



Leverage of Semantic Web Services for Practical Application: Creating Transferable Methodology with SADI Case Studies

Alexandre Riazanov, UNB Saint John

Based on joint work with Chris Baker and other colleagues from UNB, Concordia and Carleton

AWOSS 10.2, Moncton

Semantic Web Service Vision

- Web Services are programs that can be called remotely over HTTP. Useful, in particular, in applications requiring flexible data integration, distributed computation or just extra flexibility in the composition of workflows.
- Small Semantic Web Service vision: easy service look-up. *How can users find my WS that computes a full address for a postcode?* Describe the semantics of the input and output of the WS: input **is-a** string, output **is-a** Address.
- Big vision: **completely automated discovery and composition**. The user specifies **WHAT** he wants to compute, some software finds the right services and executes them completely automatically.
- A reincarnation of the program synthesis idea and more general declarative programming concept.

Practical adoption?

- Close to non-existent: google “Semantic Web Service registry/search/lookup ...”

Why?

- General answer: the complexity of solutions does not match the added value.
- Semantics-based look-up seems doable, but **not sufficiently appealing**.
- Automatic discovery and composition can solve major problems in data integration in some application domains like bioinformatics or e-commerce, but **existing approaches are rather complex**.
- **No killer application** yet that would both demonstrate the value of SWS and show possible development paths.

SADI – A Novel Semantic Services Framework

- Semantic Automated Discovery and Integration
- SADI is a **set of conventions** for creating Semantic Web Services that can be **automatically discovered and orchestrated**.
- SADI does not create new technologies or message formats. It **relies on well-established standards**: RDF, OWL and HTTP.
- **Both powerful and simple.**
- Has killer **killer app** candidate: integration of bioniformatics resources.

What we do with SADI in UNB Saint John

- C-BRASS – a trans-Canadian CANARIE-funded project.
- Mission: make bioinformatics a killer app for SADI.
- Strategy: create a **critical mass of SADI services** for bioinformatics databases and algorithmic resources. Develop SADI **infrastructure**.
- Credo of our UNB SJ group – **coherent case studies**.

Talk Plan

- Technical(ish) introduction to SADI: what SADI can do and what it takes to use it.
- Case study with information about impacts of mutations on protein properties, text-mined from biological publications.
- Case study on integrating disparate ontology-based lipid classification procedures in a pipeline.

