RAP – Report

RAP is a software package for parsing, querying, manipulating, serializing and serving RDF models.
RAP - RDF API for PHP, a semantic Web toolkit for PHP developers. It started, at the Freie Universität Berlin in 2002, as an open source project and has been extended with internal and external code contributions since then.
In the latest releases it includes:

- a statement-centric API for manipulating RDF graphs as a set of statements
- a resource-centric API for manipulating RDF graphs as a set of resources
- integrated RDF/XML, N3 and N-TRIPLE parsers
- integrated RDF/XML, N3 and N-TRIPLE serializers
- in-memory or database model storage
- support for the RDQL query language
- an inference engine supporting RDF-Schema reasoning and some OWL entailments
- an RDF server providing similar functionality as the Joseki RDF server
- a graphical user-interface for managing database-backed RDF models
- support for common vocabularies

The terms for using RAP are at GNU LESSER GENERAL PUBLIC LICENSE (http://www.gnu.org/copyleft/lesser.txt).
It can be downloaded from http://sourceforge.net/projects/rdfapi-php/

RDF graphs can be manipulated in two different programming interfaces, using RAP:

- **Model API** allows manipulating RDF graph as a set of statements. It supports adding, deleting, and replacing statements inside a model as well as adding entire models.
- **ResModel API** allows manipulating RDF graph as a set of resources.

Model API:

Implemented in four different ways

- MemModel - Model storing its RDF graph in memory. MemModel is fast, but doesn't support inference.
- DbModel - Model storing its RDF graph in a relational database. No inference support.
- InfModelF - Forward-chaining inference model storing its base graph and inferred triples in memory.
- InfModelB - Backward-chaining inference model storing its base graph in memory and creating inferred triples on the fly.

ResModel API:

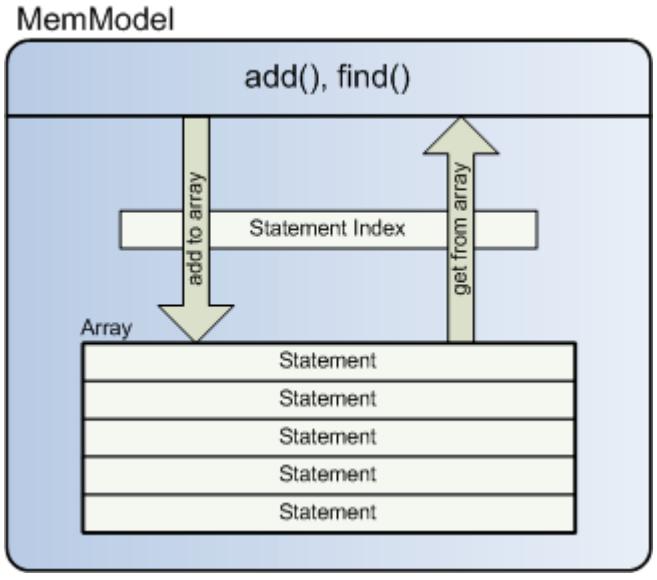
It is similar to the Jena Model API. It is implemented on top of the Model API and is only providing a resource-centric view of this model.

Two implementations of the ResModel API:

- ResModel - Basic implementaion fo the ResModel API
- OntModel - OntModel provides a ResModel implementation extended with RDF-Schema specific methods like hasSuperProperty(), addDomain() and listInstances().

Model API:

