

Reasoning in Semantic Web Using Jena

Ayesha Ameen¹, Khaleel Ur Rahman Khan² and B.Padmaja Rani³

¹ I.T Department, Deccan College of Engineering and Technology,
Hyderabad, Andhra Pradesh, India
ameenayasha@gmail.com

² Professor and Dean of Studies, Ace Engineering College,
Hyderabad, Andhra Pradesh, India
khaleelrkhan@aceec.ac.in

³ Head of Department CSE, JNTUH
Hyderabad, Andhra Pradesh, India
padmaja_jntuh@yahoo.co.in

Abstract

Semantic web extends the current web by adding semantics. By adding semantics we enable intelligent reasoning to be done on web. In this paper an application is created in eclipse using Jena semantic web development framework. Application developed consists of creating several classes and properties. Jena supports three operations on the model which were shown by creating two appropriate schemas. Reasoning capabilities of Jena is demonstrated by applying an OWL reasoner to the application for additional inference. At last the validity of the inference made after reasoning was tested and it was found to be consistent.

Keywords: *Semantic web, Ontology, Eclipse, Jena, Reasoner.*

Introduction

Semantic web is the next generation web with an aim to allow much more advanced knowledge management systems by organizing knowledge into conceptual spaces according to its meaning. Semantic web uses automated tools and reasoners for supporting knowledge maintenance by checking inconsistencies and extracting new knowledge from existing knowledge [1].

In this paper two ontology models are created in eclipse using Jena semantic web development framework. Capabilities of Jena are demonstrated by adding, differentiating and intersecting the models. At last applied a reasoner is applied for checking the consistency of the new model and inferring additional information. After checking the consistency of the model it was found consistent.

This paper is organized as; first section gives an introduction of Jena followed by the development of application. In the next section operations are performed on the schema i.e. addition, difference and intersection. In the next section method of applying a reasoner on the new

Model created after adding both the schemas is demonstrated. Last section comprises of the validation check that was performed on the inferred model created after reasoning.

2. Related work

The worked done in this domain comprised of working on DARPA Agent Markup Language (DAML) and performing inference on the Semantic Web [2]. The approach used DAMLJessKB maps DAML's semantics and put this into facts and rules for producing inferences.

Another work uses data and knowledge encoded in semantic web documents using an F-OWL inference engine based on F-logic. F-logic is an approach to describe a frame based system in logic [3].

Both the above mentioned approaches device a new inference mechanism for reasoning. The approach used in this paper is built on the inference supported in Jena. OWLReasoner which is one of the build in reasoner in Jena is used to perform inference. The validity of the inference made was tested by performing a validation test and it was found to be consistent without any inconsistencies.

3. Jena Semantic web development framework

Jena is an open source Semantic Web framework for Java. Jena has an API to extract data from and write to RDF graphs and OWL ontologies. Model represents a graph in Jena [2]. A model can be created by using data from URLs, files, databases or by combining different sources. In memory and persistent storage for storing large number of RDF triples is provided in Jena. SPARQL can be used to query model. Jena has built in support for many internal reasoners .Pellet reasoner can be used in Jena.

3.1 Jena Ontology API

Ontologies can be represented by various languages in semantic web ranging from RDFS which is weakest to OWL which is the strongest. Jena ontology API provides a consistent programming interface for ontology application development [5]. Jena ontology API is independent of ontology language used during programming. The Jena Ontology API is language-neutral class names in Java do not mention the underlying language.

OntClass Java class which represent OWL class, RDFS class, or DAML class. Profile is used to establish the differences between the various representations. Every ontology languages are associated with a profile, which contains the details of names of the classes and properties and the permitted constructs. Profile is bounded to an ontology model. OntModel is an extended version of Jena's Model class, which allows access to the statements in a collection of RDF data. OntModel extends this access by adding support for the kinds of objects in ontology such as classes, properties and individuals.

