Journal of Universal Computer Science, vol. 15, no. 9 (2009), 1970-1990
submitted: 15/8/08, accepted: 24/4/09, appeared: 1/5/09 © J.UCS

Integrating Semantic Web and Object-Oriented Programming for Cooperative Design

Po-Huan Chiu
(Institute of Information Management, National Chiao Tung University, Hsinchu, Taiwan
bohachu@gmail.com)

Chi-Chun Lo
(Institute of Information Management, National Chiao Tung University, Hsinchu, Taiwan
cclo@faculty.nctu.edu.tw)

Kuo-Ming Chao
(DSM Research Group, Department of Computing and the Digital Environment
Coventry University, Coventry, United Kingdom
csx240@coventry.ac.uk)

Abstract: Object-oriented programming (OOP) is a mainstream paradigm for engineering design software tool development. An emerging requirement is the introduction of semantics to achieve heterogeneous information sharing, but many challenges exist. Examples include using object methods to manipulate an RDF data, automatically converting data into RDF format, and supporting various programming languages. In addition, limitations to description capabilities for relationships among object-oriented classes exceed those of RDF, thus hindering direct mapping between object-oriented and Semantic Web classes. Our proposed semantic object framework (SOF) combines object-oriented design and Semantic Web features. SOF utilizes embedded comments in source code to describe semantic relationships between classes and attributes. We use a mobile phone design case study to illustrate how the proposed system operates.

Keywords: Semantic Web, Object-oriented programming, Cooperative design
Categories: D.1.5, D.2.2, D.2.13

1 Introduction

As an evolving extension of the World Wide Web, the Semantic Web [Bemers-Lee, 01] uses semantic relationships among data to perform automated sharing and processing functions. Applications focus on process automation, data searches, data integration, and data reuse. Resource description frameworks (RDFs) [Lassila, 99] are used to represent Semantic Web data models. A basic RDF document contains statements consisting of a subject, predicate, and object. Engineers use this powerful representation tool to design processes and products to maximize knowledge and information sharing. Most existing engineering design tools are based on an object-oriented (O-O) paradigm, but the mismatch between O-O and the Semantic Web hinders the seamless integration of current design tools into Semantic Web based data models. Most software developers utilize the object-oriented programming (OOP) software design paradigm, but OOP is clearly unsuitable for processing Semantic Web data [Koide, 05] [Koide, 06].

The most widely used function-dividing architecture for designing OOP classes is the model-view-controller (MVC) [Krasner, 88]. There are several object-relational mapping tools that can convert model objects associated with model classes into record formats for relational databases. Since RDF utilizes triple-oriented statements for data formatting, it differs significantly from MVC model classes. Furthermore, object-oriented classes cannot be used to describe semantic relationships among class attributes, thus making the task of converting model objects into RDF format for semantic queries more complex. Engineers have learned that the greater the amount of existing data requiring conversion into triple-oriented format, the greater the challenges in terms of performance and costs.

O-O technology hides semantic relationships in source code function data. Pure O-O technologies do not support data reasoning or inference as in Semantic Web technology. It is also hard for O-O to handle heterogonous data sources without Semantic Web technology. Object-oriented programming is mature, and many design patterns exist that can help programmers write reusable source code. The Semantic Web can publish information to the Internet as reusable data sources. It is a powerful means for integrating benefits from O-O (programmer-friendly coding style) and Semantic Web technology (machine readable web pages).

In this paper we will describe a semantic object framework (SOF) for integrating O-O design with Semantic Web features. Our main goals are simplifying the tasks of (a) publishing model objects in RDF format via object-oriented design methods, and (b) making heterogeneous data queries in accordance with semantic relationships between classes and attributes. We use a mobile phone design case study to illustrate how the proposed system operates.

2 Survey

Before providing details of our SOF proposal, we will describe four Semantic Web solutions currently being used by developers and briefly review their positive and negative features.

2.1 Jena

Currently the most popular solution, Jena uses triple-oriented APIs to read/write and query RDF data [McBride, 02][Carroll, 04]. Jena's main advantages are its full support for low-level RDF operations and the fact that it is already in wide use, thus simplifying the task of obtaining sample code. Owing to the current lack of OOP integration, each operational step must be described in detail during its use phase.

2.2 ActiveRDF

This RDF object-oriented API is based on the Ruby language [Oren, 06] [Oren, 07]. To perform the task of abstracting triple-oriented APIs, it uses O-O methods to manipulate RDF documents so as to simplify low-level API calling. Due to implementation limitations, this solution does not support the use of more than one programming language.

2.3 D2R

D2R directly converts relational database records to RDF format in order to facilitate RDF read/write and query functions [Bizer, 03] [Bizer, 04]. Since the manipulated target is a database, D2R can be applied to any programming language and automatically perform format conversion (relieving programmers of this task) as long as the mapping relationship between database tables and RDF is clearly specified. Having a database as a manipulated target means that D2R does not support object-oriented encapsulation, thereby eliminating any possibility of data manipulation using objects.

2.4 EClass

This solution changes Java syntax to embed semantic descriptions into source code. [Liu, 04] [Liu, 07]. EClass allows developers to define semantic relationships between attributes. However, an obstacle occurs when changing a widely used programming syntax, since syntax definitions affect existing programming tools such as compilers and virtual machines. Current programming tools need to be rewritten to support new syntaxes. Furthermore, the EClass solution currently lacks a query function for heterogeneous model objects.

