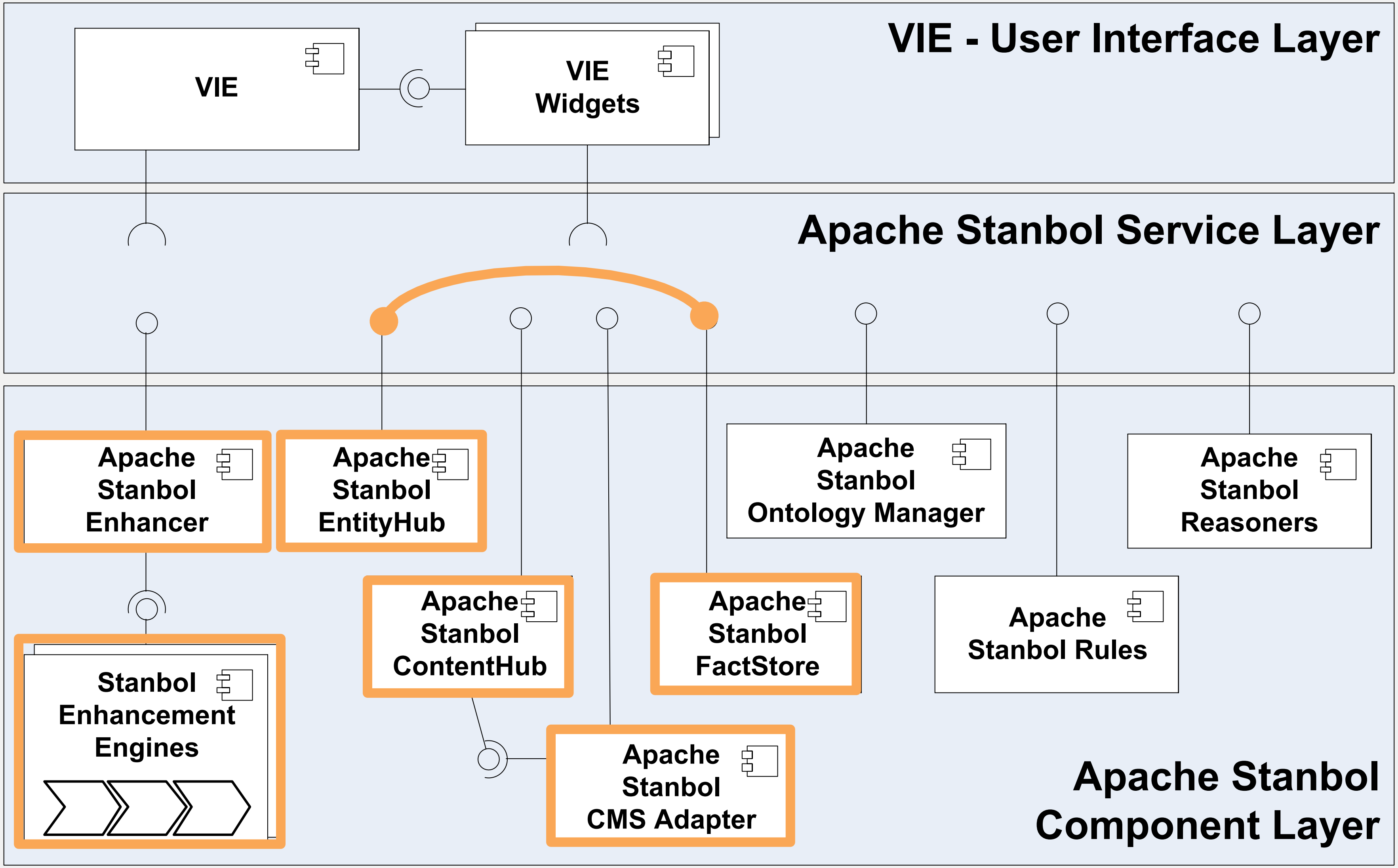
Semantic Lifting

Knowledge Representation & Reasoning

CMS Adapter Features

- Bootstrapping component to import content from a CMS into Apache Stanbol
- Import content from a CMIS/JCR compliant CMS into the Apache Stanbol Contenthub

Fact Store


VIE - User Interface Layer
VIE
VIE Widgets
Apache Stanbol Service Layer
Apache Stanbol Enhancer
Apache Stanbol EntityHub
Apache Stanbol Ontology Manager
Apache Stanbol Reasoners
Stanbol Enhancement Engines
Apache Stanbol ContentHub
Apache Stanbol FactStore
Apache Stanbol Rules
Apache Stanbol CMS Adapter
Apache Stanbol Component Layer