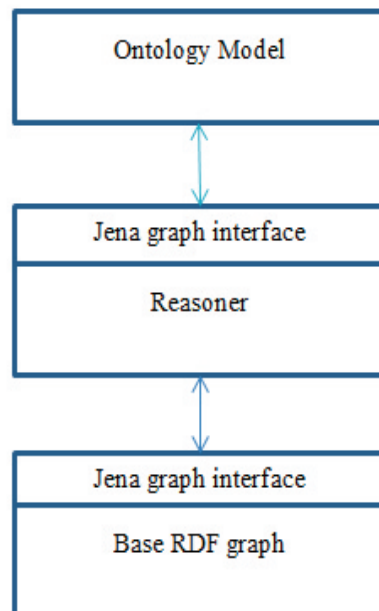


Fig. 1.Statements seen by OntModel

3.2 Reasoning

Reasoners work on the ontology to derive additional truth on the modeled concepts [6]. Jena reasoner creates a new RDF model containing asserted and derived tuples. This extended model can be queried in the same way as a plain RDF model. Jena inference subsystem allow a range of reasoners for deriving additional facts including Transitive reasoner which implements transitive and reflexive properties ,RDFS rules reasoner containing RDFS entailments, OWL reasoner ,DAML reasoner, Generic rule reasoner for supporting user defined rules.

4. Application development

For creating application in Jena ontology API [5]. Jena ontology model is used which is an extension of the Jena RDF model with an extra capabilities for handling ontologies. Jena [ModelFactory](#) is used to create ontology models .The most simple way to create an ontology model is as follows:

```
OntModel m = ModelFactory.createOntologyModel();
```

An ontology model with the default settings will be created [7]. Default settings consist of maximum compatibility with the previous version of Jena. The default settings consist of OWL-Full language support, RDF inferences producing entailments from sub property and sub class hierarchy, in memory storage.

4.1 Classes and Properties

Classes are the basic building blocks of ontology. [OntClass](#) object in Jena represents a simple class [8]. An ontology class is a facet of an RDF resource. Classes are created by calling createClass method. Properties are added by calling createObjectProperty method and createDatatypeProperty methods for creating object and datatype properties respectively.

Two ontological schemas are created in Jena .The first schema represents various categories of persons and the second schema show the categorization of working people. Person is the root class of the first schema which is further divided into several subclasses as shown in figure 2.Working is the root class of the second schema which is divided into Professional and Nonprofessional as shown in figure 3.

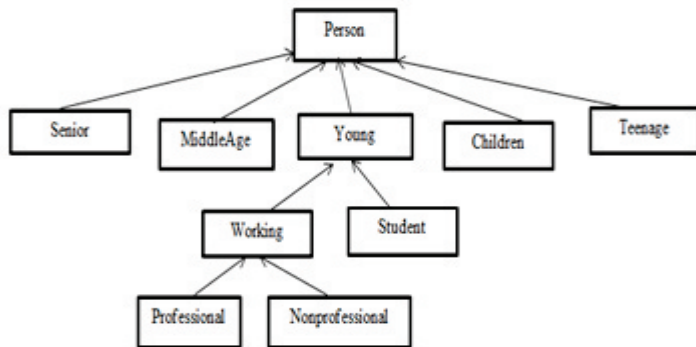


Fig. 2.Schema1

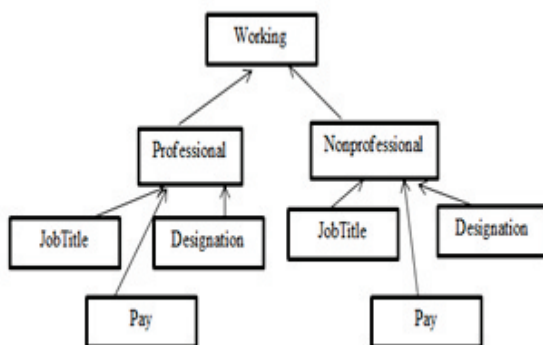


Fig. 3. Schema2

In our application several classes are created as shown in figure 4. Schema1 root class is Person. Person class is further divided into Senior, MiddleAge, Children, Teenage and Young. Young class is further divided into Working and Student. Working is again divided into Professional and NonProfessional.

Schema2 root class is working which is further divided into Professional and NonProfessional. Both Professional and NonProfessional classes are further divided into JobTitle, Pay,Designation. Subclasses are declared as disjoint and are arranged in an hierarchy with the code shown in figure 5 and figure 6.

```
schema1 = ModelFactory.createOntologyModel();
schema2 = ModelFactory.createOntologyModel();
// classes schema1
OntClass Person = ((OntModel) schema1).createClass("Person");
OntClass Senior = ((OntModel) schema1).createClass("Senior");
OntClass MiddleAge = ((OntModel) schema1).createClass("MiddleAge");
OntClass Young = ((OntModel) schema1).createClass("Young");
OntClass Student = ((OntModel) schema1).createClass("Student");
OntClass Working = ((OntModel) schema1).createClass("Working");
OntClass Professional = ((OntModel) schema1).createClass("Professional");
OntClass NonProfessional = ((OntModel) schema1).createClass("NonProfessional");
OntClass Children = ((OntModel) schema1).createClass("Children");
OntClass Teenage = ((OntModel) schema1).createClass("Teenage");
// classes schema2
OntClass Working2 = ((OntModel) schema2).createClass("Working");
OntClass Professional2 = ((OntModel) schema2).createClass("Professional");
OntClass NonProfessional2 = ((OntModel) schema2).createClass("NonProfessional");
OntClass Job Title = ((OntModel) schema2).createClass("Job Title");
OntClass Designation = ((OntModel) schema2).createClass("Designation");
OntClass Pay = ((OntModel) schema2).createClass("Pay");
```