3 Semantic Web Development Problems

As shown in Table 1, there are at least seven problems associated with the integration of Semantic Web and O-O design:

Problem	Jena	Active RDF	D2R	EClass	SOF
Use object methods to manipulate RDFs.	X	O	X	O	O
Automatically convert data into RDF format.	X	X	O	O	O
Support various programming languages.	X	X	O	X	O
Use statements to describe class and attribute semantics.	X	X	X	O	O
Maintain semantic description files and class definition synchronization.	X	X	X	O	O
Support inheritance queries and heterogeneous data between classes and attributes.	X	X	X	X	O
Verify consistency in data and semantics.	X	X	X	X	O

Table 1: A comparison of functions for five Semantic Web development schemes. X denotes "unsolvable" and O "solvable".

3.1 Using object methods to manipulate RDFs

Even though low-level RDF APIs provide complete RDF read/write and query functions, developers lack tools for utilizing objects to manipulate RDF data. As a result, development durations are longer, program codes relatively larger, and maintenance more difficult. Our proposed SOF system uses O-O design to abstract RDF APIs to support the writing of program codes. Specifically, the system supports the use of O-O APIs for making queries, with corresponding query results returned in the form of model objects.

3.2 Automatically converting data into RDF format

Although some RDF APIs are capable of storing triple-oriented data for semantic query purposes, developers must convert model objects into triple-oriented format [Carroll, 03]—a detailed and time-consuming task. Thus, any development architecture capable of automatically converting model objects into RDF format will save developers significant amounts of time and effort. In addition, we have included an embedded web server that allows third-party software programs to use HTTP protocol to read RDF format data.

3.3 Supporting various programming languages

Instead of binding SOF syntax to a specific object-oriented programming language, we adopted a strategy of utilizing comments that describe class and attribute semantics to support the use of the SOF parser (with minimum modifications) with multiple programming languages [Kramer, 99] [Leslie, 02]. Accordingly, programmers will only be required to learn SOF in order to develop applications.

3.4 Using statements to describe class and attribute semantics

The most straightforward way to combine Semantic Web and O-O design features is to describe class or attribute semantics, preferably at the same time that classes are defined. However, defining class and attribute semantics usually requires modifying programming language syntax. To address this modification issue without adversely affecting the original programming syntax, our proposed SOF system allows for embedded comments that support the limited use of RDF and OWL [McGuinness, 04] syntaxes.

3.5 Maintaining semantic description files and class definition synchronization

Some Semantic Web implementation solutions provide independent semantic description files that further modify relationships in existing data. This requires momentarily maintaining synchronous updates between files to prevent inconsistencies. Note that program API document and program code files are mutually independent and description document updates are frequently overlooked, resulting in obsolete and erroneous descriptions. JavaDoc uses embedded comments to prevent inconsistencies between API documents and program codes, which makes it easier for programmers to maintain consistency. Our SOF solution is to apply similar principles to maintain program code and semantic description synchronization.

3.6 Supporting inheritance queries and heterogeneous data between classes and attributes

Inconsistencies in column names across different databases are common (e.g., database A may use the term “Email” and database B “email”). To perform consistent queries involving all e-mails stored in two databases, the semantics of both terms must be clearly defined so that computers recognize them as equal. No architecture currently exists for defining semantic relationships between classes and attributes in OOP codes that allows a system to automatically acknowledge different attribute names with identical meanings. Problems also arise when performing unified queries of heterogeneous data sources. Our proposed SOF system allows for the utilization of comments to maintain an inheritance relationship between attributes, and lets developers make unified queries of heterogeneous model objects.

3.7 Verifying consistency in data and semantics

Conflicts can occur between model objects and semantics. For instance, assigning an Email value to one unique account in an account management system can result in a later conflict when two accounts have the same Email value. Our proposed SOF system provides APIs for querying objects that developers can use to make semantic consistency checks.

4 SOF Architecture

4.1 SOF modules

The five modules of our SOF architecture that address the above-listed problems are illustrated in Figure 1. The SOF data adapter reads data sources (i.e., CSV format files [Shafranovich, 05], database records, or proprietary data APIs) for conversion into model objects. Model objects that represent SOF data adapter output include all data content (e.g., attribute values). Those objects later serve as input parameters for the SOF query engine and SOF RDF generator.

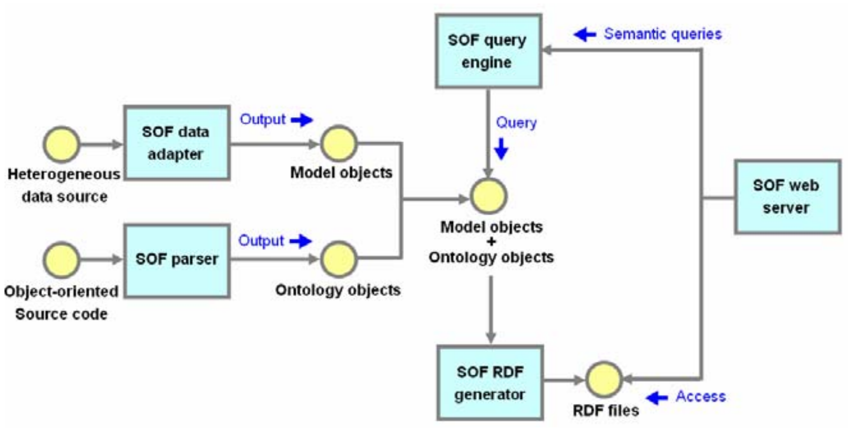


Figure 1: Five primary SOF modules

As its name implies, the function of the SOF parser is to parse SOF statements from comment lines in source code for the purpose of generating ontology objects, which include all information about semantic relationships between classes and attributes. The parser supports several of the most popular O-O languages, using a syntax that overcomes comment and descriptor variation problems. Module output consists of ontology objects in which semantic class and attribute relationships are represented as objects. Ontology objects also serve as input parameters for the SOF RDF generator and SOF query engine.

