Software Reuse : Ontological Approach to Feature Modeling

Vinod Babu Matcha*,
Prof. Prasad Reddy P.V.G.D **,
Ch.V.M.K.Hari *, G.Srinivas ***, N.SanjeevaRao ***, B.Jayachand ***, J.N.V.R. Swarup kumar***,**
G. SriRamGanesh*, N.V.R.V.Vamsi Krishna***, I.kali Pradeep***,**
Prof. Ch.Ramesh****

* Dept of Systems and software engineering, Blekinge Institute of Technology Ronneby, Sweden.
** Dept of CS & SE, Andhra University, Visakhapatnam,
*** Dept of Information Technology, GITAM University, Visakhapatnam,

**** Dept of Computer Science, AITAM College, Tekkali,

Abstract
Domain Engineering is a software reuse approach in application domain mainly used to deliver high quality software under budget and time constraints. In domain engineering the feature models play a vital role by providing common and variant concepts. However, feature models hamper the development of domain due to lack of formal semantics and at the same time consistency checking of the feature configuration is ambiguous to the stakeholders. In this paper we captured the relationships in OWL using JENA and a pallet for consistency checking of the feature configuration.
Key words:
Ontological, Approach, Reuse.

1. Introduction

In software engineering, Domain engineering is a software reuse approach in the application domain. Domain analysis, domain design and domain implementation are performed in domain engineering. Why organizations use reuse approach in domain engineering? Because to deliver a new product within a shorter time and at a lower cost. Feature modeling is one of the software reuse approach in the domain engineering. We can identify the common and variant features in the particular domain through graphical representation of the feature modeling. We have many methods in the feature modeling like FORM (Feature Oriented Reuse Method) and FODA (Feature Oriented Domain Analysis) but there is lack of ontology concept in the features and feature modeling. Due to this, there is no automated tools to check the correctness of the particular feature configuration depend on the constraints specified in feature model [1].

Manuscript received August 5, 2009
Manuscript revised August 20, 2009

2. Background

In nature no object exists in isolation. ‘Software reuse’ is a method for developing new is component adding some extra functionalities to the existing ones. Domain engineering lets the organization produce products in a particular domain in shorter time and at a lower cost. In domain engineering we perform domain analysis and capture domain knowledge in the form of reusable assets.

Feature modeling is one of the ways for domain engineering and it is widely used in industry. It is very helpful in creating product lines. Products can be rapidly developed using the product lines. The products that come from the same product line vary in few features. So, all the products that are derived from the same product line have some common features and vary in some features. So, making products from product line is nothing but reusing of assets.

So, far it is good and people are using different tools to configure the features of a product. Feature configuration is the selection of a set of features for producing a product from the product line. But in mass customization of products, feature configuration is difficult as the stakeholders misinterpret the features some times and sometimes even the customer forgets to tell some of his required features for his desired product. For example if a customer wants to have a Mercedes-Benz C-class with a customization for racing. Then vender has to check all the features of C-Class and at the same time he has to check whether the new customized features are affecting the ultimate product. For example if a car product line offers either manual control or automatic control, and let suppose C-class car is offered with manual control and racing car offered with automatic control, but the ultimate product that was needed by the customer is C-class with a customization of racing then, we will find both the features, then the product can not be delivered. This is happening because either vendor or customer or both misinterpret the feature configuration. This is happened due to lack of proper semantics for feature models. For mass customization of products we need tools but, no tool is supporting semantics. Here we try to project an ontological approach to solve this problem. Before going to discuss the ontological approach we will discuss some of the concepts of future model (car feature model shown in fig 1).