Fig. 4. Code for creating classes

Properties are created. Domain (class) and range (class) are added to the properties as shown in figure 7 and figure 8.

```
// Hierarchy of schema 1
Senior.addClass(Person);
MiddleAge.addClass(Person);
Young.addClass(Person);
Student.addClass(Young);
Working.addClass(Young);
Professional.addClass(Working);
NonProfessional.addClass(Working);
Children.addClass(Person);
Teenage.addClass(Person);
Senior.addDisjointWith(MiddleAge);
Senior.addDisjointWith(Young);
Senior.addDisjointWith(Children);
Senior.addDisjointWith(Teenage);
Student.addDisjointWith(Working);
```

Fig.5 .Code for creating hierarchy in schema1

```
// Hierarchy of schema2  
Professional2.addSuperClass(Working2);  
NonProfessional2.addSuperClass(Working2);  
JobTitle.addSuperClass(Professional2);  
Designation.addSuperClass(Professional2);  
Pay.addSuperClass(Professional2);  
JobTitle.addSuperClass(NonProfessional2);  
Designation.addSuperClass(NonProfessional2);  
Pay.addSuperClass(NonProfessional2);  
Professional2.addDisjointWith(NonProfessional2);  
JobTitle.addDisjointWith(Designation);  
JobTitle.addDisjointWith(Pay);
```

Fig. 6 .Code for creating hierarchy in schema2

```
// Adding properties to Schema 1  
ObjectProperty hasChildren = ((OntModel schema1).createObjectProperty("hasChildren");  
DatatypeProperty hasName = ((OntModel schema1).createDatatypeProperty("hasName");  
DatatypeProperty hasAge = ((OntModel schema1).createDatatypeProperty("hasAge");  
    hasChildren.addDomain(Person);  
    hasChildren.addRange(Children);  
    hasAge.setDomain(Person);  
    hasAge.setRange(XSD.integer);  
    hasName.setDomain(Person);  
    hasName.setRange(XSD.Name);
```

Fig. 7.Code for adding Properties to classes

```
// Adding Properties to Schema 2  
DatatypeProperty hasDegree = ((OntModel schema2).createDatatypeProperty("hasDegree");  
DatatypeProperty hasExperience = ((OntModel schema2).createDatatypeProperty("hasExperience");  
    hasDegree.setDomain(Working2);  
    hasDegree.setRange(XSD.Name);  
    hasExperience.setDomain(Working2);  
    hasExperience.setRange(XSD.integer);
```

Fig. 8.Code for adding properties to classes

5. Operations on schemas

Jena provides three kinds of operations on schema to combine information .The operations are union, intersection and difference [5].

5.1 Union

Union (Model): A new model is created containing all the statements in this model with all of those in another given model. It can also merge data from different data sources. After applying union to both the schemas by giving `schema1.add (schema2)` command the following output is obtained at console as shown in figure 8. All classes present in both the schemas are displayed along with the properties and the root of the classes after adding is displayed as person. Output schema generated after adding is shown in figure 9.

```

    Problems @ Javadoc Declaration Properties Console
    <terminated> JenaDemo [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (26-Nov-2013)
    Schema After Adding /n
    List of classes After adding are JobTitle
    List of classes After adding are Children
    List of classes After adding are Senior
    List of classes After adding are Student
    List of classes After adding are Young
    List of classes After adding are Professional
    List of classes After adding are Pay
    List of classes After adding are Designation
    List of classes After adding are Working
    List of classes After adding are MiddleAge
    List of classes After adding are Person
    List of classes After adding are Teenage
    List of classes After adding are NonProfessional
    Root hierchary of classes after adding is Person
    object properties of classes after adding are hasChildren
    Datatype properties of classes after adding are hasExperience
    Datatype properties of classes after adding are hasDegree
    Datatype properties of classes after adding are hasAge
    Datatype properties of classes after adding are hasName
    
```

Fig. 8. Output after adding schema2 to schema1

5.2 Intersection

Intersection (Model): Intersection creates a new model containing all the statements which are in both this model and another model. After applying intersection to schema1 and schema2 by giving command `schema2.intersection (schema1)` the output obtained is shown in figure 10.