The purpose of the SOF RDF generator module is to output model objects in RDF format so that third-party software programs can read RDF format data. Semantic relationships among model objects are recorded in the form of ontology objects that support RDF format file generation.

The SOF query engine module supports unified object-oriented API queries involving multiple heterogeneous data sources. Query results are presented as unified object arrays. Since returned model objects may be matched with different classes, APIs that are suitable for specific conversion types must be provided to address format conversion issues.

Finally, the SOF web server module provides an entry point for HTTP protocol so that third party programs can read RDF documents. Since our proposed SOF system utilizes dynamic conversion processes, all model object changes are updated to RDF documents in real time, thus eliminating data consistency concerns.

4.2 Module Design

4.2.1 Data adapter

The input terminal of this adapter is capable of handling several types of data sources. After performing model object format output conversions, object-oriented APIs are used to read and write model objects. The four SOF data adapters are a DatabaseAdapter for reading records via database APIs, an RdfAdapter for reading data files in RDF format, a GmailContactAdapter for reading address book data via

Gmail APIs, and a ThunderBirdContactAdapter for reading address books in ThunderBird data file format. Since these adapters are inherited from the SofDataAdapter class, they share common operation methods. Adapter output format is presented as MVC model objects, which generally provide operation methods for reading and writing object attributes.

4.2.2 Parser

We have included three SOF parsers: PythonSofParser for reading Python code [Van Rossum, 03] [Vrandecic, 05] [Babik, 06], JavaSofParser for reading Java code, and RdfSofParser for reading class semantics in RDF file format. Since they are all inherited from the SofParser class, program code sharing is supported. Ontology objects generated by the SOF parser contain semantic relationships between classes and attributes. If ontology and model objects are used concurrently, heterogeneous data source semantic queries [Prud'Hommeaux, 06] [Ying, 07] can be performed.

4.2.3 Query engine

Inputs consist of model and ontology objects. Our engine is capable of accepting query statements and outputting results in the form of model objects. The three SOF query engines are a FilterSofQueryEngine for conditionally filtering semantic queries, a ValidSofQueryEngine for querying model objects that coincide with semantic rules, and an InvalidSofQueryEngine for querying model objects associated with illegal semantics. Since all are inherited from SofQueryEngine, all output results are presented as model objects.

For results generated as model objects by the FilterSofQueryEngine, only those that match query conditions are listed. During a query, developers can input object arrays for various classes, meaning that query results can also include different object classes. Our proposed SOF system supports the use of APIs to obtain original model object class types; special processes can be used for different model object classes as necessary. For query results generated by the InvalidSofQueryEngine, model objects also include explanations for illegal objects—a useful tool for making corrections.

4.2.4 RDF generator

Generator inputs are model and ontology objects. The generator is capable of combining the two and outputting RDF strings. final RDF string output can be stored in file format and accessed by other HTTP applications via the SOF web server. Since strings are expressed in standard W3C format and include model object data content as well as ontology object semantic relationships.

5 Examples

5.1 Adding semantic relationships to classes and attributes

The address book data used in the following examples are supported by Gmail. Taking Python language as a specific example, our SOF approach is to add semantic relationships to classes and attributes when they are declared. Before making a unified query across various address books, a user must first define a class named “Contact” for sharing common attributes. From a semantics perspective, this class is inherited to GContact (Gmail Contact) and TContact (ThunderBird Contact).

```
class Contact(Model):
   partOfName=''
   partOfAddress=''
   #owl:InverseFunctionalProperty Contact_email
   email=''
   phoneNumber=''
   #Contact_officePhoneNumber rdfs:subClassOf Contact_phoneNumber
   officePhoneNumber=''
   #Contact_homePhoneNumber rdfs:subClassOf Contact_phoneNumber
   homePhoneNumber=''
   #Contact_mobilePhoneNumber rdfs:subClassOf Contact_phoneNumber
   mobilePhoneNumber=''
   #Contact_faxPhoneNumber rdfs:subClassOf Contact_phoneNumber
   faxPhoneNumber=''
```

According to the MVC design model, Contact class belongs to the Model data class, therefore class Contact(Model) is declared as representing a Contact inherited to the Model class.

The presentation meaning of the “partOfName” attribute is a contact person's name, which contains a surname/middle name/full name/nickname, etc. Here we allow partOfName to represent a full name or any name segment. If the semantics of any other attribute are inherited to partOfName, the attribute is used to identify one contact person’s name string.

In Python, the pound sign (#) designates a comment. Since SOF syntax is embedded in comments, any instance of ‘owl:’ or ‘rdfs:’ included in a comment means the statement is SOF-specific. For example, ‘#owl:InverseFunctionalProperty Contact_email’ utilizes OWL syntax to modify its semantics, meaning that Contact_email string values must be unique. This should not occur in cases where two different Contact objects have the same email attribute value. In situations where they have the same email string, our proposed SOF system identifies conflicting Contact objects and notifies programmers, who can apply various strategies to resolve the illegal semantics. OWL statements are helpful for programmers in terms of applying rich syntaxes to limit relationships between model objects.

E-mail attribute names differ across various applications. Examples in address book software programs include Email, email, mail, Mail, emailAddress, and EmailAddress—all with identical semantics. In order to display all attribute values for all emails across heterogeneous address books, all E-mail-related attributes must be inherited to Contact_email.