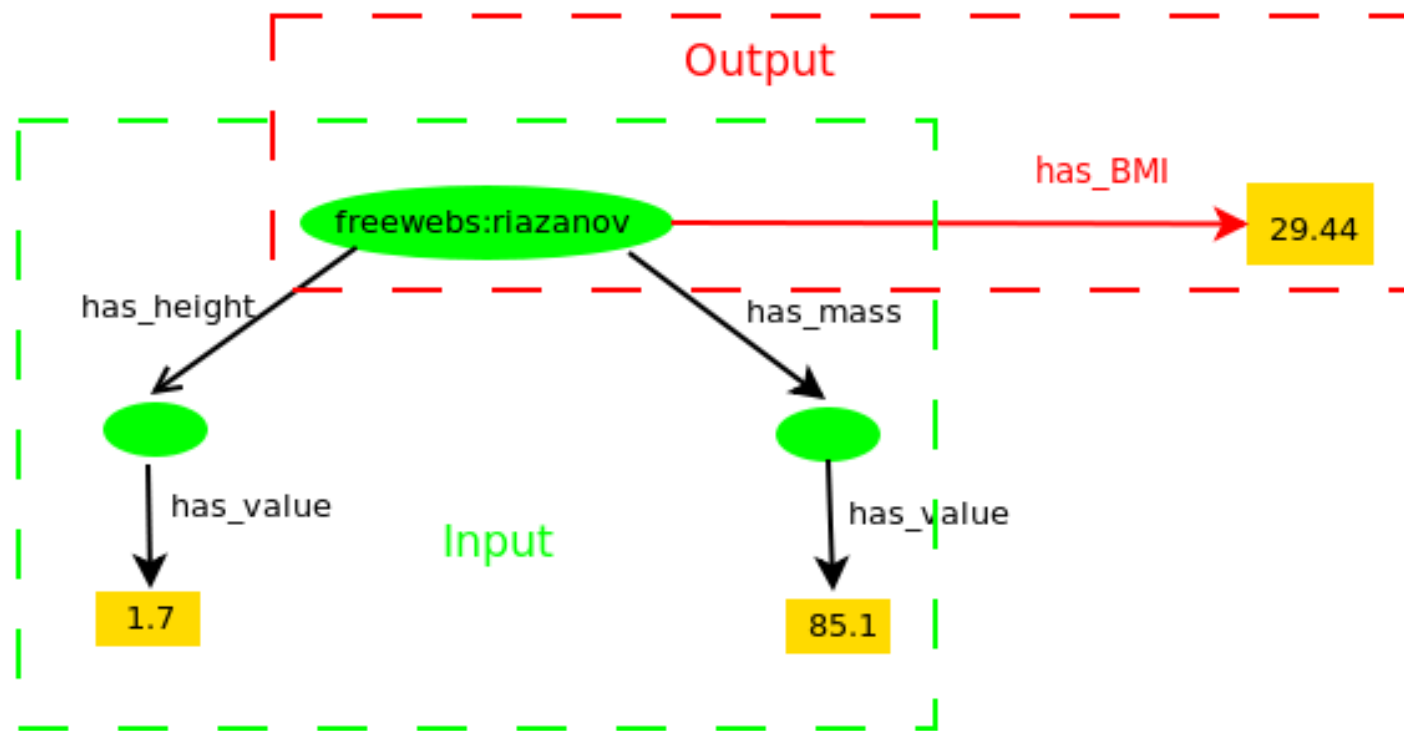
Part I: What is SADI and how it works

SADI Outline

- A SADI service is a regular HTTP service: GET delivers the metadata (semantic description) about the service and POST invokes the service.
- A SADI service **consumes an RDF graph** with a designated node and produces an **RDF graph** about the same node with some **new properties attached**.
- Declaration of the added property predicates **describes the semantics** of the service and makes it **discoverable**.
- SADI services can serve information from **DBs** or by running **analytical resources**:
 - EntrezGene ID \longrightarrow symbolic name of the gene and its textual description (retrieval)
 - weight and height of a person \longrightarrow Body Mass Index (on-the-fly computation)

SADI Service Input/Output

- RDF \rightarrow RDF
- One main input node \rightarrow same node in the output + new properties
- Output = annotated input



Input Class

- The OWL class of the main input node, specifying what the service expects to see in the input.
- Can be just an OWL class name from a domain ontology: *foaf : Person*.
- Typically, a bunch of existential property restrictions: **which properties on the main input node the service expects.**
- If the input class is defined as

`(has_height some (has_value some xsd:float))`

and

`(has_mass some (has_value some xsd:float))`

the service expects something like this in the input:

`<http://www.freewebs.com/riazanov> has_height [has_value float"1.7"] .`

`<http://www.freewebs.com/riazanov> has_mass [has_value float"85.1"] .`

Output Class

- The OWL class of the main output node specifying **what the service promises to produce as the output.**
- Should imply some existential property restrictions. **The corresponding predicates semantically define what the service does.**
- The service computing BMI has this output class:

`(has_BMI exactly 1 xsd:float)`

The service advertises that it can compute the Body Mass Index.

- We say that the service *annotates* its input with the predicate *has_BMI*.

Typical output:

```
<http://www.freewebs.com/riazanov> has_BMI float"29.44"
```

Making Use of Semantic Service Declarations: Automatic Service Discovery and Composition

- Services are registered in a registry and indexed by the predicates they provide.
- If a client program needs to compute the BMI of a person, it will
 - look for a service providing *has_BMI*
 - check, possibly with the help of a reasoner, that the client's data match the input class of a found services
- If we only have a person's URI, but there is a service decorating the URI with *has_height* and *has_mass*, the client program can figure out that it should call this service prior to calling the *has_BMI* service.
- **This is automatic service discovery and composition!**
- Service calls can be interleaved with arbitrary reasoning steps deriving more facts logically, using an OWL TBox or a rule set
 - ⇒ **Very intelligent computing!**