5.3 Difference

Difference (Model): Difference create a new model containing all the statements in this model which are not in another model after applying difference to both the schemas by giving command `schema1.difference (schema2)` output obtained is shown in figure 11.

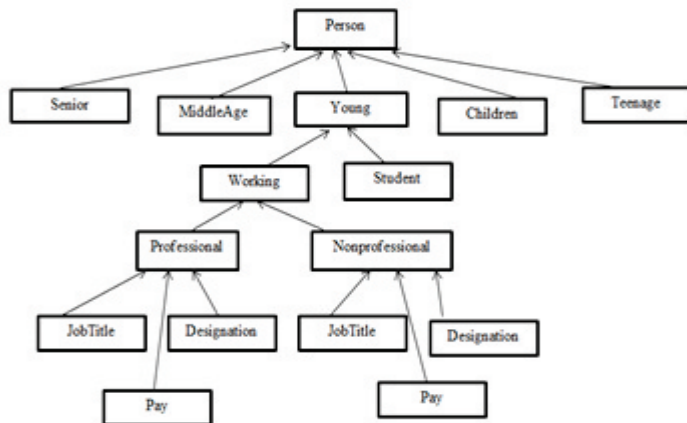
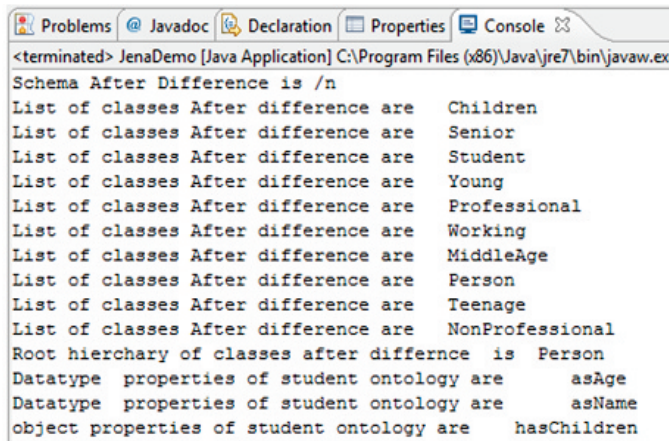


Fig. 9. Schemas after adding

```

    Problems @ Javadoc Declaration Properties Console
    <terminated> JenaDemo [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (27-Nov-2013)
    Schema After Intersection is /n
    List of classes After Intersection is Pay
    List of classes After Intersection is Designation
    List of classes After Intersection is JobTitle
    List of classes After Intersection is NonProfessional
    List of classes After Intersection is Professional
    List of classes After Intersection is Working
    Root hierchary of classes after Intersection is Working
    Datatype properties of student ontology are asExperience
    Datatype properties of student ontology are asDegree
    
```

Fig. 10. Output after applying intersection operation on schema1 and schema2



```
<terminated> JenaDemo [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe
Schema After Difference is /n
List of classes After difference are Children
List of classes After difference are Senior
List of classes After difference are Student
List of classes After difference are Young
List of classes After difference are Professional
List of classes After difference are Working
List of classes After difference are MiddleAge
List of classes After difference are Person
List of classes After difference are Teenage
List of classes After difference are NonProfessional
Root hierarchy of classes after difference is Person
Datatype properties of student ontology are asAge
Datatype properties of student ontology are asName
object properties of student ontology are hasChildren
```

Fig. 11. Output after applying difference operation

6. Reasoning

Reasoner is applied after merging both the schema by following steps given below [6]. The first step in applying the reasoner is to find the appropriate reasoner as Jena support many types of reasoner and there is also build in support for the reasoners in Jena. Reasoner must be applied after selecting appropriate reasoner. Next step is to create an inference model to study the output after reasoning. At last access the inference model which contains the output of the reasoning.