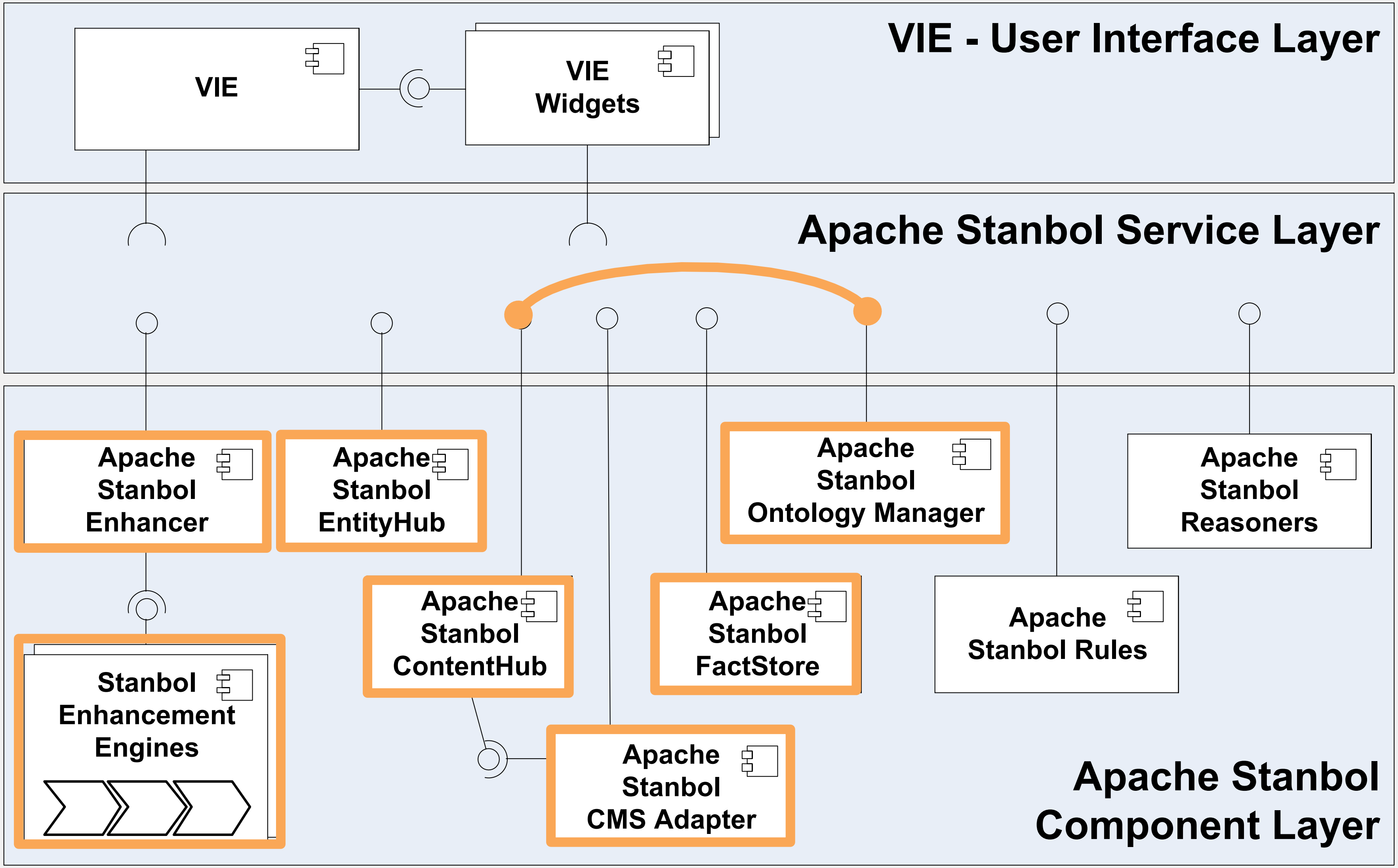
Semantic Lifting

Knowledge Representation & Reasoning

Fact Store Features

- Simple storage for relations between entities, i.e. facts
- Definition of custom semantic relations, i.e. fact schemata
- Not limited to triples – support for N-ary relations
- Simple query language for facts, no SPARQL

Ontology Manager



VIE - User Interface Layer

VIE

VIE Widgets

Apache Stanbol Service Layer

Apache Stanbol Enhancer

Apache Stanbol EntityHub

Apache Stanbol Ontology Manager

Apache Stanbol Reasoners

Stanbol Enhancement Engines

Apache Stanbol ContentHub

Apache Stanbol FactStore

Apache Stanbol Rules

Apache Stanbol CMS Adapter

Apache Stanbol Component Layer

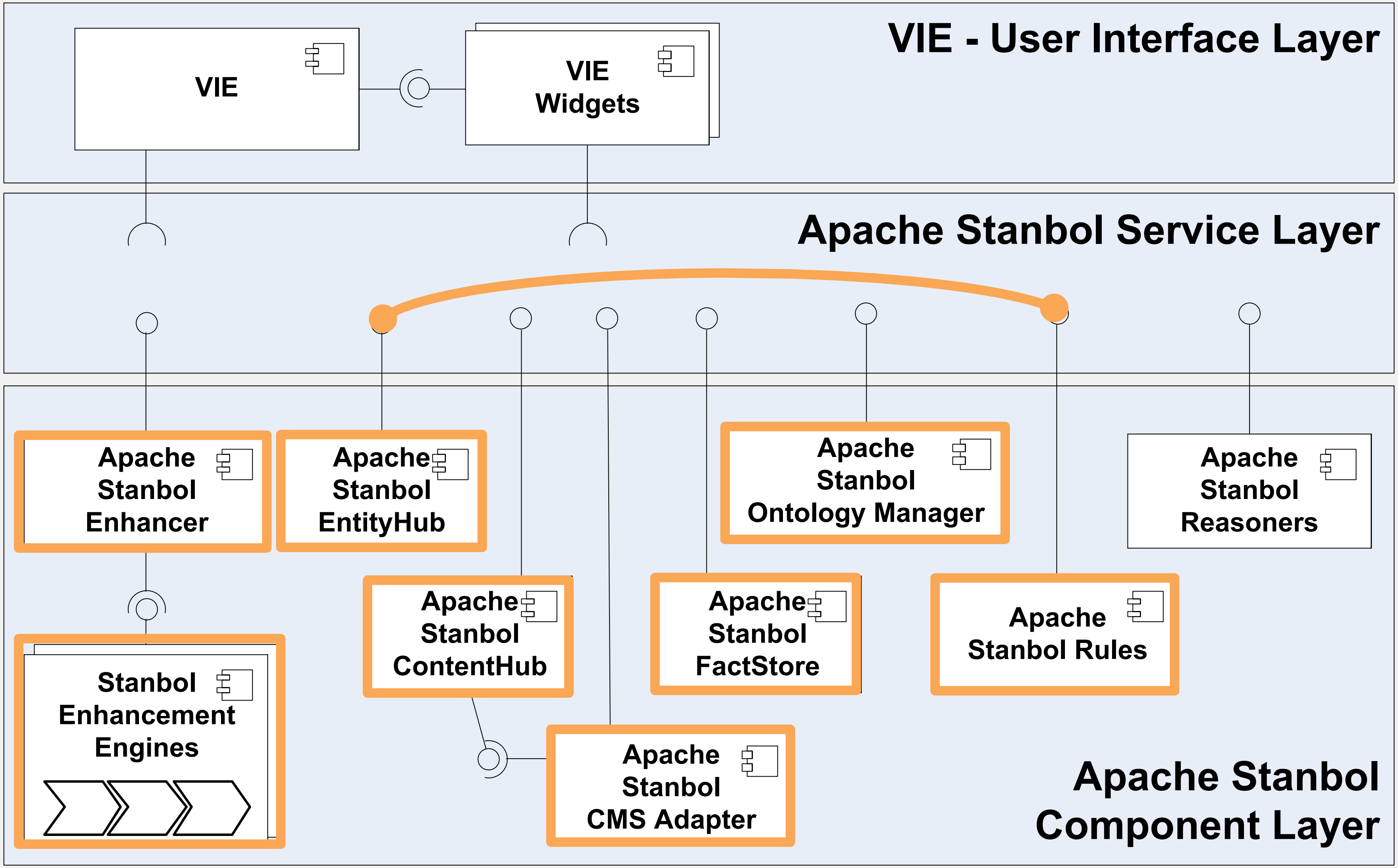
Semantic Lifting

Knowledge Representation & Reasoning

Ontology Manager Features

- Controlled environment for managing ontologies
- Manage ontology networks to activate/deactivate parts of complex ontologies
- Manage user sessions for ontologies allowing local user changes