The next topic is the process through which GContact is inherited to well-defined Contact attributes.

```
#GContact rdfs:subClassOf Contact
class GContact(Model):
   #GContact_name rdfs:subClassOf Contact_partOfName
   name=''
   #GContact_email rdfs:subClassOf Contact_email
   email=''
```

#GContact_phone rdfs:subClassOf Contact_officePhoneNumber
#GContact_phone rdfs:subClassOf Contact_homePhoneNumber
phone=''
#GContact_mobile rdfs:subClassOf Contact_mobilePhoneNumber
mobile=''
#GContact_fax rdfs:subClassOf Contact_faxPhoneNumber
fax=''
company=''
title=''
#GContact_address rdfs:subClassOf Contact_partOfAddress
address=''

The representative meaning of "#GContact rdfs:subClassOf Contact" is that the GContact class is semantically inherited to the Contact class, therefore if any object query commands are used to query all Contact model objects, the GContact object inherited to the Contact class will remain within the scope of the queried targets. In a later section we will show that TContact is also semantically inherited to Contact. Accordingly, when developers want to query model objects from two different address books (e.g., Gmail or ThunderBird), SOF automatically recognizes that both GContact and TContact objects must be involved within the query scope if Contact class is the target being queried. In this manner, the goal of querying heterogeneous address books can be easily accomplished.

According to the comment line "#GContact_name rdfs:subClassOf Contact_partOfName," the name attribute in the GContact class is semantically inherited to the partOfName attribute of the Contact class. Thus, if developers specify the string value of the Contact_partOfName attribute that is being queried at a later time, our SOF system will also automatically query the string value of the GContact_name attribute.

GContact_phone refers to a multiple inheritance relationship. The attribute represented by GContact_phone can be a business or residence telephone. Since RDF syntax supports multiple inheritance relationships, SOF still allows for semantic multiple inheritance descriptions for classes or attributes. This is true even if the programming language (e.g., Java) does not support multiple inheritance relationships. Using GContact_phone as an example, regardless of whether a developer chooses Contact_officePhoneNumber or Contact_homePhoneNumber as a query target at a later time, SOF will always automatically query GContact_phone attributes.

5.2 Implementation Details

Sequence diagram (Figure 2) showing how SOF supports the automatic conversion of data into RDF format.

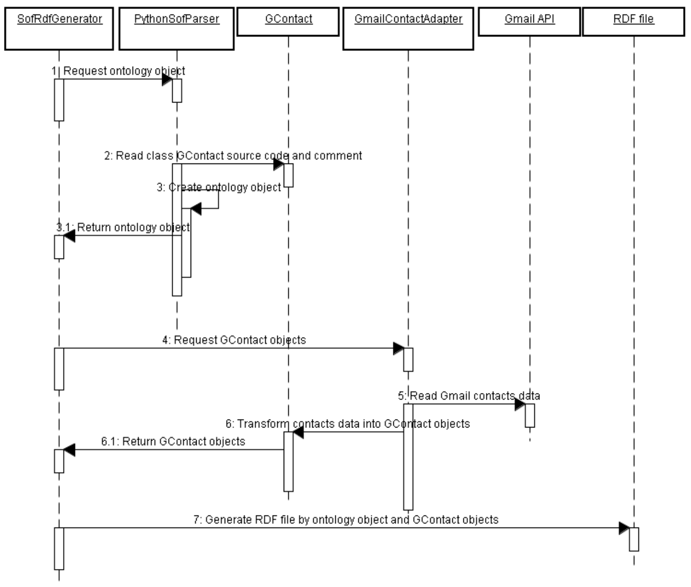

SofRdfGenerator
PythonSofParser
GContact
GmailContactAdapter
Gmail API
RDF file
1: Request ontology object
2: Read class GContact source code and comment
3: Create ontology object
3.1: Return ontology object
4: Request GContact objects
5: Read Gmail contacts data
6: Transform contacts data into GContact objects
6.1: Return GContact objects
7: Generate RDF file by ontology object and GContact objects

Figure 2: How SOF supports the automatic conversion of data into RDF format

We use a sequence diagram to help readers understand how SOF automatically generates RDF files from a data source. We use Gmail API as our data source for reading gmail contact information.

Request ontology object: SofRdfGenerator is responsible for initializing the RDF generation process. It sends initial requests to PythonSofParser and attempts to get ontology objects as return values.

Read class GContact source code and comment: To produce ontology objects, PythonSofParser needs to parse python source code containing GContact (Google contact) class definitions and semantic descriptions.

Create ontology object: Ontology objects are dynamically created by PythonSofParser and preserved in Python run-time memory. Semantic relationships (represented by ontology objects) are like a directed graph data structure.

Return ontology object: After transforming embedded comments to ontology objects, PythonSofParser returns them to SofRdfGenerator.

Request GContact objects: SofRdfGenerator needs two input parameters to generate RDF files—ontology objects and model objects such as GContact. GmailContactAdapter receives requests from SofRdfGenerator and tries to return GContact model objects.

Read Gmail contacts data: Google gmail provides a Google data API to read contact information from its distributed network storage. GmailContactAdapter needs to call the Google data API. GmailContactAdapter sends a user's account name and password to the Google data API; after authentication, it can read the user's contact data.

Transform contact data into GContact objects: GmailContactAdapter transforms data from a Google data API to a GContact object. Data field names are mapped one-to-one. It is easy to transform data values as strings in GContact objects.

Return GContact objects: GContact objects are returned to SofRdfGenerator.