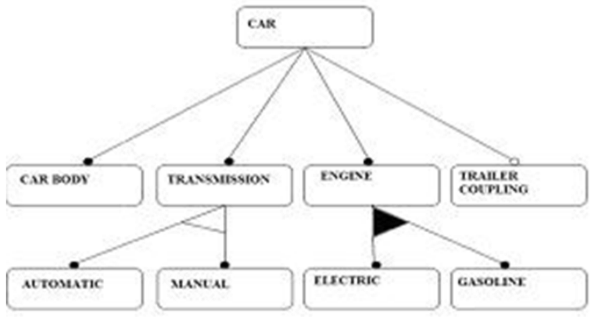
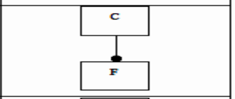
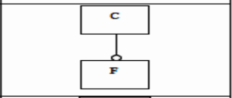
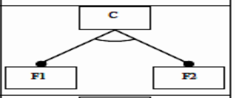
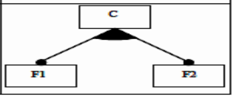


Fig 1: Car feature model [8].

2.1 Feature and Concept:

Feature represents a noticeable characteristic of a concept where as concept is a set of related features with constraints.

Table 1: Feature types [1].

Type	Notation
Mandatory	
Optional	
Alternative	
Or	

In Table 1, we are assuming the concept C is selected and explaining its child features as follows:
Mandatory means the feature F must compulsory be required in the description of a conceptual instance. Optional means the feature F may or may not be required in the description of a conceptual instance. Alternative means we have two features F1 and F2 in the diagram. Exactly one feature required from a set of features in the description of a conceptual instance. Or means we have two features F1 and F2 in the diagram. One or more features required from a set of features in the description of a conceptual instance.

2.2 Feature Configuration:

Future configuration is an instance of a set of futures that a concept holds. From the Figure.1 we can derive a set (CarBody, Transmission, Engine, Automatic, Electric). This set is a valid feature configuration and the number of valid feature configurations that can be derived from the above diagram are 12. Those are the cars with two types of transmission, three kinds of engines (electric, gasoline, both) and with a optional trailer coupling feature.

2.3 Ontology:

Ontology is a shared knowledge between people or software agents under common agreement [2]. We have different ontology languages like OWL (Web ontology language), OWL-S (Web ontology language for services), OWL-R (Web ontology language for Rules)...Etc. We are using OWL for the development of feature ontology.

OWL is the extension of RDF Schema [4], [5] and it is endorsed by World Wide Web consortium. Most of the elements of the OWL concerns with the classes, properties, instances of class and relationship between the instances.

3. Related Work

H Wang et.al., proposed theory about the feature modeling for domain engineering with respect to OWL and they have proposed a protégé tool and racer inference engine [1] [7]. But here we tried to use JENA through which we can construct ontology programmatically unlike the protégé where construction of ontology is done in tool. When coming to inference engine we have used a pallet which is supported in JENA. Besides that we have explained the whole process briefly using an example and reference [1].

4. Ontology Vs Feature Modeling:

Every feature in future diagram is mapped to OWL class and every attribute in the feature are mapped to object attribute. With reference to above feature diagram (Figure 1) we are going to explain mandatory, optional, alternative, or; features. These are discussed below by referencing [1], [3], [4], and [6].

Conceptual Modeling

1. First we need to build OWL ontology for nodes and edges in the diagram. Then we model the feature relations

in the diagram. We are going to construct ontology using parent node and child nodes. By taking the above diagram we draw the table 2 for parent and child relations and we shown child nodes in table 3.

Table 2.

Parent Node	Child Nodes
Car	CarBody, Transmission, Engine, TrailerCoupling
Transmission	Automatic, Manual
Engine	Electric, Gasoline

Now each node is modeled as OWL class (owl:Class) and we make them these nodes are mutually disjoint. We have to be noted that every class is subclass of owl:Thing in OWL.

Table 3.