Rules


VIE - User Interface Layer
VIE
VIE Widgets
Apache Stanbol Service Layer
Apache Stanbol Enhancer
Apache Stanbol EntityHub
Apache Stanbol Ontology Manager
Apache Stanbol Reasoners
Stanbol Enhancement Engines
Apache Stanbol ContentHub
Apache Stanbol FactStore
Apache Stanbol Rules
Apache Stanbol CMS Adapter
Apache Stanbol Component Layer

Semantic Lifting

Knowledge Representation & Reasoning

Rules Features

- Construction and execution of inference rules
- Inference rules, also called transformation rules, take premises and return conclusions
- Rules can be organized in recipes which allow to execute a set of rules as a whole

- Example
  - Define rules for doing integrity checks on data fetched from heterogeneous external data sources

What is a rule?

- In logic, a *transformation rule* or *rule of inference* is a syntactic rule or function which takes premises and returns a conclusion
- The rule is **sound** with respect to the semantics of classical logic in the sense that if the premises are interpreted to be true then so is the conclusion.

Rule Examples

- Rule pattern (modus ponens)
  - *if condition then consequent*

- A rule example
  - ***if** X is a person **then** X has a father*

    (i.e. every person has a father)
    - $\forall x.\ \exists y.\ Person(x) \Rightarrow hasFather(x, y)$

  - ***if** Y is the father of X and Z the brother of Y **then** Z is the uncle of X*

    (i.e. the brother of the father is the uncle)
    - $\forall xyz.\ hasFather(x,y)\ \wedge\ hasBrother(y,z) \Rightarrow hasUncle(x,z)$

Stanbol Rule Syntax

In Stanbol a rule is defined as

ruleName[body -> head]

where:

- The ruleName identifies the rule
- The body is a set of **atoms** that must be satisfied by evaluating the rule
- The head or consequent is a set of **atoms** that must be true if the condition is evaluated to be true

Stanbol Rule Syntax

ruleName[body -> head]

Where

- Both body and head consist of a list of conjunctive atoms
  - *body = atom1 . atom2 . … . atomN*
  - *head = atom1 . atom2 . … . atomM*
- The conjunction ∧ in Stanbol Rules is expressed with the symbol “ . ”

Example of rule

Considering Stanbol Rules, the formula

$$hasFather(x,y) \land hasBrother(y,z) \Rightarrow hasUncle(x,z)$$

expresses in predicate calculus becomes

```
myRule[ has(<http//myont.org/hasFather>, ?x, ?y) .
has(<http/myont.org/hasBrother>, ?y, ?z)
                                    ->
has(<http//myont.org/hasUncle>, ?x, ?z) ]
```

Rule Atoms

- An atom is the smallest unit of the interpretation of a rule
  - e.g.: in predicate calculus

    Person(x) ⇒ hasFather(x, y)

    Person(•) and hasFather(•,•) are two atoms

- In Stanbol basic atoms are
  - Class assertion atom
  - Individual assertion atom
  - Data value assertion atom
  - Range assertion atom
- There are also comparison atoms, string and integer manipulation atoms

Atom’s notation

- The atoms may contain
  - Constants: they consist of URI (we are in Web context) or Literal (values)
    - e.g. http//dbpedia.org/resource/Bob_Marley is a constant, but “Bob Marley”^^xsd:string is a constant too
  - Variables: any identifier preceded by **?**
    - e.g. ?x is a variable, but also ?y is a variable

Class assertion atom

A class assertion atom is identified by the operator

is(classPredicate, argument)

where

- *classPredicate* is a URI that identifies a class
- *argument* is the resource that has to be proven as typed with the classPredicate. It can be both a constant (a URI) or a variable

e.g. is(<http://xmlns.com/foaf/0.1/Person>, ?x) returns true if the concrete value associated to ?x is typed as

http://xmlns.com/foaf/0.1/Person

Individual assertion atom

has(properyPredicate, arg1, arg2)

where

- *propertyPredicate* is the object property that has to be evaluated. It can be a constant (URI) or a variable (?x)
- *arg1* and *arg2* are the two arguments of the property. They can be either constants (URI) or variables (?x)

Data value assertion atom

values(properyPredicate, arg1, arg2)

where

- *propertyPredicate* is the data property that has to be evaluated. It can be a constant (URI) or a variable (?x)
- *arg1* can be either a constant (i.e. URI) or a variable (i.e. ?x)
- arg2 can be either a constant (i.e. a literal) or a variable (i.e. ?x)

Namespace Prefixes

- URIs are useful, but sometime too long for humans
- We could use namespace prefixes instead of full URIs in rule atoms
- E.g:

```
myont = <http//myont.org/> .
myRule[ has(myont:hasFather, ?x, ?y) .
has(myont:hasBrother, ?y, ?z)
                                              ->
has(myont:hasUncle, ?x, ?z) ]
```

Comparison atoms

- *same(arg1, arg2)*: returns true if *arg1* is equal to *arg2*
- *different(arg1, arg2)*: returns true if *arg1* is different from *arg2*
- *greaterThan(arg1, arg2):* returns true if *arg1* > *arg2*
- *lessThan(arg1, arg2)*: returns true if *arg1* < *arg2*
- *startsWith(arg1, arg2)*: returns true if the string associated to *arg1* starts with the string associated to *arg2*
- *endsWith(arg1, arg2)*: returns true if the string associated to *arg1* ends with the string associated to *arg2*

String manipulation

- *concat(arg1, arg2)*: returns a string that is the concatenation of arg1+arg2
- *substring(arg, start, length): returns the sub-string of arg from position start for length chars*
- *lowercase(arg):* returns the lower case representation of arg
- *uppercase(arg):* returns the upper case representation of arg
- *str(arg):* returns the literal value of any RDF object
- namespace(arg): returns the namespace as a string of any URI
  - e.g. namespace(<http://www.foo.org#obj>) -> "http://www.foo.org#"