Generate RDF file by ontology object and GContact objects: After SofRdfGenerator receives both ontology and GContact objects, it gets all necessary information for generating RDF schema and RDF data formatting. The Django framework for our development tool provides a template architecture to dynamically generate files in any format. SofRdfGenerator transforms ontology and GContact objects as string variables in a hashtable data structure, and then uses the Django template architecture to produce RDF files.

6 Case Study

Mobile phone design and manufacturing managers must work with component suppliers to create new products and systems. They must address such issues as component costs, compatibility, functionality, and capability. In this section we will discuss real and potential problems encountered in mobile phone design, show how our proposed SOF can be used to develop a mobile phone assisted design system (MPADS) to address them, and evaluate MPADS performance.

Mobile phone companies regularly manufacture and market multiple products concurrently. Product managers delay the need to design completely new mobile phones by referencing the component combinations of existing models—in other words, most successful designs can be reused and repackaged to create new phones with incremental specification changes. However, doing so raises challenges in terms of efficient information exchanges among independent design teams so as to achieve the greatest benefits from their different knowledge bases.

Here we will describe the case of a company using Excel files for purposes of documenting and sharing mobile phone specifications with design teams working in Taiwan, China, and Germany. According to current limitations, product managers wanting information for a specific component must manually open all Excel files and combine the required data into a new Excel spreadsheet. To support efficient knowledge sharing, we designed our proposed MPADS to produce efficient semantic queries without having to manually merge and edit files.

6.1 MPADS Goals

A mid-level mobile phone consists of between 50 and 60 components. During the design process, product managers must repeatedly perform design information queries based on previous experiences and product success. An efficient query system [Vega-Gorgojo, 08] allows product managers to make quick but informed decisions about new components and compositions. We therefore designed our proposed MPADS according to six goals: performing heterogeneous data queries; converting data to RDF format; converting component measurement units (e.g., speaker component dimensions) to fit query statements; analyzing mobile phone models and specifications based on required conditions; analyzing individual component specifications; and reviewing component defect reports.

During the mobile phone design process, a product manager will generally want to use the lowest price components that are sufficiently compatible. To accomplish this they must constantly perform data queries according to a complex mix of parameters. MPADS can help product managers perform such queries quickly and more efficiently than Excel files by using SOF development tools to define component classes and attributes in order to identify semantic relationships [Burger, 08] [Burkard, 08] [Valkeapaa, 08]. This process requires the conversion of Excel files (also referred to as comma separated values, or CSV) to model objects so that a programming language can directly read the information. Our SOF data adapter is capable of performing this reading/conversion task.

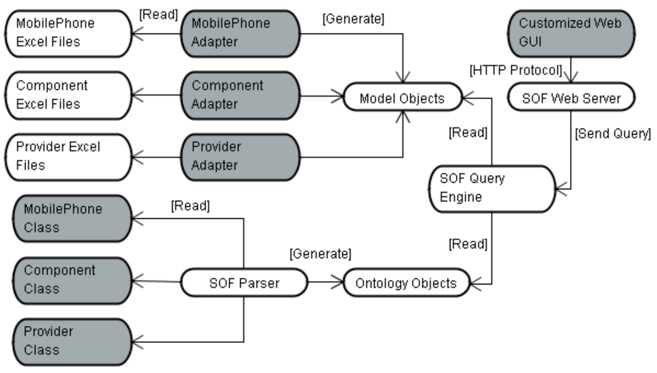


Figure 3: Primary MPADS modules

After creating classes and attributes from heterogeneous data sources, MPADS uses SOF syntax to define semantic relationships for further queries. Data adapters use Excel CSV records as input and generate model objects as output for semantic queries. Readable outputs require a Customized Web GUI to convert text strings from SOF Web Server output format to HTML table format to help product managers compare component attributes. Figure 3 shows modules requiring developer implementation (grey background) and modules provided by SOF without additional programming requirements (white background).

6.2 Cooperative Design

Our goal for MPADS is to help mobile phone designers working on a single sign on computer-supported cooperative system. We integrated MPADS with subversion (open source version control system) and mantis (open source issue tracking system) projects. The following Table 2 presents the cooperative design features of MPADS:

Feature	Description
1. Knowledge sharing and semantic querying.	Product managers can perform conditional semantic queries to reference previous mobile phone design documents. Product managers can upload new designs to MPADS for sharing.
2. Structured component database sharing.	Mobile phone component specifications are originally stored in Excel or Word files without structure. MPADS allows designers to store structured information on component attributes in databases for spec. sharing purposes.
3. Design document co-editing.	Design documents can be uploaded, shared, opened, and edited by multiple users.
4. Online discussion.	MPADS users can post questions or share opinions online. Replies are collected in thread form and emailed to participating users.
5. Access control for user groups.	Users are divided into different groups. Each group has flexible access control as determined by an administrator. Design specifications are categorized to assist with controlling access.
6. Task assignments.	Managers can divide large design tasks into several subtasks and assign them to different developers. Priorities and task statuses can be monitored online by team members.
7. Merge modifications by version control.	If there are multiple users editing the same document, the version control feature can be used to solve collision problems via the automatic or manual merging of modification results.