Car	owl:Class
CarBody	owl:Class
Transmission	owl:Class
Engine	owl:Class
TrailerCoupling	owl:Class
Automatic	owl:Class
Manual	owl:Class
Electric	owl:Class
Gasoline	owl:Class

We make all these classes mutually disjoint to each other.
(owl:disjointWith Car CarBody),
(owl:disjointWith CarBody Transmission)
(owl:disjointWith Transmission Engine)
(owl:disjointWith Engine TrailerCoupling)
(owl:disjointWith TrailerCoupling Automatic)
(owl:disjointWith Automatic Manual)
(owl:disjointWith Manual Electric)
(owl:disjointWith Electric Gasoline)

We can use owl:AllDifferent to make all classes mutually disjoint to each other.

2. Now we are going to model the feature relations in the diagram. These relations are edge types in the diagram. These features are modeled as object properties in OWL (owl:ObjectProperty) and at the same time we give ranges to every object property. We define ranges in OWL as rdfs:range. We define range of every property is its corresponding class. We have defined these properties in the below table4.

Table 4.

Property Name	Property Type	Range
hasCar	owl:ObjectProperty	Car
hasCarBody	owl:ObjectProperty	CarBody
hasTransmission	owl:ObjectProperty	Transmission
hasEngine	owl:ObjectProperty	Engine
hasTrailerCoupling	owl:ObjectProperty	Trailer Coupling
Has Automatic	owl:ObjectProperty	Automatic
hasManual	owl:ObjectProperty	Manual
hasElectric	owl:ObjectProperty	Electric
hasGasoline	owl:ObjectProperty	Gasoline

3. We create Rule class for each node that is there in the feature diagram. We create the Rule class using with existential restriction. The restriction will be on the corresponding property with a restriction value of the corresponding feature class in table 5.

Table 5.

RuleClass Name	Restriction type	On Property	Restriction Value
CarRule	owl:some ValuesFrom	hasCar	Car
CarBodyRule	owl:some ValuesFrom	hasCarBody	CarBody
Transmission Rule	owl:some ValuesFrom	Has Transmission	Transmission
EngineRule	owl:some ValuesFrom	hasEngine	Engine
Trailer Coupling Rule	owl:some ValuesFrom	hasTrailer Coupling	Trailer Coupling
Automatic Rule	owl:some ValuesFrom	has Automatic	Automatic
ManualRule	owl:some ValuesFrom	hasManual	Manual
ElectricRule	owl:some ValuesFrom	hasElectric	Electric
Gasoline	owl:some ValuesFrom	hasGasoline	Gasoline

Now we are going to model the feature relations (mandatory, optional, alternative, or) as fallows:

Mandatory:

In the feature diagram we have mandatory features namely for Car are CarBody, Transmission and Engine. The mandatory relation says that, if parent node is included then, we have to include its child nodes. This can stated in another manner like, for ever instance of PRule (parent rule class) there should be some instance of F_i Class(child feature class where $1 \leq i \leq n$) on property hasF_i(property of corresponding class F_i) . This can be realized as, every PRule is subset or equal to owl:someValueFrom restricted type of F_i Class on property hasF_i. The mandatory features in the given diagram are realized in the table6.

Table 6.

Parent Rule Class	Subset of
CarRule	(owl:someValuseFrom hasCarBody CarBody)
CarRule	(owl:someValueFrom hasTransmission Transmission)
CarRule	(owl:someValueFrom hasEngine Engine)

Optional:

We have only one optional feature in the diagram, which is TrailerCoupling. The optional feature indicates that if a parent is included then a child may or may not be included. We can think in logical perspective that if a child is included then we have to include parent. Because, in feature configuration if you find any optional feature then have to see whether parent is included for the consistency of the ontology. This can be accomplished by making every F_i Rule (Child feature rule class where $1\leq i \leq n$) is subset or equal to owl:someValueFrom restricted type of PRule (Parent Rule Class). The optional feature in the diagram is realized in the table7.

Table 7.