- localname(arg): returns the local as a string of any URI
  - e.g. localname(<http://www.foo.org#obj>) -> "obj"

Production atoms

newIRI(arg1, arg2)

where

- arg1 is a variable
- arg2 is an expression that returns a literal
- e.g*: newIRI(?x, “http://stlab.istc.cnr/Aldo Gangemi”)* binds the variable ?x to the URI obtained from the literal *http://stlab.istc.cnr/Aldo_Gangemi, namely*
  - *<http://stlab.istc.cnr/Aldo_Gangemi>*

Production atoms (contd)

newLiteral(arg1, arg2)

where

- arg1 is a variable
- arg2 is an expression that returns a literal
- e.g*: newLiteral(?x, concat("Aldo ", "Gangemi"))* binds the variable ?x to the string literal obtained from the literal "*Aldo Gangemi", namely "Aldo Gangemi"*

Arithmetical atoms

- *sum(arg1, arg2):* returns a new integer the is equal to *arg1+arg2*
- *sub(arg1, arg2):* returns a new integer the is equal to *arg1-arg2*
- mult*(arg1, arg2):* returns a new integer the is equal to *arg1*arg2*
- div*(arg1, arg2):* returns a new integer the is equal to *arg1/arg2*

- *arg1* and *arg2* can be numerical expression or numbers

Example of rule

Considering Stanbol Rules, the formula

hasFather(x,y) ⋀ hasBrother(y,z) ⇒ hasUncle(x,z)

expresses in predicate calculus becomes

myRule[has(<http//myont.org/hasFather>, ?x, ?y) .
has(<http/myont.org/hasBrother>, ?y, ?z)
->
has(<http//myont.org/hasUncle>, ?x, ?z)]

Stanbol Refactor

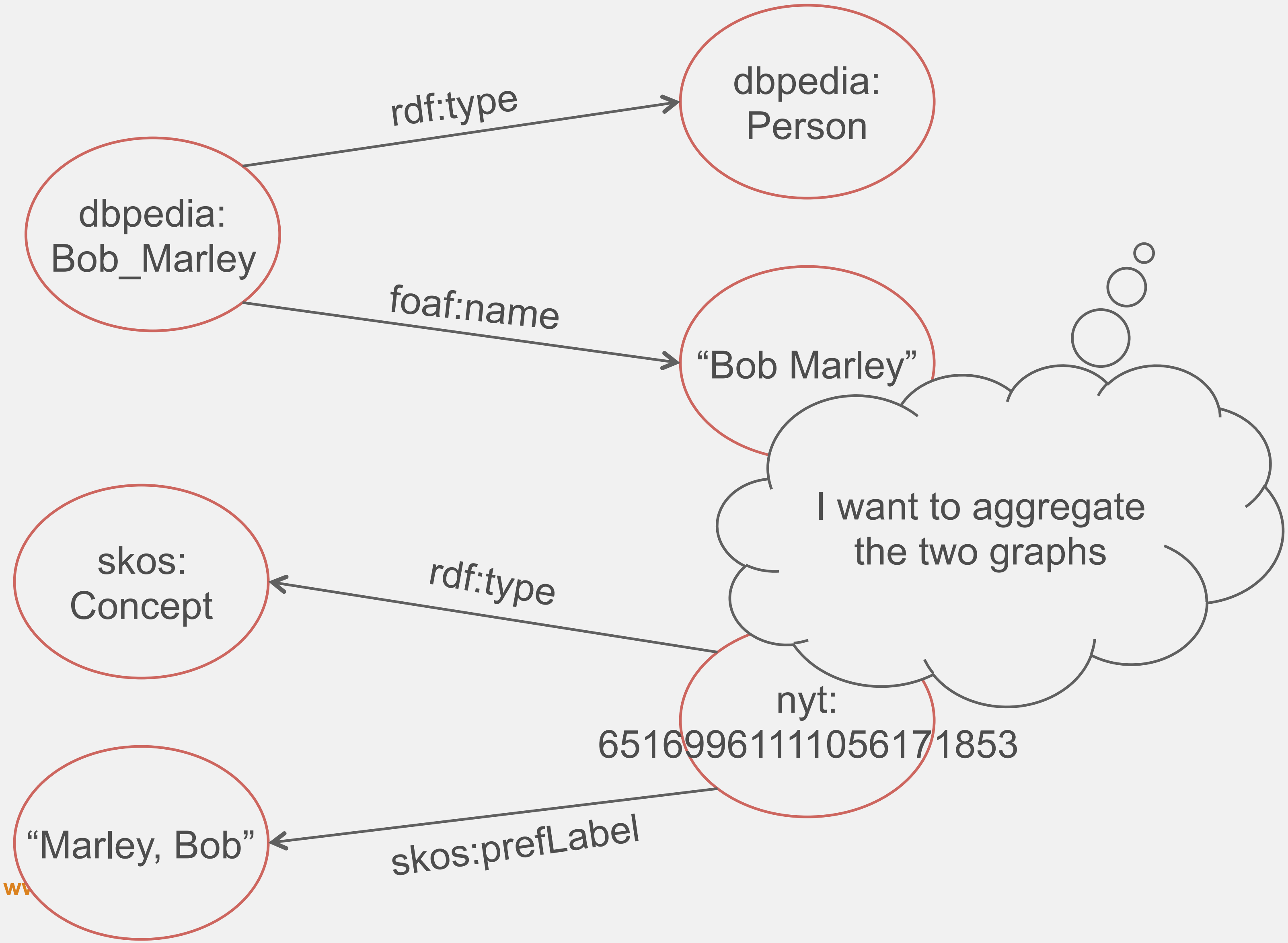
- Add to Stanbol a framework for RDF graph refactoring
- Rule-based refactoring

Why do I need refactoring?