Automatic discovery and composition with SHARE

- SHARE is a prototypical client for SADI services.
- For the user, it is
 - SPARQL querying over a virtual RDF graph build from
 - some initial RDF graph
 - by adding information that can be computed by calls to SADI services
 - and derivations made by an OWL reasoner
- The user just submits a SPARQL query and gets the results – **the service discovery and composition is completely transparent.**

SHARE snapshot

SPARQL query:

```
PREFIX bmi: <http://sadirframework.org/examples/bmi.owl#>  
SELECT ?patient ?bmi  
FROM <http://biordf.net/cardioSHARE/patients.rdf>  
WHERE {  
  ?patient rdf:type patients:AtRiskPatient .  
  ?patient bmi:BMi ?bmi  
}
```

 [View results as RDF](#). There were warnings executing the query. Click for details.

Submit

Query results

bmi	patient
26.32017237098755	http://biordf.net/cardioSHARE/patients.r

Writing SADI Services is Easy!

- There is a Java library that isolates the business logic of a service from the environment: HTTP POST → Jena Model (RDF graph) → ... business logic ... → Jena Model (RDF graph) → HTTP response
- Service provider implements all the business logic in

```
void processInput(Resource input, Resource output)
```
- Whole project skeleton is generated: provider only needs to supply basic parameters, like the service name. Or use a skeleton Eclipse project.
- Similar things for Perl exist.
- The hardest part is knowledge engineering: defining what the service does ontologically

Typical Resource Publishing Scenario

- **Identify a source of data or a piece of software you want to publish with SADI.** For example, if you have a DB about some substances with literature reference, you can have a service linking your substances to PubMed IDs
- **Model your data semantically:** find ontologies describing your domains and decide how you data will be expressed in the terms of these ontologies.
- **Model your services I/O semantically:** decide how to describe the operation of your services in the terms of the domain ontologies.
- **Code the business logic of your services in Java or Perl.** If a service wraps a DB, convert the input RDF into a query and the query results back to RDF. The effort is typically negligent, compared to the modelling