Optional Feature name	Subclass of
TrailerCouplingRule	(owl:someValueFrom hasCar Car)

Alternative:

We have two alternative features that are Automatic and Manual. Alternative relation indicates that only on of the feature is to be included if the parent is included. This we can think in logical perspective that, if PRule (Parent Rule Class) is included then there is some feature F_i should be include. This can be realized as PRule is subset or equal to union of owl:someValueFrom restricted type of F_i Rule (Child feature rule class where $1\leq i \leq n$) and PRule is subset or equal to intersection of owl:someValueFrom restricted type of F_i Rule (Child feature rule class where $1\leq i \leq n$). This is shown in below table8.

X≡ (owl:uinonOf {(owl:someValueFrom hasAutomatic Automatic), (owl:someValueFrom hasManual Manual)})
Y≡ (owl:intersectionOf {(owl:someValueFrom hasAutomatic Automatic), (owl:someValueFrom hasManual Manual)})

Table 8.

Parent Feature Rule class	Subclass of
TransmissionRule	X
TransmissionRule	Complement of Y

We use complement of Y in order to ensure only one feature is included.

OR:

We have two feature as optional in the diagram those are Electric and gasoline. The or-relation indicates that any number of the features can be included if the parent is included. The logic is similar to alternative but here we do not use negation of the conjunction. That is Y as described above. Because, here can include more than one feature if parent is included. This can be realized as fallows and it is shown in below table9.

X≡ (owl:uinonOf {(owl:someValueFrom hasElectric Electric), (owl:someValueFrom hasGasoline Gasoline)})

Table 9

Parent Feature Rule class	Subclass of
Engine Rule	X

5. Feature Configuration Model

In the feature configuration we will set the features and check for its consistency. It is impertinent to check consistency in order to verify whether the set has any contradictory features. Suppose if we take a feature configuration set as (CarBody, Transmission, Engine, Automatic, Manual, Electric), then we can say that it is inaccurate. This is because, both Automatic and Manual features should not be in the configuration set as these two features are alternatives. Then this type of sets should be shown as inconsistent set. The consistency checking should be done using inference engine. The feature configuration is done as follows.

- We are going to make concept class as subclass of Rule class root node. In this context the Rule class of the root node is CarRule.
- We put existential restriction to each and every feature that is there in configuration for the features that are not included can not be have existential restriction.
- While configuring the future set we should explicitly state whether a feature is included or not. This is because OWL adapts open world assumptions, due to this reasoning engine can infer wrongly. To make a feature obscene we should use “cardinality=0”. OWL element for cardinality is owl:cardinality.
- The concept class should be made of conjunction of the above constraints.

Now we will take a valid configuration, that is (CarBody, Transmission, Engine, Automatic, Electric). This can be realized in the ontology as fallows:

(owl:subClassOf C CarRule) where

C ≡ (owl:intersectionOf {(I,J,K,L,M,N,O,P)}) where I,J,K,L,M,N,O,P are there in table 10.

Table 10.

Restricted Class Name	Restricted type	On Propery	Resricted Value	Cardinality	Feature Included
I	owl:some ValueFrom	hasCarBody	CarBody		Yes
J	owl:some ValueFrom	hasTransmission	Transmission		Yes
K	owl:some ValueFrom	hasEngine	Engine		Yes
L	No need of Restriction	hasTrailer Coupling		**0**	No
M	owl:some ValueFrom	Has Automatic	Automatic		Yes
N	No need of Restriction	hasManual		**0**	No
O	owl:some ValueFrom	hasElectric	Electric		Yes
P	No need of Restriction	hasGasoline		**0**	No

Here we L,N,P corresponds to features TrailerCoupling, Manual and Gasoline. These are not included in the configuration, that is reason we make those cardinalities 0, but we will input these features to inference engine as we stated before that every feature presence(present/absent) should shown explicitly. Now the inference engines checks its consistency and it will not contradict as it has no contradicted statements.

6. Tools

We have used JENA-2.5.7 for ontology construction which is a java frame work for OWL. For, to check consistency we have used inference engine pallet-1.3 which is compatible with JENA and it also supported in JAVA.