- A possible scenario
  - My system fetches knowledge from different sources in LOD
  - Each of these sources uses its own ontology/vocabulary

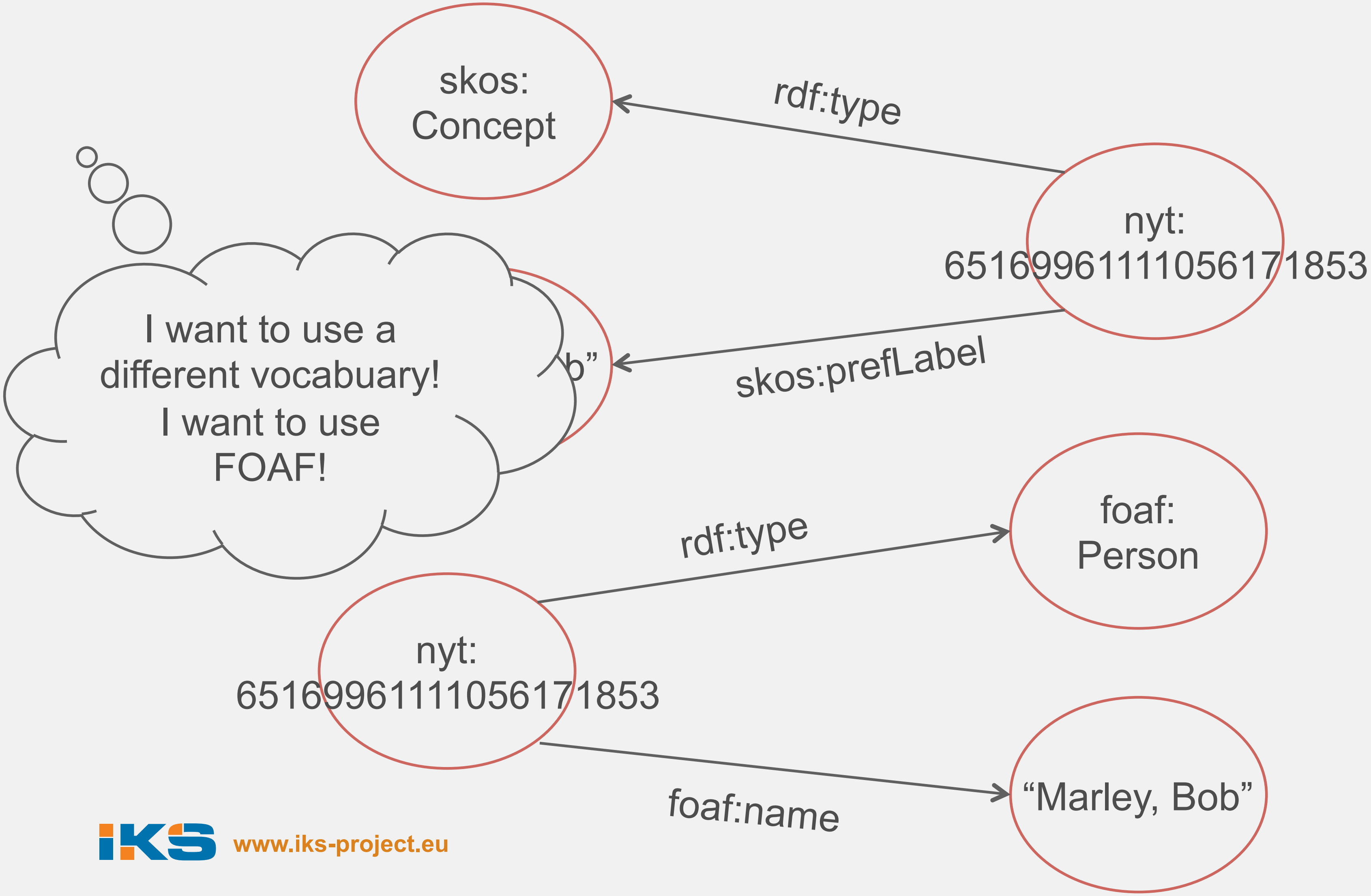
How to add a homogeneous representation of knowledge expressed with heterogeneous vocabularies?

An example



I want to aggregate the two graphs

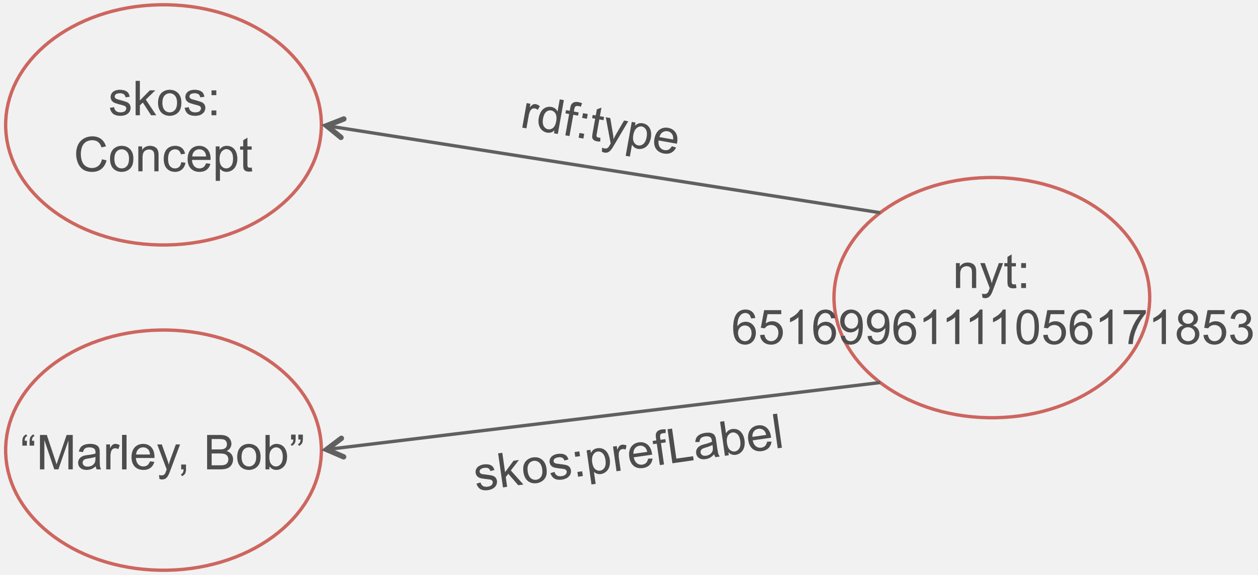
Another example



Rule-based refactoring

- The Stanbol Refactor applies RDF graph transformation by means of transformation rules
- Rules drive the transformation
- A set of rules for a tranformation task characterize a **recipe**
- A recipe identifies the kind of transformation and the rules need by the transformation task