Part II: Case Study with Mutation Impact Mining Pipeline and DB

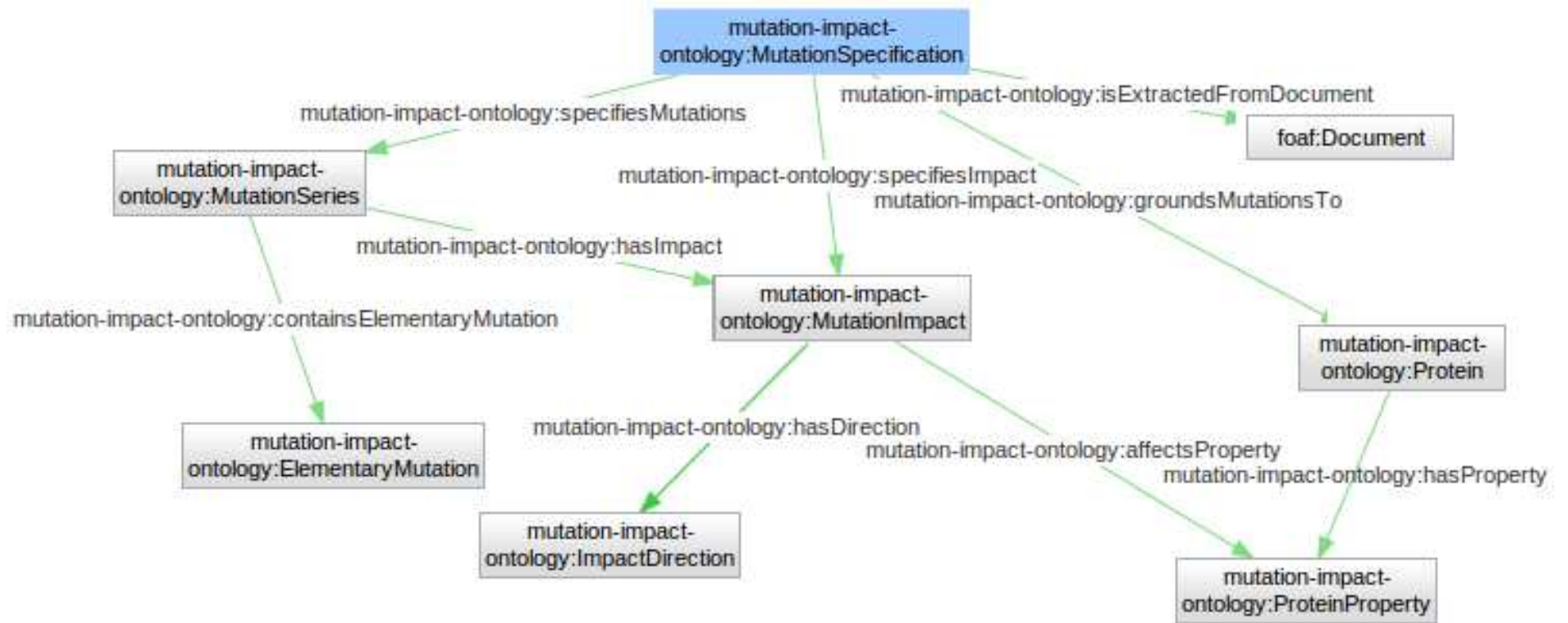
Mutation Impact Text-Mining Pipeline

The haloalkane dehalogenase from the nitrogen-fixing hydrogen bacterium *Xanthobacter autotrophicus* GJ10 (DhlA) prefers 1,2-dichloroethane (DCE) as substrate and converts it to 2-chloroethanol and chloride ... DhlA shows only a small decrease in activity when Trp-125 is replaced with phenylalanine

- “haloalkane dehalogenase” is a protein
- its UniProt Id is P22643
- “Trp-125 is replaced with phenylalanine” is the point mutation W125F
- “activity” is the protein property being affected, GO_00188786 in Gene Ontology
- “decrease” means the impact is negative

UNB-Concordia joint work. GATE pipeline wrapped as a Java library. Outputs Java objects representing mutation specifications: statements about mutations and their impacts on protein properties.

Mutation Impact Ontology



Main Service: Text → Mutation Specifications

- Just wraps the Mutation Impact mining pipeline as a SADI service
- Output can be submitted to our Mutation Impact DB (triplestore)

Input:

```
<http://example.com/text1>    rss:link    anyURI"http://example.com/text1"
```

Output:

```
<http://example.com/text1>    foaf:topic                midb:mutationSpec243
miodb:mutationSpec243         mio:groundsMutationsTo    miodb:protein528
miodb:protein528              mio:hasSwissProtId        "P22643"
miodb:mutationSpec243         mio:specifiesImpact       miodb:mutationImpact624
miodb:mutationImpact624      mio:hasDirection          mio:Positive
miodb:mutationImpact624      mio:affectsProperty       miodb:proteinProperty326
miodb:proteinProperty326     mio:hasType                GO:GO_0018786
```

.

Services based on Mutation Impact DB

- Wildtype protein → mutation specifications (complete description of)
- Mutant protein → mutation specifications
- Specific property of a specific protein → known mutation impacts
- Mutation impact (on a specific property of a specific protein) → mutation specifications
- Bio entity type (e.g., `mio:Protein` or `mio:PointMutation`) → known instances
- Set of elementary mutations → subsets described in the literature (with links to the corresponding `mio:MutationSpecification`)

Use Case 1

- A protein engineer is looking for mutations that can improve catalytic activity of an enzyme
- Query: find all mutations and the structure images of wild type proteins that were mutated, where the impact of the mutation is an enhanced haloalkane dehalogenase activity (GO_0018786)
- Predicates from our ontology take us from GO_0018786 to mutations and proteins: *proteinPropertyHasType + affectsProperty + hasDirection + specifiesImpact + containsElementaryMutation + groundMutationsTo*
- Two external SADI services provide *has3DStructure* to link proteins to their structure descriptions, and *hasJmol3DStructureVisualization* to link the structure to a 3D image file