7. Conclusion And Future Work

In this paper we have enumerated about the construction of feature model in ontology and checked the consistency of the feature configuration with an appropriate example. We have also mentioned the tools that we have exerted. Software reuse is the best practice for deliver product facility. Briefly we want to conclude that ontological approach for feature modeling is very much beneficial in order check the consistency of the feature configurations as there is no formal semantics available for domain engineering.

So far we have worked on a prototype, but in near feature we want to apply this for lager features set.

References

[1] Hai Wang, Y F Li, J Sun, H Zhang and J Pan, “A Semantic Web Approach to Feature Modeling and Verification”.
[2] G. Antoniou and F.V.Hermelen, A Semantic web Primer. London: The MIT Press, 2004
[3] M.K. Smith, C. Welty and D.L.Mc.Guinness (2004), OWL Web Ontology Language Guide, http://www.w3.org/TR/owl-guide
[4] D.L.Mc.Guinness and F.V.Harmelen (2004), OWL Web Ontology Language Overview, http://www.w3.org/TR/owl-features
[5] B. Lee, Godel and Turing, Thinking on the web. United States of America: Wiley interscience publication, 2006
[6] http://www-ksl.stanford.edu/software/jtp/doc/owl-reasoning.html. [Online][Cited: 11 02, 2008.]
[7] H Wang, Y F Li, J Sun and H Zhang,” Verify Feature Model using protégé –owl”, ACM, Japan,MAY-2005.
[8] Czarnecki, K. and Eisenecker, U. Eomponents of Generative Programming. Proc. Of the 7th European Software Engineering Conference, September 1999.