Table 2: Cooperative design features of MPADS

For example, there are multiple roles [Aqqal, 08] in mobile phone design processes. MPADS allows for collaboration among various roles as shown in the following Table 3:

Role	Description	Cooperative Design Feature
Sales team	Collects customer feedback and new feature requirements from mobile phone markets. Posts market feedback on MAPDS and discusses feedback online with product manager.	4,5,7
Product manager (PM)	Coordinates business and technical teams with help from MAPDS. Responsible for tracking progress for new design and providing design specifications. Can use MPADS to perform semantic queries for hardware or software components.	1,2,3,4,5,6,7
Man-machine interface (MMI) team	Responsible for designing high-level software applications.	1,3,4,5,6,7
Layer 1 team	Provides application programming interface (API) for MMI team to control hardware functions.	1,2,3,4,5,6,7
Baseband team	In charge of mobile phone hardware layout and physical components. Can upload hardware component images and specifications to MPADS for users to perform queries.	1,2,3,4,5,6,7

Table 3: Multiple roles in mobile phone design process

6.3 Flexibility Evaluation

We evaluated differences among Excel files, relational database management system (RDBM), and MPADS in terms of query flexibility (Table 4) and efficiency (Table 5). Regarding the first parameter, product managers generally specify component attribute values to perform conditional semantic queries. A drawback of Excel is the tendency for design teams to use different formats; this is especially true when those teams work in different countries, but it is not unusual among teams working for the same firm. As stated above, this requires the manual merging of query results into a new Excel datasheet, a time-consuming task. Developers who use RDBM cannot query heterogeneous data by simply applying SQL commands, since semantic relationships among database table fields require definitions.

MPADS allows for the easy processing of heterogeneous data by simplifying the task of defining semantic relationships for a body of data. As a result, product managers are only required to input single-line query commands to perform design information searches. MPADS automatically combines and presents search results in HTML.

	Excel	RDBM	MPADS
Query heterogeneous data.	O (manually)	X	O
Convert data to RDF format.	X	O (D2R)	O
Convert component measurement units to fit query statement.	O (manually)	X	O

Table 4: A comparison of Excel, RDBM, and MPADS in terms of conditional query flexibility. X, unsolvable; O, solvable

Regarding RDF format conversion, a D2R system is available for automatically converting database records stored in RDBM format into RDF. While D2R is a convenient tool, it does not allow developers to manipulate data in an O-O fashion; lack of integration with OOP programming is its most significant drawback. The absence of OOP translates into more time required for product development tasks. MPADS lets developers define semantic relationships between classes and attributes, convert Excel files into model objects, and use an SOF Web Server to publish output in RDF format for reading by third party applications. It is equipped with OOP to reduce coding efforts, thereby releasing developers from having to write additional code for conversion tasks.

In the next area of comparison, mobile phone component attributes are frequently expressed in different measurement units—for example, costs may be expressed in US dollars or Euros, dimensions may be expressed in millimeters or inches, and chip memory may be expressed in MB or KB. Data stored in Excel format must be converted manually; RDBM is also incapable of supporting automatic conversions for measurement units. Our proposed MPADS allows developers to define conversion formulas prior to performing queries. For example, Money class can be defined as

```
class Money:
  intAmount
  strMoneyType
```

Here intAmount represents quantity and strMoneyType a chosen currency. Using Usd, Eur or Gbp as Money subclasses, MPADS allows for value comparisons using a MoneyConverter class:

```
def getConverted(strSourceType,strTargetType,intAmount)
```

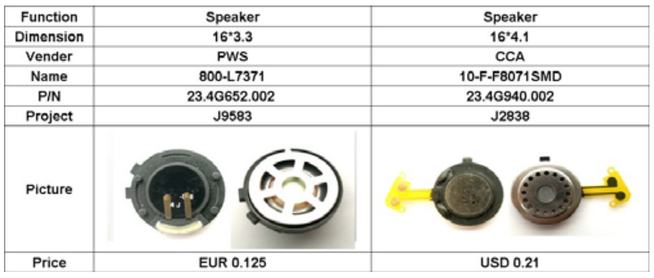
This method returns a converted currency quantity, strSourceType (representing the original currency type), intAmount (representing the original quantity), and strTargetType (representing the converted currency type). Once the MoneyConverter class is implemented, MPADS uses a SOF query command to perform a search—for example:

```
Speaker.objects.get(“price < Usd(0.22)”)
```

This query finds all speaker components costing less than $0.22 US, with prices for components manufactured in other countries automatically converted into a designated currency.

6.4 Efficiency Evaluation

Locating sources of less expensive components is a common product manager responsibility. An example of HTML query output is shown in Figure 4. Product managers can use this feature to compare component attributes from various suppliers by reading user-friendly HTML output. Efficiency comparisons for three related tasks are shown in Table 5. Note that RDBM was not considered, since SQL commands cannot be used to perform queries based on heterogeneous data.



Function	Speaker	Speaker
Dimension	16*3.3	16*4.1
Vender	PWS	CCA
Name	800-L7371	10-F-F8071SMD
P/N	23.4G652.002	23.4G940.002
Project	J9583	J2838
Picture		
Price	EUR 0.125	USD 0.21

Figure 4: Search results for speaker components priced below $0.22 USD

	Excel	MPADS
Analyze mobile phone models and specifications based on required conditions.	1,219.46 secs	37.69 secs
Analyze single component specifications.	97.94 secs	13.88 secs
Review component defect reports.	46.28 secs	11.42 secs

Table 5: Query efficiency comparisons between Excel and MPADS

Using Excel files to perform manual queries requires locating strings in existing datasheets and cutting-and-pasting all matching data to a new datasheet. To determine the time required to complete this task, we performed each example query 3 times to obtain an average speed for finding information on 22 existing mobile phones and 140 components. For tests involving Excel, time was measured from the first opening of an Excel file to the completion of a datasheet. For MPADS, time was measured from the inputting of query strings in a customized Web GUI to the complete loading of a HTML result page into a browser.