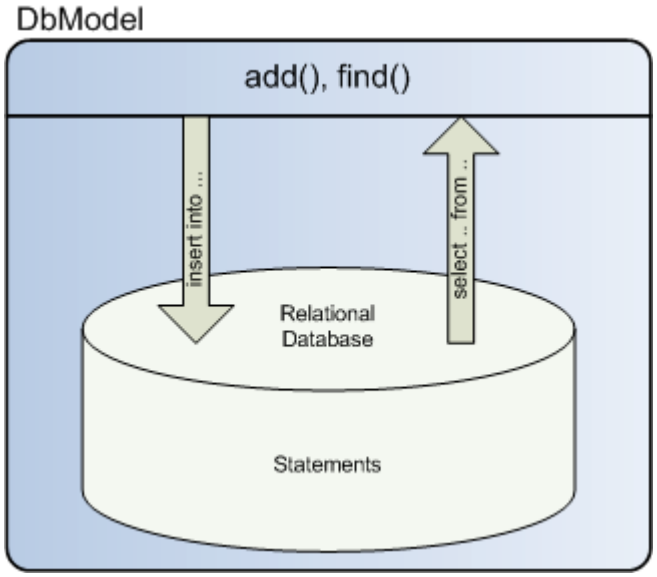
- MemModel



In this implementation, the statements are stored in an array in the system memory. The array gets appended with the new statements. It's fastest among all the Model API implementations.

- The DBModel :

  This implementation stores statements in a relational database. The core of RAP's database backend is the classes DbStore and DbModel. The former represents all models stored in a database, whereas the latter provides methods for manipulating these models.

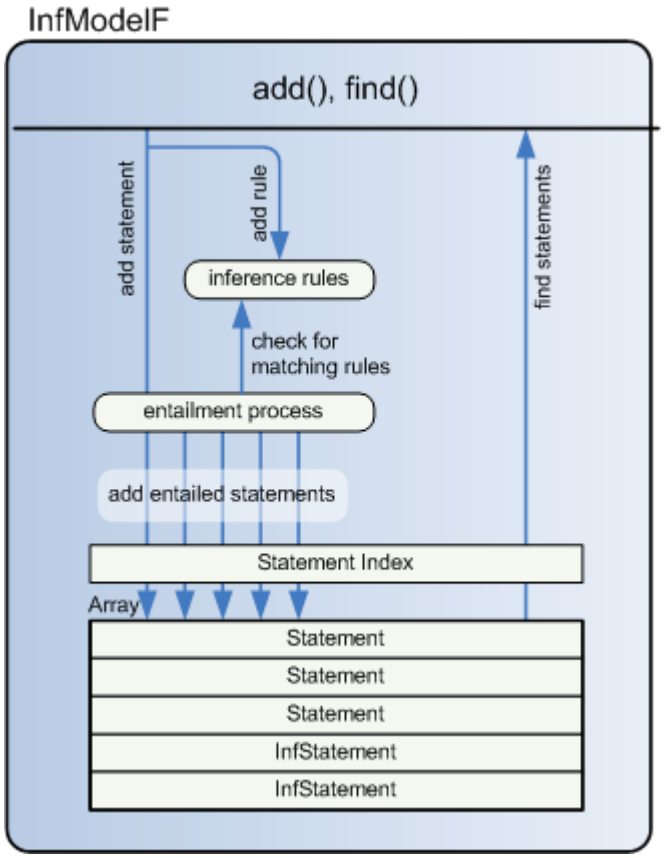


- The InfModelF

  It uses forward chaining inference algorithm; when a new statement from RDFS or OWL namespace is added to the model, a corresponding InfRule is added to the model's rule-base. Each rule has a trigger and an entailment. The statement which matches the trigger of an InfRule is added to the model, the entailment will be recursively computed until no more rule-triggers matches the statement or statements inferred from it. In this way, the base statement and all inferred statements are added to the model.

  But find operations are very fast, because the find method just looks into the statement index and return the matching statements, including inferred statements.