6.1 Finding a reasoner

Reasoner Factory is the factory class present for every type of reasoner. A reasoner can be created by calling an instance of reasoner factory or by retrieving from reasoner registry which contains instances indexed by URI assigned to the reasoner. There are additional methods on ReasonerRegistry for locating the instance of the reasoner like `getTransitiveReasoner`, `getRDFSReasoner`, `getRDFSReasoner`, `getOWLReasoner`, `getOWLMiniReasoner`, `getOWLMicroReasoner`. An OWL reasoner is used in our example by giving following statement.

```
Reasoner reasoner = ReasonerRegistry.getOWLReasoner();
```

6.2 Applying a reasoner to data

After the creation of reasoner instance it must be attached to both schema data and instance data. In our example only schema data is present. Reasoner.bindSchema method to bind a reasoner to schema.

```
reasoner = reasoner.bindSchema(schema1);
```

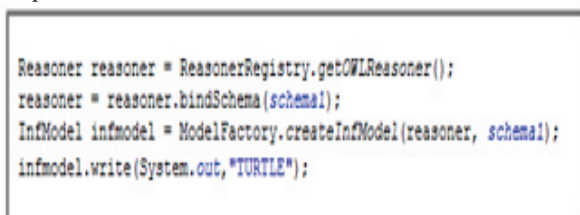
6.3 Creating an inference model

An inference model has to be created after bind. ModelFactory.createInfModel method is called to create an inference model.

```
InfModel infmodel = ModelFactory.createInfModel(reasoner, schema1);
```

6.4 Accessing inferences

Information stored in inference model must be accessed. The content of inference model can be written to an output file in turtle and a check can be made on the inferences made by the owl reasoner as shown in figure 12.



```
Reasoner reasoner = ReasonerRegistry.getOWLReasoner();
reasoner = reasoner.bindSchema(schema1);
InfModel infmodel = ModelFactory.createInfModel(reasoner, schema1);
infmodel.write(System.out, "TURTLE");
```

Fig. 12. Code for reasoning

6.5 Inferences

Output generated after performing inference are displayed on the console in turtle format. Reasoner is applied after merging both the schemas a list of all classes present in added schema must be generated in the output along with the properties for class. Details of designation class in the output inference model is as shown in below figure 13.

```
<Designation>
a owl:Class, rdfs:Resource, rdfs:Class;
rdfs:subClassOf, <Person>, owl:Thing, <Young>,
<Working>, <Professional>,
rdfs:subClassOf, <Person>, owl:Thing, <Young>,
<Working>, <NonProfessional>
owl:disjointWith, <JobTitle>, <Pay>
owl:equivalentClass, <Designation>.
```

Fig.13. Designation class details after reasoning.

```
<hasExperience>
a rdf:Property, owl:DatatypeProperty, rdfs:Resource;
rdfs:domain <Person>, owl:Thing, <Young>,
rdfs:Resource, <Working>;
rdfs:range rdfs:Resource, xsd:decimal, xsd:integer;
owl:equivalentProperty <hasExperience>.
```

Fig. 14. hasExperience property details after reasoning

```
<hasDegree>
a rdf:Property, owl:DatatypeProperty, rdfs:Resource;
rdfs:domain <Person>, owl:Thing, <Young>,
rdfs:Resource, <Working>;
rdfs:range rdfs:Resource, xsd:Name;
owl:equivalentProperty <hasDegree>.
```

Fig.15. hasDegree property details after reasoning

Properties hasExperience and hasDegree were created for the schema2 but after adding they are included in schema1 as the properties of subclasses of Person, Young, working, Professional and NonProfessional as shown in figure 14 and figure 15.

7. Validation

Validation interface is used to check and detect whether some constraint expressed using ontology languages in semantic web are violated or not [6]. `InfModel.validate()` interface is used to check for inconsistencies in data using a reasoner, it performs global check on the schema and instance data looking for inconsistencies and creates `ValidityReport` object consisting of a simple pass/fail flag. If `ValidityReport.isValid()` method returns true then they are no inconsistencies in the reasoned data if it is not valid then a report consisting of detected inconsistencies will be generated as an instances of the `ValidityReport.Report` interface.

The code used for validation of the current application is given in the figure 15. In the code first a call is made to `InfModel.validate()` interface which returns a variable `validity` of `ValidityReport` type. Check for inconsistencies is carried by calling `validity.isValid()` if the output is pass then output consists of statement no errors after validation otherwise error report consisting of inconsistencies is printed as shown in the below figure 16.


```
ValidityReport validity = infmodel.validate ();
if (validity.isValid())
{
    System.out.println("Output of validation test");
    System.out.println("No errors after validation");
}
else
{
    System.out.println("Conflicts")
    for (Iterator i = validity.getReports(); i.hasNext(); )
    {
        ValidityReport report = (ValidityReport)i.next();
        System.out.println(" - " + report);
    }
}
```

Fig. 16 .Code for validation tests.