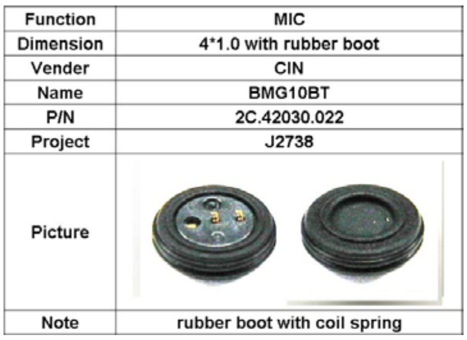
Our tests were based on the knowledge that product managers are frequently required to perform conditional queries and to check component attributes. For example, in order to design a mobile phone that highlights multimedia functionalities, a product manager will likely perform at least three conditional queries regarding display size, camera resolution, and memory size. An example of a MPADS query command is

MobilePhone.objects.get('display.size > Pixels(120,160) and camera.megaPixels > MegaPixels(3) and internalMemory.size > MegaBytes(64)')

For designing and manufacturing a very slim mobile phone, an example of a MPADS query for MIC components is

Mic.objects.get('dimension < DimensionInMm(6.5,2.3)')

An example of results for such a query is shown in Figure 5.

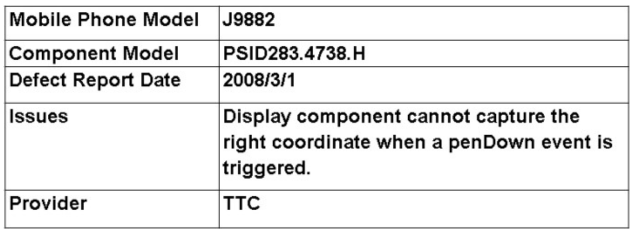


Function	MIC
Dimension	4*1.0 with rubber boot
Vender	CIN
Name	BMG10BT
P/N	2C.42030.022
Project	J2738
Picture	
Note	rubber boot with coil spring

Figure 5: Search results for MIC components

Another common product manager function is checking defect reports as a means of avoiding unreliable components. In this case study, a defect was found in the handwriting display—it was incapable of capturing the correct coordination following a penDown event. To perform a MPADS query for defective component reports, a project manager would write
DefectReport.objects.get('component=Display')

An example of query results is shown in Figure 6.



Mobile Phone Model	J9882
Component Model	PSID283.4738.H
Defect Report Date	2008/3/1
Issues	Display component cannot capture the right coordinate when a penDown event is triggered.
Provider	TTC

Figure 6: Results from defect report query

6.5 Costs and Benefits Evaluation

Table 6 addresses tasks for which implementation costs of MPADS are larger than a simple Excel file. The two cost types are (a) static (one-time efforts during development cycle); and (b) dynamic (to integrate a new data source format into MPADS, developers must implement new classes).

Task	Description	Cost Type
SOF adapter	New data sources need new SOF adapters for reading and parsing into model objects.	Dynamic
Model class definition	New data formats need to define new model classes to represent them.	Dynamic
Semantic definition	New data sources need new semantic definitions in source code.	Dynamic
Customized web GUI	Web GUI differs according to the application being used.	Dynamic
SOF parser	Individual programming languages need specific SOF parsers to process semantic definitions in source code. Once a SOF parser is implemented, it can be reused in all projects.	Static
SOF query engine	Can be reused in all projects.	Static
SOF web server	Provides HTTP access; can be reused in all projects.	Static
Training	Users need to be trained only one time to use SOF-based system.	Static
Server hardware	SOF system needs server hardware to provide web-based service to users.	Static
Server maintenance	SOF system needs an administrator to maintain proper function.	Static

Table 6: Tasks for which Implementation costs of MAPDS are larger than a simple Excel file

Excel files and MPADS have their individual benefits and drawbacks as follows (Table 7):

Benefit	Favored
Low static and dynamic costs for implementation.	Excel
Different departments can use unique data formats without extra communication, reducing overhead.	Excel
Low user-training costs.	Excel
No need for a hardware server to provide web-based service.	Excel
Ease and efficiency in querying heterogeneous data.	MPADS
Provides standard RDF formats for third-party data exchanges.	MPADS
Automatically transforms different semantic query units (e.g., USD, Euro).	MPADS
Various cooperative design features.	MPADS

Table 7: Benefits and drawbacks of Excel and MPADS

7 Conclusion and Future Work

The focus of this paper was on the publication of model objects to RDF documents that provide SOF solutions for automatic conversion tasks, as opposed to existing methods that require the manual conversion of model objects to a triple-oriented format. Our proposed SOF system can be used to modify class and attribute semantics embedded in program code as well as to enhance descriptions of relationships between classes and attributes in object-oriented languages. In addition to preserving the synchronization of relationship descriptions between classes and program codes, our proposed system may support multiple programming languages. SOF provides a direct publication flow for the Semantic Web, allowing users to conduct queries across heterogeneous data sources and to incorporate positive features from both O-O programming and the Semantic Web.

Our main contributions are embedding semantic descriptions in source code without changing programming language syntax. Although EClass can also put semantic descriptions in class definitions, it changes the Java syntax and requires the rewriting of compilers. In a computer-supported cooperative work environment, it is very important to use developing tools with interoperability. SOF provides a better solution for developers to extend semantic features for existing object-oriented compilers or interpreters without rewriting them.