Define a refactoring recipe



We want to use the FOAF vocabulary instead of SKOS

Define a refactoring recipe

```
skos = <http://www.w3.org/2004/02/skos/core#> .
foaf = <http://xmlns.com/foaf/0.1/> .

conceptToPerson[ is(skos:Concept, ?x) -> is
   (foaf:Person, ?x) ] .
labelRule[ values(skos:prefLabel, ?x, ?y) ->
   values(foaf:name, ?x, ?y) ]
```

How the refactor works

- Each rule is executed individually starting from the first in the recipe
- Each rule is interpreted and executed as a SPARQL CONSTRUCT

From rules to CONSTRUCT

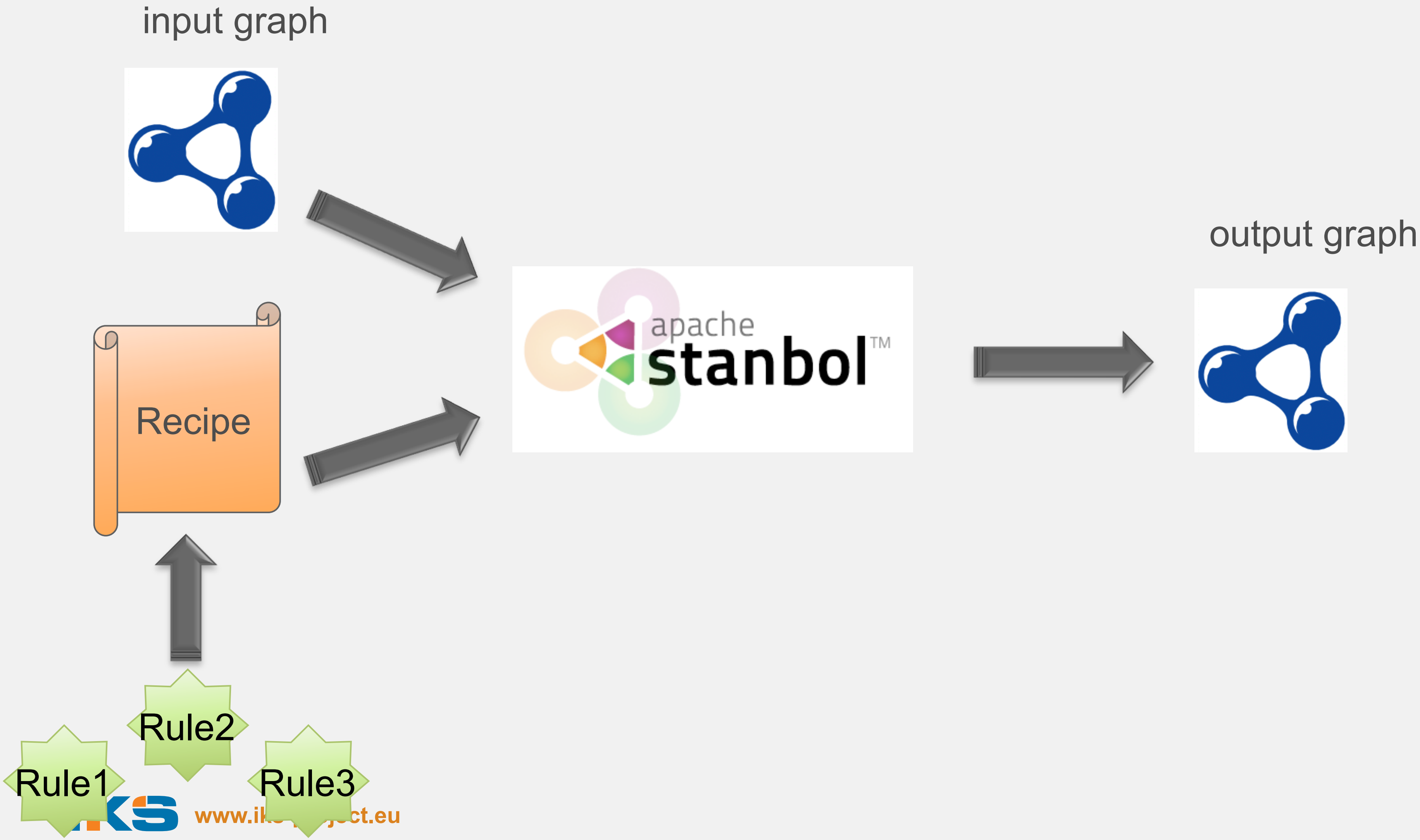
The rule

```
skos = <http://www.w3.org/2004/02/skos/core#> .
foaf = <http://xmlns.com/foaf/0.1/> .

conceptToPerson[ is(skos:Concept, ?x) -> is(foaf:Person, ?x) ]
```