After applying validation test to our application the output consists of as no errors after validation, our application has successfully passed the validation test and is consistent with no conflicts as shown in figure 17.

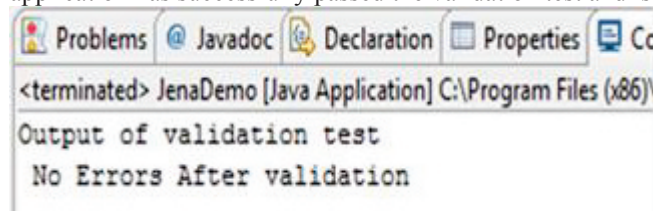


Fig. 17.Output after validation

8. Conclusions

This paper started with an introduction of Jena semantic web development framework. An application was developed with two ontological schemas. Jena capabilities of adding, intersecting and differentiating were demonstrated by taking input as schemas. Reasoner was applied on the new schema which was generated after adding both schemas .Validation was performed after reasoning to check for inconsistencies in the schema generated after reasoning .Our application had passed the validation test with no conflicts in the schema.

9. Future work

In future we want to focus more on the reasoning capabilities of Jena. We want to develop user defined rule using generic rule reasoner. At last we want to develop an application for personalization of preferences for user based on user defined rule.

References

- [1]Antoniou, Grigoris. *A semantic web primer*. the MIT Press, 2004.
- [2]Kopena, Joseph B., and William C. Regli. "DAMLJessKB: A tool for reasoning with the semantic web." In *The Semantic Web-ISWC 2003*, pp. 628-643. Springer Berlin Heidelberg, 2003.
- [3]Zou, Youyong, Tim Finin, and Harry Chen. "F-owl: An inference engine for semantic web." In *Formal Approaches to Agent-Based Systems*, pp. 238-248. Springer Berlin Heidelberg, 2005.
- [4] McBride, Brian. "Jena: A semantic web toolkit." *IEEE Internet computing* 6, no. 6 (2002): 55-59.
- [5]Dickinson, Ian. "The jena ontology api." Jena. Sourceforge. net,[online] (2009).
- [6]Reynolds, Dave. "Jena 2 inference support." Online manual at <http://jena.sourceforge.net/inference/index.html> (2004).
- [7]<http://jena.apache.org/documentation/javadoc/jena/com/hp/hpl/jena/ontology/OntClass.html>.
- [8]<http://jena.apache.org/documentation/javadoc/jena/com/hp/hpl/jena/ontology/OntModel.html>.

Ayesha Ameen is Associate Professor at Deccan college of Engineering and Technology. She completed her M.Tech from JNTU Anantapur in 2007. Her research interest includes semantic web and Personalization.

Dr .Khaleel Ur Rahman Khan is currently Professor & Dean (Academics) at ACE Engineering College. He completed Ph.D in Computer Science and Engineering in June 2009 from Osmania University, Hyderabad, India. His Dissertation Topic was "Integration of Wireless Mobile Ad Hoc Networks and the Internet".His areas of interest are Ad Hoc Networking, Wireless Sensor Networks, Transaction Management in MANETs, Opportunistic Networks, VANETs, Semantic Web Personalization, Web Content mining.

Dr Padmaja Rani is currently Professor & Head of Department CS,JNTU Hyderabad. She completed her Ph.D in the year 2009. Her area of interest is information retrieval, Embedded systems and semantic web.

The IISTE is a pioneer in the Open-Access hosting service and academic event management. The aim of the firm is Accelerating Global Knowledge Sharing.

More information about the firm can be found on the homepage:
<http://www.iiste.org>

CALL FOR JOURNAL PAPERS

There are more than 30 peer-reviewed academic journals hosted under the hosting platform.

Prospective authors of journals can find the submission instruction on the following page: <http://www.iiste.org/journals/> All the journals articles are available online to the readers all over the world without financial, legal, or technical barriers other than those inseparable from gaining access to the internet itself. Paper version of the journals is also available upon request of readers and authors.

MORE RESOURCES

Book publication information: <http://www.iiste.org/book/>

Recent conferences: <http://www.iiste.org/conference/>

IISTE Knowledge Sharing Partners

EBSCO, Index Copernicus, Ulrich's Periodicals Directory, JournalTOCS, PKP Open Archives Harvester, Bielefeld Academic Search Engine, Elektronische Zeitschriftenbibliothek EZB, Open J-Gate, OCLC WorldCat, Universe Digital Library, NewJour, Google Scholar