Use Case 1 SHARE Query (simplified)

```
SELECT DISTINCT ?StructureImage ?NormalizedMutation
WHERE {
  # Property type --> property
  ?Property mio:proteinPropertyHasType go:GO_0018786 .

  # property --> positive impact
  ?Impact mio:affectProperty ?Property .
  ?Impact mio:hasDirection mio:Positive .

  # impact --> mutation spec --> mutation series --> elementary mut.
  ?MutationSpecification mio:specifiesImpact ?Impact .
  ?MutationSeries mio:mutationSeriesIsSpecifiedBy ?MutationSpecification .
  ?MutationSeries mio:containsElementaryMutation ?Mutation .
  ?Mutation mio:hasNormalizedForm ?NormalizedMutation .

  # wildtype protein
  ?MutationSpecification mio:groundMutationsTo ?Protein .

  # protein --> structure --> image file
  ?Protein pred:has3DStructure ?Structure .
  ?Structure objects:hasJmol3DStructureVisualization ?StructureImage . }
```

Use Case 2

- Systems biologist is seeking to understand the likely impact of reported mutations on signalling or metabolic pathways in which the mutated protein participates
- Query: find pathways, together with their images, that might have been altered by a mutation of the protein P22607
- Our predicates take us from P22607 to its mutations with positive or negative impact on some property:
groundsMutationsTo + specifiesImpact + hasDirection
- External predicate *isEncodedBy* links P22607 to the corresponding gene
- *isParticipantIn* links the gene to pathways
- *visualizedByPathwayDiagram* links a pathway to its image

Use Case 2 SHARE Query (simplified)

```
SELECT DISTINCT ?Pathway ?PathwayDiagram
WHERE {
  # protein --> mutation specification
  ?MutationSpecification mio:groundsMutationsTo uniprot:P22607 .

  # mutation spec --> impact
  ?MutationSpecification mio:specifiesImpact ?Impact .

  # impact cannot be neutral
  {?Impact mio:hasDirection mio:Positive}
  UNION {?Impact mio:hasDirection mio:Negative} .

  # protein --> gene
  uniprot:P22607 pred:isEncodedBy ?Gene .

  # gene --> pathway
  ?Gene ont:isParticipantIn ?Pathway .

  # pathway --> image
  ?Pathway pred:visualizedByPathwayDiagram ?PathwayDiagram
}
```

Use Case 3

- A researcher in drug discovery is looking for existing drugs targeting a new disease condition
- Query: find all drugs related to mutated proteins, together with their interaction partners, where the mutation impact is a increased carbonic anhydrase activity (GO_0008270)
- Our predicates link GO_0008270 to proteins via the instances of this protein properties, positive impacts and mutation specifications
- External ontologies facilitate the linking of proteins to the IDs of drugs affecting them
- Another external predicate, *hasMolecularInteractionWith*, links the proteins to proteins they interact with

Use Case 3 SHARE Query (simplified)

```
SELECT ?DrugName ?InteractingProtein
WHERE {
  # protein property type --> instance
  ?Property mio:proteinPropertyHasType go:GO_0008270 .

  # protein property --> positive impact
  ?Impact mio:affectProperty ?Property .
  ?Impact mio:hasDirection mio:Positive .

  # impact --> mutation spec
  ?MutationSpecification mio:specifiesImpact ?Impact .

  # mutation spec --> protein
  ?MutationSpecification mio:groundMutationsTo ?Protein .

  # protein --> drug id --> drug name
  ?Protein objects:isTargetOfDrug ?Drug .
  ?Drug objects:hasDrugGenericName ?DrugName .

  # protein --> interaction partner
  ?Protein pred:hasMolecularInteractionWith ?InteractingProtein }
```