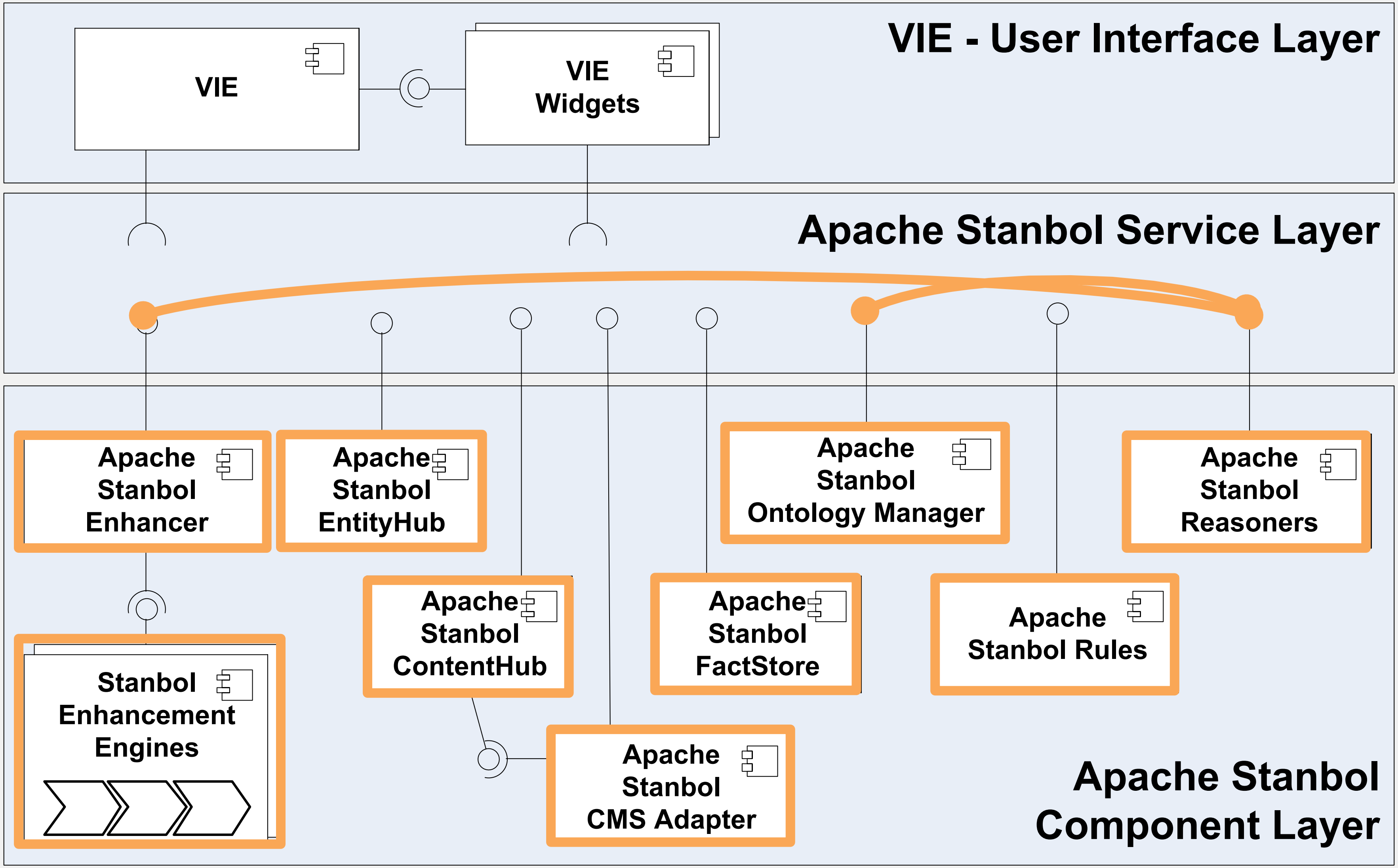
is interpreted as

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT { ?x rdf:type foaf:Person }
WHERE { ?x rdf:type skos:Concept }
```

Recipes, rules and Stanbol



Reasoners



VIE - User Interface Layer

VIE

VIE Widgets

Apache Stanbol Service Layer

Apache Stanbol Component Layer

Apache Stanbol Enhancer

Apache Stanbol EntityHub

Apache Stanbol Ontology Manager

Apache Stanbol Reasoners

Stanbol Enhancement Engines

Apache Stanbol ContentHub

Apache Stanbol FactStore

Apache Stanbol Rules

Apache Stanbol CMS Adapter

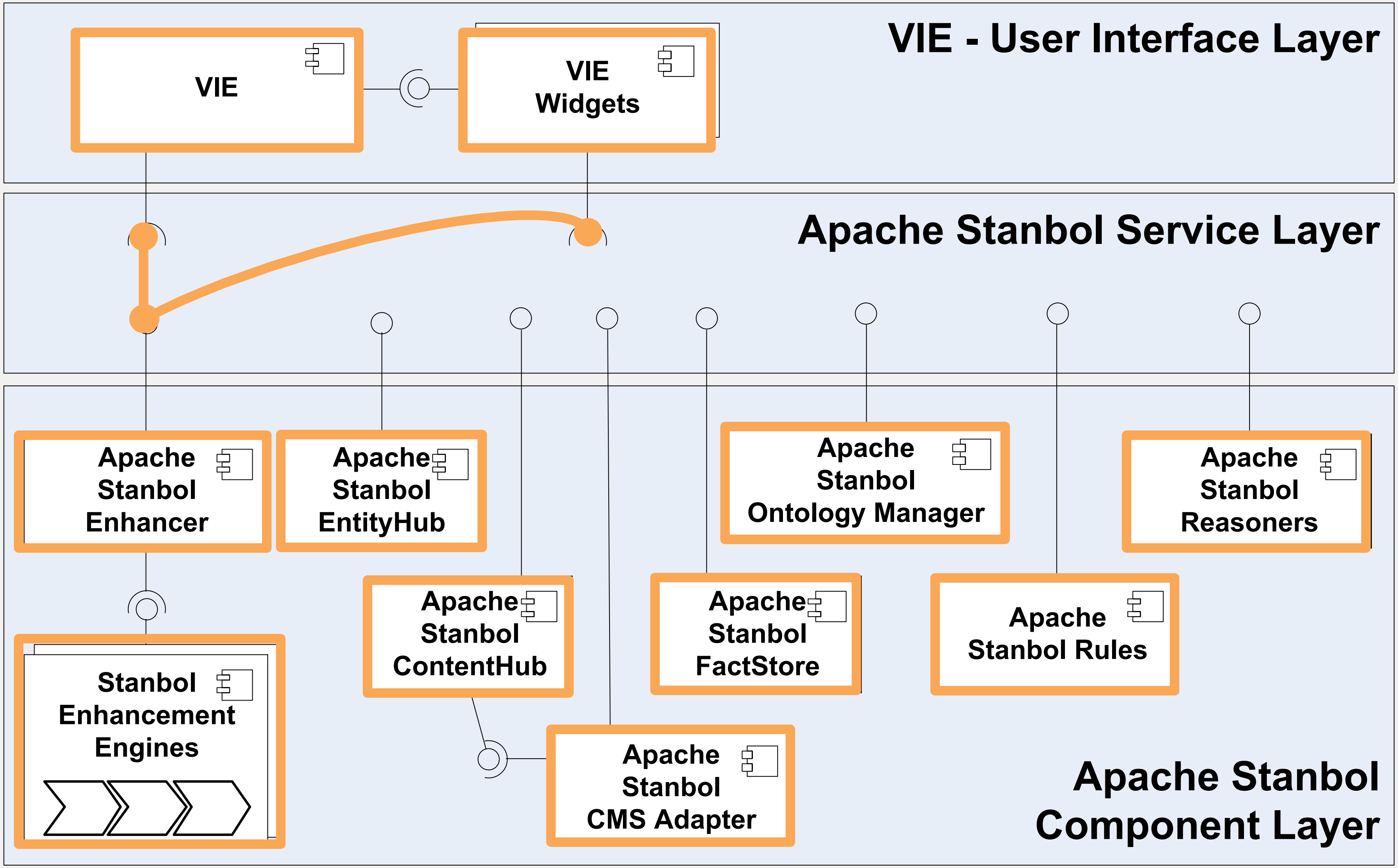
Semantic Lifting

Knowledge Representation & Reasoning

Reasoners Features

- Common API for existing (open source) reasoning services
- Supports different reasoners and configuration in parallel
- Supported third-party reasoners
  - Jena RDFS
  - OWL
  - OWLMini
  - HermiT

VIE & VIE Widgets



VIE - User Interface Layer

VIE

VIE Widgets

Apache Stanbol Service Layer

Apache Stanbol Enhancer

Apache Stanbol EntityHub

Apache Stanbol Ontology Manager

Apache Stanbol Reasoners

Stanbol Enhancement Engines

Apache Stanbol ContentHub

Apache Stanbol FactStore

Apache Stanbol Rules

Apache Stanbol CMS Adapter