Appendix A

The JENA implementation is

```
package featuremodelconsistancy;
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.vocabulary.OWL;
import java.io.FileOutputStream;
import org.mindswap.pellet.jena.PelletReasonerFactory;
public class Main {
   public static void main(String[] args) {
        try{
      String
filepath="F:\\FeatureModel\\OwlFiles\\fmont.owl";
      String uri="http://FeatureModel/OwlFiles/fmont#";
      String
namespace="http://FeatureModel/OwlFiles/fmont.owl";
     OntModel
m=ModelFactory.createOntologyModel(OntModelSpec.O
WL_MEM_MINI_RULE_INF);
    // OntModel
m=ModelFactory.createOntologyModel(OntModelSpec.O
WL_DL_MEM_RULE_INF);
     m.setNsPrefix("",uri);
     Ontology fmont=m.createOntology(namespace);
     //******Classes**********
    OntClass Car=m.createClass(uri+"Car");
    Car.addComment("Car Feature","EN");
    OntClass CarBody=m.createClass(uri+"CarBody");
    CarBody.addComment("CarBody Feature","EN");
    OntClass
Transmission=m.createClass(uri+"Transmission");
    Transmission.addComment("Transmission
Feature","EN");
    OntClass Engine=m.createClass(uri+"Engine");
    Engine.addComment("Engine Feature","EN");
    OntClass
TrailerCoupling=m.createClass(uri+"TrailerCoupling");
    TrailerCoupling.addComment("TrailerCoupling
Feature","EN");
    OntClass Automatic=m.createClass(uri+"Automatic");
    Automatic.addComment("Automatic Feature","EN");
    OntClass Manual=m.createClass(uri+"Manual");
    Manual.addComment("Manual Feature","EN");
    OntClass Electric=m.createClass(uri+"Electric");
    Electric.addComment("Electric Feature","EN");
    OntClass Gasoline=m.createClass(uri+"Gasoline");
    Gasoline.addComment("Gasoline Feature","EN");
    //******* Making all classes disjoint to each other
    RDFList Classlist=m.createList(new
RDFNode[]{Car,CarBody,Transmission,Engine,TrailerCo
upling,Automatic,Manual,Electric,Gasoline});
    AllDifferent
DisjointClasses=m.createAllDifferent(Classlist);
    //********** Making Feature Relations*******
    ObjectProperty
hasCar=m.createObjectProperty(uri+"hasCar");
    hasCar.addRange(Car);
    ObjectProperty
hasCarBody=m.createObjectProperty(uri+"hasCarBody");
    hasCarBody.addRange(CarBody);
    ObjectProperty
hasTransmission=m.createObjectProperty(uri+"hasTrans
mission");
    hasTransmission.addRange(Transmission);
    ObjectProperty
hasEngine=m.createObjectProperty(uri+"hasEngine");
    hasEngine.addRange(Engine);
    ObjectProperty
hasTrailerCoupling=m.createObjectProperty(uri+"hasTrai
lerCoupling");
    hasTrailerCoupling.addRange(TrailerCoupling);
    ObjectProperty
hasAutomatic=m.createObjectProperty(uri+"hasAutomati
c");    hasAutomatic.addRange(Automatic);
    ObjectProperty
hasManual=m.createObjectProperty(uri+"hasManual");
    hasManual.addRange(Manual);
    ObjectProperty
hasElectric=m.createObjectProperty(uri+"hasElectric");
    hasElectric.addRange(Electric);
    ObjectProperty
hasGasoline=m.createObjectProperty(uri+"hasGasoline");
hasGasoline.addRange(Gasoline);
    //************ Making RuleClasses
    SomeValuesFromRestriction
CarRule=m.createSomeValuesFromRestriction(uri+"CarR
ule",hasCar,Car);
    SomeValuesFromRestriction
CarBodyRule=m.createSomeValuesFromRestriction(uri+
"CarBodyRule",hasCarBody,CarBody);
    SomeValuesFromRestriction
TransmissionRule=m.createSomeValuesFromRestriction(
uri+"TransmissionRule",hasTransmission,Transmission);
SomeValuesFromRestriction
EngineRule=m.createSomeValuesFromRestriction(uri+"E
ngineRule",hasEngine,Engine);
    SomeValuesFromRestriction
TrailerCouplingRule=m.createSomeValuesFromRestrictio
n(uri+"TrailerCouplingRule",hasTrailerCoupling,TrailerC
oupling);
    SomeValuesFromRestriction
AutomaticRule=m.createSomeValuesFromRestriction(uri
+"AutomaticRule",hasAutomatic,Automatic);
    SomeValuesFromRestriction
ManualRule=m.createSomeValuesFromRestriction(uri+"
ManualRule",hasManual,Manual);
    SomeValuesFromRestriction
ElectricRule=m.createSomeValuesFromRestriction(uri+"
ElectricRule",hasElectric,Electric);
    SomeValuesFromRestriction
GasolineRule=m.createSomeValuesFromRestriction(uri+"
GasolineRule",hasGasoline,Gasoline);
    //*********Mandatory Features***********
    CarRule.addSuperClass(CarBodyRule);
    CarRule.addSuperClass(TransmissionRule);
    CarRule.addSuperClass(EngineRule);
    //************ Optional Features**********
    TrailerCouplingRule.addSuperClass(CarRule);
        //*********** Alternative Features*********
    UnionClass
XUnion=m.createUnionClass(uri+"XUnion",m.createList
(new RDFNode[] {AutomaticRule,ManualRule}));
    TransmissionRule.addSuperClass(XUnion);
    IntersectionClass
YIntersection=m.createIntersectionClass(uri+"YIntersecti
on",m.createList(new RDFNode[]
{AutomaticRule,ManualRule}));
    ComplementClass
YComplement=m.createComplementClass(uri+"YCompl
ement", YIntersection);
    TransmissionRule.addSuperClass(YComplement);
    //*********** OR Feartures**************
     UnionClass
ORUnion=m.createUnionClass(uri+"ORUnion",m.create
List(new RDFNode[] {ElectricRule,GasolineRule}));
```

```
      EngineRule.addSuperClass(ORUnion);
      FileOutputStream out;
      out= new FileOutputStream(filepath);
      m.write(out,"RDF/XML-ABBREV");
       // ******Checking ontology consistancy********
       //************ Feature Configuration
      OntModel m1 =
ModelFactory.createOntologyModel( PelletReasonerFact
ory.THE_SPEC,m );
      //CardinalityRestriction
ManualRestriction=m1.createCardinalityRestriction(uri+"
ManualRestriction", hasManual, 0);
      // If we replace ManualRule with ManualRestriction
in FeatureConfiguration then FeatureConfiguartion
becomes valid configuration set
      IntersectionClass
FeatureConfiguration=m1.createIntersectionClass(uri+"Fe
atureConfiguration",m.createList(new RDFNode[]
{CarBodyRule,TransmissionRule,AutomaticRule,Manual
Rule,EngineRule,ElectricRule,GasolineRule,TrailerCoupl
ingRule}));
      FeatureConfiguration.addSuperClass(CarRule);
      // listing the inconsistent classes ...it means listing
equivalentclasses to owl.nothing which is complement of
owl.Thing(owl.Thing is superclass of every class)--null
argument indicates it matches anything
      StmtIterator i = m1.listStatements( null,
OWL.equivalentClass, OWL.Nothing );
      while (i.hasNext()) {
      System.out.println( "Class " +
i.nextStatement().getSubject().getLocalName() + " is
unsatisfiable" );
        } String
filepath_config="F:\\FeatureModel\\OwlFiles\\configurati
on.owl";
        out= new FileOutputStream(filepath_config);
        m1.write(out,"RDF/XML-ABBREV");
       }catch(Exception
e){System.out.println(e.getMessage());}}}
```