Use Case 4

- A researcher in genomics asks for all known mutations reported in the literature for a protein containing a non-synonymous SNP
- Query: find all reported mutations of the protein with the nsSNP rs2305178
- External predicate *correspondsToEntrezGene* finds the Entrez ID of the gene corresponding to rs2305178
- Predicates *correspondsToEntrezGene*, *hasRefSeqTranscript* and *isEncodedBy* link the Entrez gene ID to the sequence, to the KEGG gene ID, and then to the protein
- Complication: there are services that provide *correspondsToEntrezGene* and *hasRefSeqTranscript*, but no services yet providing their inverses: KEGG ID \rightarrow sequence \rightarrow Entrez ID, but not Entrez ID \rightarrow sequence \rightarrow KEGG ID
- Temporary solution: enumerate proteins from Mutation Impact DB, and then link them to other entities

Use Case 4 SHARE Query (simplified)

```
SELECT DISTINCT ?NormalizedMutation ?Document
WHERE {
  # enumerate known proteins
  ?Protein mioe:biologicalEntityHasType mio:Protein .

  # protein --> KEGG gene ID --> sequence --> Entrez gene ID
  ?Protein pred:isEncodedBy ?KeggGene .
  ?KeggGene objects:hasRefSeqTranscript ?RefSeq .
  ?RefSeq objects:correspondsToEntrezGene ?EzGene .

  # SNP --> Entrez gene ID (and join)
  dbsnp:rs2305178 objects:correspondsToEntrezGene ?EzGene .

  # protein --> mutation spec --> mutation and document
  ?MutationSpecification mio:groundMutationsTo ?Protein .
  ?MutationSeries mio:mutationSeriesIsSpecifiedBy ?MutationSpecification .
  ?MutationSeries mio:containsElementaryMutation ?Mutation .
  ?Mutation mio:hasNormalizedForm ?NormalizedMutation .
  ?Document foaf:topic ?MutationSpecification }
```

Part III: Case Study with Lipid Classification

LiPrO: Ontology of Lipids

- Hong-Sang Low, Christopher J.O. Baker, Alexander Garcia and Markus R. Wenk
- The starting point of our research on lipid classification.
- Based on the de facto standard LIPID MAPS hierarchy.
- Hierarchy: *Epoxyeicosatrienoic Acid is-a Eicosanoid is-a Fatty Acyl is-a Lipid*
- Lipid classes are defined via coarse descriptions of molecular structures in terms of participating molecular substructures – **functional groups**.
- “A molecule is an *Epoxyeicosatrienoic Acid* if and only if it has 3 *Alkenyl groups*, 1 *Primary Acyl chain*, at least one *Epoxy* and at least one *Carboxylic Acid*”
- (hasProperPart **exactly** 1 Primary_Acyl_Chain) **and**
(hasProperPart **some** Epoxy) **and**
(hasProperPart **some** Carboxylic_Acid) **and**
(hasProperPart **exactly** 3 Alkenyl_Group)

Automatic Ontology-based Lipid Classification

- Express a description of a molecule in terms of identified participating functional groups as an OWL class:
 - (hasProperPart **exactly** 1 Primary_Acyl_Chain) **and**
 - (hasProperPart **exactly** 1 Carboxylic_Acid) **and**
 - (hasProperPart **exactly** 4 Alkyl_Chain) **and**
 -
- Use an OWL reasoner to test if this class is a subclass of a LiPrO class.
- Packaged as a SADI service: lipid description with *hasProperPart* and instances of functional groups → predicted lipid class.
- Potentially, can be used (1) to classify lipids, possibly new, identified in high-throughput experiments or (2) to validate existing manual classifications.
- Can be much more useful in conjunction with some software that can detect functional groups in low-level chemical structure descriptions.