Apache Stanbol Component Layer

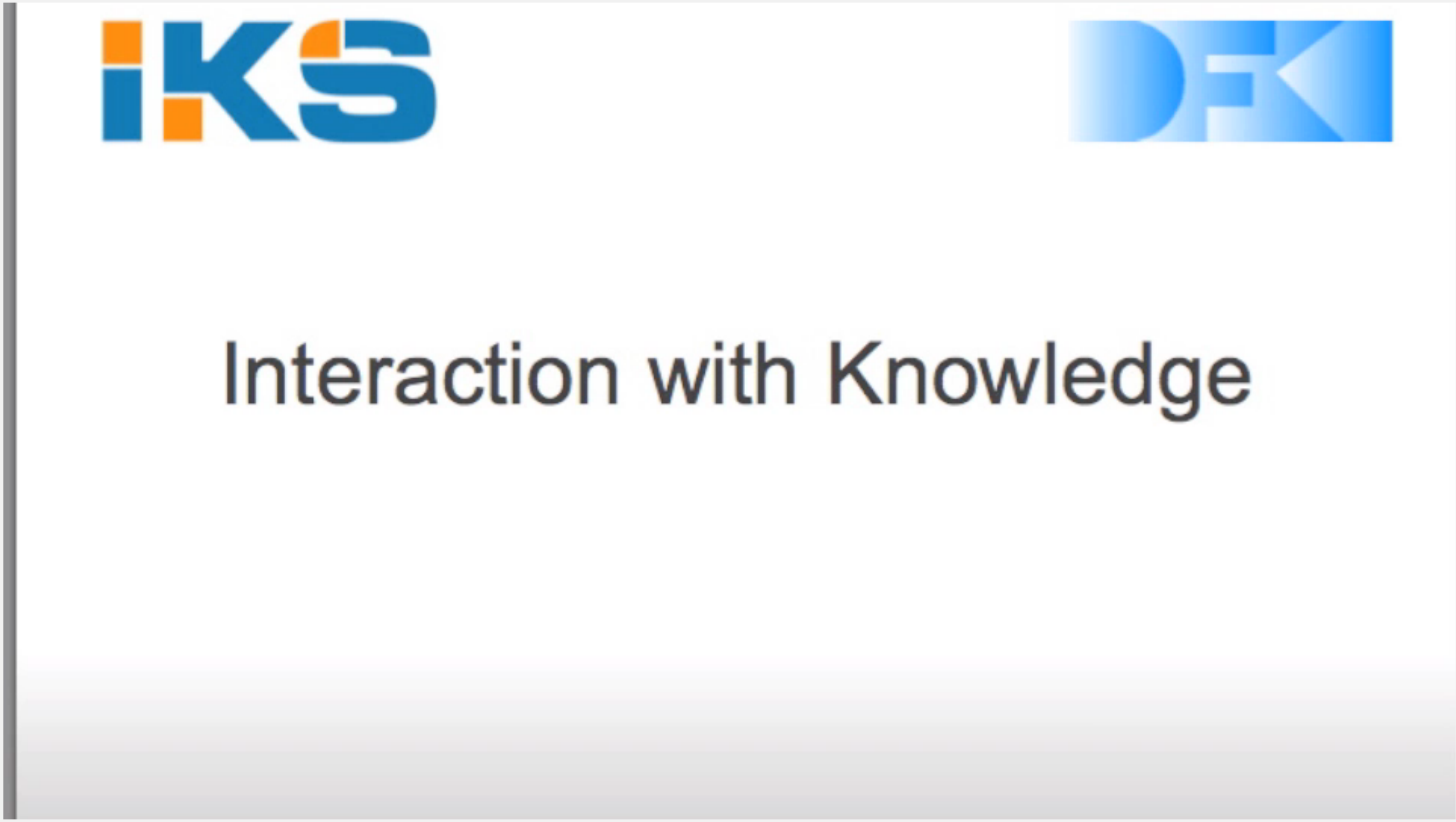
Semantic Lifting

Knowledge Representation & Reasoning

VIE & VIE Widgets Features

- VIE is a JavaScript library for implementing decoupled CMS and semantic interaction in web applications
- VIE provides easy access to the semantic metadata (RDFa) within a web page
- VIE Widgets are user interface components that implement semantic user interactions
- Examples
  - Semantic image search
  - Automatic tagging of entities
  - Semi-automatic content annotation

VIE Demo



Interaction with Knowledge

License

- IKS software is licensed under business-friendly open source software licenses.
- IKS software can be freely used / changed / distributed in your products.

- For the rare cases where artifacts use a less permissive license, you will find a notice.
  - e.g. we use models for natural language processing from the Apache OpenNLP project whose licenses are not clarified, yet.

Get in Contact

- VIE
  - Homepage
    http://viejs.org
  - Google User Group
    https://groups.google.com/forum/#!forum/viejs

- Apache Stanbol
  - Homepage
    http://incubator.apache.org/stanbol
  - Mailinglist subscription
    stanbol-dev-subscribe@incubator.apache.org