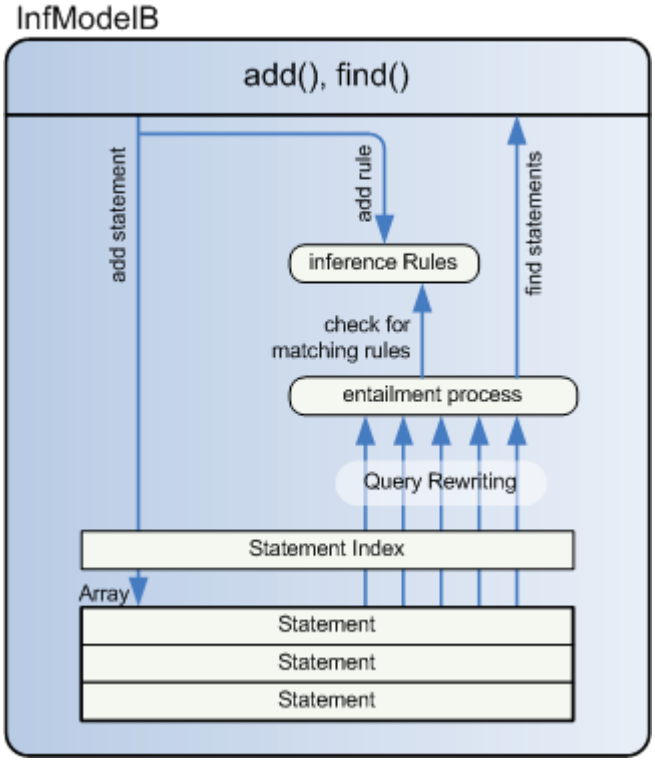
  It's advised to use if your model doesn't change much and querying model is a lot.



- InfModelB

  It uses a backward chaining inference algorithm. No inferences are done when a new statement is added to the model. During querying, the inferences are done. Firstly, find-pattern is executed and is checked against inference-rules index. If any matches are found, the find-pattern is rewritten and new search is done. The new statements are inferred and added to the result. The iteration process repeats until there are no rules to produce matching statements.

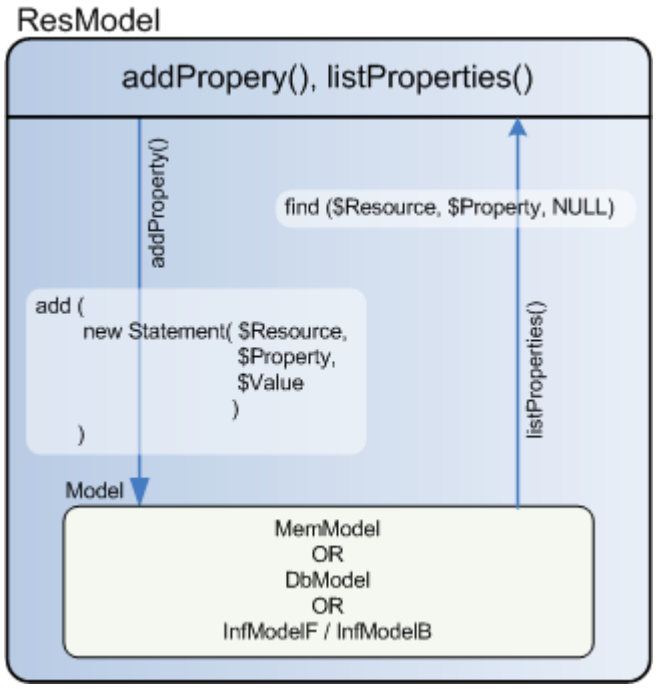
  It is advised to use if model has lots of changes and executing not too many queries against the model.



ResModel API:

- ResModel

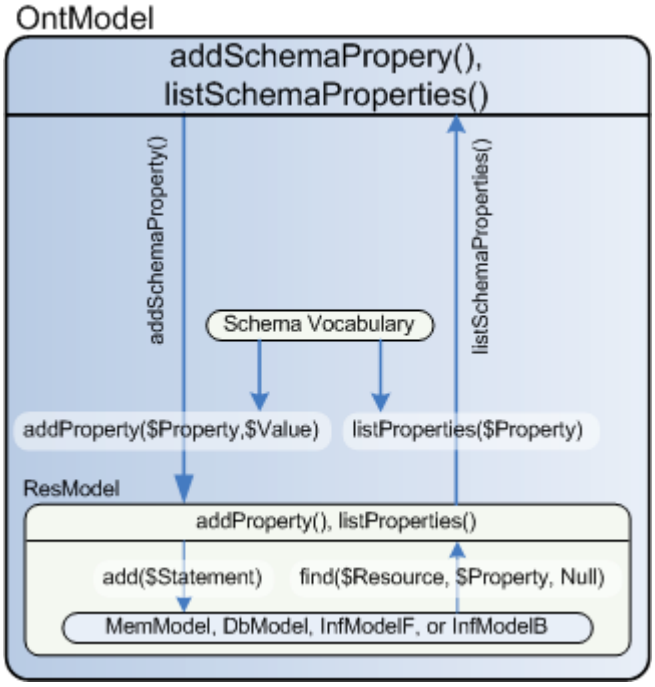
  It provides a resource centric view on an underlying Model. It represents and RDF graphs as a set of resources having properties, similar to the Jena Model API. It is implemented on top of the Model API. Each ResModel method call is translated into a find()-, add()-, or remove()- call to the underlying model directly. ResModel API also supports for special resources like rdf:containers and rdf:collections.



- OntModel

  OntModel API provides a ResModel implementation extended with RDF-Schema specific methods like addSubClass(), listSubClasses(),hadSuperProperty(), addDomain(), and listInstances().

  OntModel just provides a more convenient API for working with RDF-S models, but doesn't do any inference by itself. It is achieved by combining OntModel with underlying InfModelF or InfModelB.



Example of ModelAPI:

Download the most recent version of RDF API for PHP from http://sourceforge.net/projects/rdfapi-php/ and unzip the package.

```
define("RDFAPI_INCLUDE_DIR", "C:/Apache/htdocs/rdf_api/api/");
include(RDFAPI_INCLUDE_DIR . "RdfAPI.php");
```

Place this in the php script to include all RAP classes.

Consider the following statement written in N-Triple notation:
<http://www.example.org/someDocument.html>
<http://www.purl.org/dc/elements/1.1/creator> "Radoslaw Oldakowski"

In order to represent this statement in RAP we first create its components (subject and predicate - both are resources indicated by URIs)

$someDoc = new Resource ("http://www.example.org/someDocument.html");
$creator = new Resource ("http://www.purl.org/dc/elements/1.1/creator");

and then pass them together with the third component (object Literal) to the constructor function:

$statement1 = new Statement ($someDoc, $creator, new Literal ("Radoslaw Oldakowski"));

To generate MemModel :

$model1 = ModelFactory::getDefaultModel();

Add statement to the model:

```
$model1->add($statement1);
```

More models can be added while generating a model like:

$model2 = ModelFactory::getDefaultModel()
$model2->add(new Statement($someDoc, new Resource("http://www.example.org/myVocabulary/title"), new Literal("RAP tutorial")));
$model2->add(new Statement($someDoc, new Resource("http://www.example.org/myVocabulary/language"), new Literal("English")));

And add second model to the first model

```
$model1->addModel($model2);
```

The size of the model gives 3 statements that is both model1 and model2

```
echo "\$model1 contains " .$model1->size() ." statements";
```

Printing the model to output

// Output $model1 as HTML table
echo "Output the MemModel as HTML table: <p>";
$model1->writeAsHtmlTable(); //Method prints the model as HTML table

// Output the string serialization of $model1
echo "Output the plain text serialization of the MemModel: <p>";
echo $model1->toStringIncludingTriples(); //creates a plain text serialization of the model

// Output the RDF/XML serialization of $model1
echo "Output the RDF/XML serialization of the MemModel: <p>";
echo $model1->writeAsHtml(); //Serialize the model to RDF/XML

Saving - model Serialization to File using saveAs function.