Functional Group Annotator

- Created by Leonid Chepelev and Michel Dumontier, Carleton University
- Takes a SMILES description of a molecule as input:
“CCCCC[C@H](O)/C=C/[C@H]1[C@H](O)C[C@H](O)[C@@H]1CC(=O)CCCCC(=O)O”
- Identifies substructures that are functional groups:
4 x Alkyl_Chain, 1 x Primary_Acyl_Chain, 1 x Carboxylic_Acid, ...
- Packaged as a SADI service: SMILES description of a molecule → lipid description with *hasProperPart* and instances of functional groups
- Ready for programming-free integration with the lipid classification service: the use of LiPrO ensures semantic interoperability

Main Use Case: Integration of FG Annotator and Classifier

```
SELECT ?liProClass ?LIPIDMAPSClass
FROM <http://unbsj.biordf.net/lipids/service-data/LMFA03010001.rdf>
WHERE
{
  #          annotator service    +    classifier service
  # molecule structure -> functional groups -> LiPr0 lipid class

  <http://semanticscience.org/LMFA03010001> lco:lipidHasLiProClass ?liProClass .

  # LiPr0 lipid class -> LIPID MAPS lipid class
  <http://semanticscience.org/LMFA03010001> lmo:lipidHasLMClass ?LIPIDMAPSClass .
}
```

Once our software is in the SADI service form, **the integration with other SADI services comes for free.**

Use Case 2: Integration with Reference Literature

- We have identified a lipid. What is known about it or similar lipids in the literature?
- Need a semantic index: lipid or lipid type name → publications mentioning it.
- Simple text-mining program identifies lipid mentions in texts and maps them to LiPrO classes. Also packaged as a SADI service that annotates a document URI with lipid classes – just for installation-free use.
- Semantic annotation results are accumulated in an RDF triplestore.
- Separate service queries the triplestore for papers associated with a lipid class and its subclasses. Provides predicate *isReferredToBy* = *inverse(isAbout)*.

SHARE Query for Use Case 2

```
SELECT ?liProClass ?LIPIDMAPSClass ?document
FROM <http://unbsj.biordf.net/lipids/service-data/LMFA03010001.rdf>
WHERE
{
  # Same as in the previous query:
  <http://semanticscience.org/LMFA03010001> lco:lipidHasLiProClass ?liProClass .
  <http://semanticscience.org/LMFA03010001> lmo:lipidHasLMClass ?LIPIDMAPSClass .

  # LiPr0 lipid class -> literature references
  ?liProClass sio:isReferredToBy ?document .
}
```

Programming-free zero-cost integration. **The initial effort to package our programs as SADI services is amortised.**

Use Case 3: Connection to Proteins

- A biologist wants to look at proteins related to the identified lipid classes via metabolic or signalling pathways.
- LIPID MAPS provides the mapping. We wrap it as a SADI service:
top-level lipid class → UniProt ID.
- Had to write another service to map a LIPID MAPS class to its top-level superclass (providing *rdfs:subClassOf*).

SHARE Query for Use Case 3

```
SELECT ?liProClass ?LIPIDMAPSClass ?LMCategory ?protein
FROM <http://unbsj.biordf.net/lipids/service-data/LMFA03010001.rdf>
WHERE
{
  # Same as in the previous queries:
  <http://semanticscience.org/LMFA03010001> lco:lipidHasLiProClass ?liProClass .
  <http://semanticscience.org/LMFA03010001> lmo:lipidHasLMClass ?LIPIDMAPSClass .

  # LIPID MAPS lipid class -> top level LIPID MAPS lipid class
  ?LIPIDMAPSClass rdfs:subClassOf ?LMCategory .

  # top level LIPID MAPS lipid class -> protein URI
  ?LMCategory sio:isRelatedTo ?protein .
}
```

Five services are automatically combined to do the job: annotator → classifier → LiPrO to LM → LM top level → lipid-protein map. The user only deals with a declarative query.

Conclusions and Future Work

- Our experiences can be transferred to many use scenarios, at least in bioinformatics.
- More case studies are needed to cover more typical situations, so that first external adopters can just copy existing solutions. After the exploratory phase, our experience will be systematised and published as **best practices/patterns/recipes**.
- Our case studies have also been helpful for identifying the soft spots in the SADI infrastructure and the framework itself \Rightarrow **guiding for future development of SADI**.
- Work in progress: integrating experimental fish toxicology data into the SADI “ecosystem”. Hopefully, another coherent case study.
- Clients other than SPARQL query interfaces:
 - **Virtual RDF browsing** with Sentient Knowledge Explorer (IO Informatics)
 - **Computer-aided workflow construction** with Taverna (Manchester University).