The development tools associated with the SOF are insufficient, especially in terms of automation support for integrated development environment (IDE). An IDE development environment for various languages is required to support autocomplete, dynamic syntax checking, and mutual synchronization between semantic diagrams and program codes [Astels, 02]. In cases where illegal SOF statement syntax occurs or where a semantic conflict between SOF statements emerges, a more powerful tool is needed to automatically analyze the problem and to report results in a form that developers can use. In future projects we will work on IDE development tools to support the SOF system.

References

[Aqqal, 08] Aqqal, A., Rensing, C., Steinmetz, R., Elkamoun, N., Berraissoul, A.: Using taxonomies to support the macro design process for the production of Web Based Trainings, Journal of Universal Computer Science, (2008)

[Astels, 02] Astels, D.: Refactoring with UML, In Proceedings of 3rd International Conference on eXtreme Programming and Flexible Processes in Software Engineering (XP2002), (2002) 67-70

[Babik, 06] Babik, M., Hluchy, L.: Deep Integration of Python with Web Ontology Language, 2nd Workshop on Scripting for the Semantic Web (ESWC 2006), (2006)

[Bemers-Lee, 01] Bemers-Lee, T., Hendler, J., Lassila, O.: The Semantic Web, Scientific American, 284, 5, (2001) 34-43

[Bizer, 03] Bizer, C.: D2R MAP-A Database to RDF Mapping Language, Proceedings of the 12th International World Wide Web, (2003) [Bizer, 04] Bizer, C., Seaborne, A.: D2RQ-Treating Non-RDF Databases as Virtual RDF Graphs, Proceedings of the 3rd International Semantic Web Conference, (2004)

[Burger, 08] Burger, T.: The need for formalizing media semantics in the games and entertainment industry, Journal of Universal Computer Science, (2008)

[Burkard, 08] Burkard, B., Vogeler, G., Gruner, S.: Informatics for historians: Tools for medieval document XML markup, and their impact on the history-sciences, Journal of Universal Computer Science, (2008)

[Carroll, 03] Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: Implementing the Semantic Web Recommendations, Technical Report HPL-2003-146, HP Laboratories, (2003)

[Carroll, 04] Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: implementing the semantic web recommendations, Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, (2004) 74-83

[Koide, 05] Koide, S., Aasman, J., Haflich, S.: OWL vs. Object Oriented Programming, International Workshop on SemanticWeb Enabled Software Engineering (SWESE), (2005)

[Koide, 06] Koide, S., Takeda, H.: OWL-Full Reasoning from an Object Oriented Perspective, Asian Semantic Web Conf., ASWC2006, 4185, (2006) 263-277

[Kramer, 99] Kramer, D.: API documentation from source code comments: a case study of Javadoc, Proceedings of the 17th annual international conference on Computer documentation, (1999) 147-153

[Krasner, 88] Krasner, G.E., Pope, S.T.: A cookbook for using the model-view controller user interface paradigm in Smalltalk-80, Journal of Object-Oriented Programming, 1, 3, (1988) 26-49

[Lassila, 99] Lassila, O., Swick, R.R.: Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation, (1999)

[Leslie, 02] Leslie, D.M.: Using Javadoc and XML to produce API reference documentation, Proceedings of the 20th annual international conference on Computer documentation, (2002) 104-109

[Liu, 04] Liu, F.F., Wang, J., Dillon, T.S.: An Object-oriented Approach on Web Information Representation and Derivation, Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service, (2004) 309-314

[Liu, 07] Liu, F.F., Wang, J., Dillon, T.S.: Web Information Representation, Extraction, and Reasoning based on Existing Programming Technology, Web Information Representation, Extraction and Reasoning based on Existing Programming Technology, Studies in Computational Intelligence, 37, (2007) 147-168

[McBride, 02] McBride, B.: Jena: A Semantic Web Toolkit, IEEE Internet Computing, 6, (2002) 55-59

[McGuinness, 04] McGuinness, D.L., Van Harmelen, F.: OWL Web Ontology Language Overview, W3C Recommendation, (2004)

[Oren, 06] Oren, E., Delbru, R.: Object-oriented RDF in Ruby, Scripting for Semantic Web (ESWC), (2006) [Oren, 07] Oren, E., Delbru, R., Gerke, S., Haller, A., Decker, S.: ActiveRDF: object-oriented semantic web programming, Proceedings of the 16th international conference on World Wide Web, (2007) 817-824

[Prud'Hommeaux, 06] Prud'Hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF, W3C Working Draft, (2006)

[Shafranovich, 05] Shafranovich, Y.: Common Format and MIME Type for Comma-Separated Values (CSV) File, Internet Eng. Task Force draft, (2005)

[Valkeapaa, 08] Valkeapaa, O., Alm, O., Hyvonen, E.: An adaptable framework for ontology-based content creation on the semantic web, Journal of Universal Computer Science, (2008)

[Van Rossum, 03] Van Rossum, G., Drake Jr, F.L.: Python Language Reference Manual, Network Theory Ltd, (2003)

[Vega-Gorgojo, 08] Vega-Gorgojo, G., Bote-Lorenzo, M.L., Gomez-Sanchez, E., Asensio-Perez, J.I., Dimitriadis, Y.A., Jorrin-Abellan, I.M.: Ontoolcole: Supporting educators in the semantic search of CSCL tools, Journal of Universal Computer Science, 14, 1, (2008) 27-58

[Vrandecic, 05] Vrandecic, D.: Deep integration of scripting language and semantic web technologies, ESWC Workshop on Scripting for the Semantic Web, (2005)

[Ying, 07] Ying, P., Tianjiang, W., Xueling, J.: Building Intelligent Information Retrieval System Based on Ontology, Electronic Measurement and Instruments, 2007. ICEMI '07. 8th International Conference, 4, (2007) 612-615