The above code generates product line ontology (fmont.owl) and features confirmation ontology (configuration.owl).

Vinod Babu Matcha received his M.Sc in Computer science and M.tech in Information Technology from Andhra University in 2002 and 2007 respectively. Currently he is doing his M.Sc in software engineering from Blekinge Institute of Technology (Sweden). He has keen interest in ontology development and researching semantic web.

Dr Prasad Reddy P.V.G.D Professor, Over 20 years of Experience in Teaching with Andhra University handled courses for B.Tech, M.Tech, M.C.A and specialized in Enterprise wide Computing, XML based object models and scalable web applications. Research areas include Soft Computing, Software Architectures, and Knowledge Discovery from Databases, Image Processing, Number theory & Cryptosystems.

Mr. Ch.V.M.K. Hari, M.Tech., (Ph.D), Associate Professor. Over 9 years of Experience in Teaching with Andhra University, Acharya Nagarjuna University, Adhikavi Nannaya University, GITAM university handled courses for B.Tech, M.Tech, M.C.A and specialized in Software Engineering. Research area include Software Engineering, Knowledge Discovery from databases.

Mr. G. Srinivas, M.Tech., (Ph.D) Assistant Professor, Over 5 years of Experience in Teaching with GITAM University, handled courses for B.Tech, M.Tech and specialized in Machine Learning and Image Processing.

Prof. Ch.Ramesh, Dept of Computer Science, Over 12 years of Experience in Teaching with AITAM College of Engineering. Research area includes Image Processing.

Mr. N. Sanjeeva Rao, B.Tech., M.Tech(IT) Assistant Professor, Over 2 years of Experience in Teaching with GITAM University.

Mr. B. Jaya Chand, M.Tech(IT). Assistant Professor, Over 2 years of Experience in Teaching with GITAM University.

J.N.V.R. Swarup kumar, M.Tech(IT) from GITAM University.

G. SriRamGanesh, M.Tech(IT) from GITAM University.

N.V.R.V.Vamsi Krishna, M.Tech(IT) from GITAM University.

I. kali Pradeep, M.Tech(IT) from GITAM University.