$model1->saveAs("model1.rdf", "rdf");
$model1->saveAs("model1.n3", "n3");

Similarly loading RDF model from a file we us load()

The literal "Radoslaw Oldakowski" is replaced, being an object in the first statement, with a Blank Node. Additional statements describing this B-Node are made. To do this, first create a new BlankNode allowing the identifier to be automatically generated for in MemModel $model1:

$bNode = new BlankNode($model1);

Next using replace(), to exchange the objects.

```
$model1->replace(NULL, NULL, new Literal("Radoslaw Oldakowski"),
$bNode);
```

This will search for all statements having an object equal to the passed literal and replace this literal with the value of the fourth parameter ($bNode). Subsequently new statements are added describing the Blank Node using RAP's pre-defined vCard vocabulary ($VCARD_FN, $VCARD_EMAIL):

include(RDFAPI_INCLUDE_DIR . "vocabulary/VCARD.php");
$model1->add(new Statement($bNode, $VCARD_FN, new Literal("Radoslaw Oldakowski")));
$model1->add(new Statement($bNode, $VCARD_EMAIL, new Literal("radol@gmx.de")));

Typed Literals,

Add statement to the model $model1 describing the age of person represented by the blank node.

```
$age = new Literal("26"); //creating literal object

$age->setDatatype("http://www.w3.org/TR/xmlschema-2/integer");
//setting the datatype explicitly

$model1->add(new Statement ($bNode, new
Resource("http://www.example.org/myVocabulary/age"), $age))
```

Language of a literal can be specified by calling the method setLanguage() or by passing the language string as second parameter to the constructor method on an object Literal.

To traverse the model one by one, getStatementIterator() is used.

```
$it = $model2->getStatementIterator();//instantiation
```

Once instanced, the Statement Iterator can be ordered to return the current (current()), next (next()), or previous (previous()) statement. It can also move to a desired position (moveFirst(), moveLast(), moveTo()).

```
while ($it->hasNext()) {
      $statement = $it->next();
      echo "Statement number: " . $it->getCurrentPosition() . "<BR>";
      echo "Subject: " . $statement->getLabelSubject() . "<BR>";
      echo "Predicate: " . $statement->getLabelPredicate() . "<BR>";
      echo "Object: " . $statement->getLabelObject() . "<P>";
}

Above, the iterator is used to output the label of the subject,
predicate and object of each statement in $model2.
```

To models can be compared using equals().Methods containsAny() and containsAll() return a corresponding Boolean value about statements shared by two models being compared. Furthermore, methods unite(), subtract() or intersect(), which will return a new MemModel respectively.

Querying a model

In $model1, the statements having subject http://www.example.org/someDocument.html are searched using find() and pass the object Resource $somedoc representing the particular subject as parameter.

$result = $model1->find($someDoc, NULL, NULL);
$result->writeAsHtmlTable();

Build indices for better performance.

$model1 = ModelFactory::getDefaultModel();
$model1->index(IND_SPO); // IND_SP, IND_SO respectively or NO_INDEX to delete all indices.

Reification is done using reify() method.

```
$reified = $model2->reify();
$reified->writeAsHtmlTable();
```

Close() is used to terminate any model.

Example of ResModel API:

In RAP, the classes representing resources, properties and literals are called ResResource, ResProperty and ResLiteral.

```
//change the RDFAPI_INCLUDE_DIR to your local settings
define("RDFAPI_INCLUDE_DIR", "C:/!htdocs/rdfapi-php/api/");
include(RDFAPI_INCLUDE_DIR . "RdfAPI.php");

// Some definitions
define('VCARD_NS', 'http://www.w3.org/2001/vcard-rdf/3.0#');
$personURI = "http://somewhere/JohnSmith";
$fullName = "John Smith";

// Create an empty Model
$model = ModelFactory::getResModel(MEMMODEL); //underlying memmodel

// Create the resources
$fullNameLiteral = $model->createLiteral($fullName);
$johnSmith = $model->createResource($personURI);
$vcard_FN= $model->createProperty(VCARD_NS.'FN');
$vcard_NICKNAME= $model->createProperty(VCARD_NS.'NICKNAME');

// Add the property
$johnSmith->addProperty($vcard_FN, $fullNameLiteral);

// Retrieve the John Smith vcard resource from the model
$vCard = $model->createResource($personURI);

// Retrieve the value of the FN property
$statement = $vCard->getProperty($vcard_FN);
$value = $statement->getObject();

// Add two nickname properties to vcard
$literal1 = $model->createLiteral("Smithy");
$literal2 = $model->createLiteral("Adman");
$vCard->addProperty($vcard_NICKNAME, $literal1);
$vCard->addProperty($vcard_NICKNAME, $literal2);
```

//card resource has only one vcard:FN property. RDF permits a resource to have several properties of the same name Calling $vcard->getProperty(VCARD.NICKNAME) will return one of the values, but it is indeterminate which one. So if it is possible that a property might occur more than once, then the $resource->listProperties($property) method has to be used to get an array containing all statements with this resource and property

```
// List the nicknames
echo '<b>Known nicknames for '.$fullNameLiteral->getLabel().':</b><BR>';
foreach ($vCard->listProperties($vcard_NICKNAME) as $currentResource)
{
   echo $currentResource->getLabelObject().'<BR>';
};
```

Querying a Model:

$model->listSubjects() returns an iterator over all resources that have properties, *ie* are the subject of some statement. $model->listSubjectsWithProperty($property, $value) will return an iterator over all the resources which have a property $property with value $value.

```
// Iterate over all vcards which having FN property
$iter = $model->listSubjectsWithProperty(new ResResource(VCARD_NS.'FN'));
for ($iter->rewind(); $iter->valid(); $iter->next())
{
   $currentResource=$iter->current();
};

// Create a bag
$bag_smiths = $model->createBag();

$beckySmith = $model->createResource('http://somewhere/BeckySmith');
$beckySmithFN = $model->createLiteral('Becky Smith');
$beckySmith->addProperty($vcard_FN,$beckySmithFN );

// Add persons to bag
$bag_smiths->add($beckySmith);
$bag_smiths->add($johnSmith);
```

Serialize this Model, it gives:

```
<rdf:Description rdf:nodeID="A3">
   <rdf:type rdf:resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag'/>
   <rdf:_1 rdf:resource='http://somewhere/BeckySmith'/>
   <rdf:_2 rdf:resource='http://somewhere/JohnSmith'/>
</rdf:Description>
```

which represents the Bag resource.

```
// Print out the full names of the members of the bag
echo '<BR><BR><b>Print out the full names of the members of the bag:</b></BR>';
foreach ($bag_smiths->getMembers() as $resResource)
{
```

```
    // Retrieve the value of the FN property
    $statement = $resResource->getProperty($vcard_FN);
    echo $statement->getLabelObject().'<BR>';
};
```

The RAP ResContainer classes currently ensure that the the list of ordinal properties starts with rdf:_1 and is contiguous. The RDFCore WG have relaxed this constrain, so RAPS's container implementation might follow in the future.

```
echo '<BR><BR>All Statements as HTML table';
$model->writeAsHTMLTable();
```

Write the model as HTML table.

Conclusion:

RAP is easy to install and work with RDF models. Smaller implementations can be done using ModelAPI and higher level applications are dealt with ResModelAPI based on the user requirements. SPARQL is supported and easy to implement. RDQL which is part of RAP model and a W3 standard like SPARQL is effective over ResModelAPI. It gives flexibility to user by giving choice of database to implement i.e. using MySQL, MSACCESS, ODBC or other databases. Official link for the RAP tool is http://www4.wiwiss.fu-berlin.de/bizer/rdfapi/

Note:

While running the PHP files, some points to be considered as the latest rdf-api package downloaded from server has some bugs. We fixed them while working. Ereg depreciation error: when encountered such errors using the PHP 5.3 or higher version, http://devthought.com/2009/06/09/fix-ereg-is-deprecated-errors-in-php-53/ refer this to fix the problem.

We would be happy to answer any questions related to using the tool or doubts.

-Lakshmi Narayana Gupta Kollepara – lkollepara1@student.gsu.edu

-Srujana Gorge – sgorge1@student.gsu.edu