Modeling Language and Tools for the Semantic Link Network

Hai Zhuge, Kehua Yuan, Jin Liu, Junsheng Zhang and Xiaofeng Wang

China Knowledge Grid Research Group, Key Lab of Intelligent Information Processing
Institute of Computing Technology, Chinese Academy of Sciences
P.O.Box 2704-28, 100080, Beijing, China

zhuge@ict.ac.cn; {kehua, jliu, junsheng, xfwang}@kg.ict.ac.cn

Summary

The Semantic Link Network (SLN) is the extension of the hyperlink Web by attaching semantics to hyperlinks. It is an approach to construct a semantic overlay on Web resources. The specification of the semantics of the SLN model is an essential issue of SLN application. This paper proposes a modeling language for SLN consisting of an algebraic definition for SLN, a SLN metamodel and a UML profile for SLN. The SLN metamodel specifies the primitives of the modeling language. The UML profile for SLN defines the specific syntax on SLN to make the modeling language understandable and usable. The development of the SLN builder implementing this language and the graphical SLN browser is introduced. This work is a part of the Semantic Link Network model.

KEY WORDS: Browser, Metamodel, Modeling Language, Semantic Link Network, UML.

1. INTRODUCTION

The simple hyperlink design is the key to the success of the World Wide Web. It enables any page to link to any page. Human's operating behaviors lead to the network effect and the uneven distribution of links. Another key to the success of the Web is its easy utility mode. It enables ordinary people to use it without any previous training. But the hyperlink is limited in ability to support relation reasoning because it does not contain any semantics.

The Semantic Web is to upgrade the Web to support intelligent applications by establishing machine-understandable semantics [3]. Much research has been done toward this goal by developing semantic richer markup languages and domain ontologies. XML is to reflect the structural information of documents (www.w3.org/XML). Resource Description Framework (RDF, www.w3.org/RDF) has been proposed and used to describe relations between objects, attributes and values. The Web Ontology Language OWL has been developed to support description of relations and logical reasoning. It is designed for use by applications that need to process the content of information instead of just presenting

information to humans, and it facilitates greater machine interpretability on Web content than those supported by XML and RDF by providing additional vocabulary together with a formal semantics (www.w3.org/TR/owl-features). More powerful languages are expected to emerge in the near future.

Rethinking the success of the Web — the simplicity and easy-to-use features can help evaluate the current efforts towards the Semantic Web.

Can we develop a Semantic Web by inheriting the successful features of the Web?

Research on the Semantic Link Network (SLN) model is increasing in recent years [18, 19-24]. The model is the extension of the current hyperlink Web by attaching semantics to hyperlinks. It is a simple way to realize the Semantic Web by establishing a semantic linking overlay on Web resources including Web pages and underlying strucutres [3]. The SLN enriches the semantics of the Web and supports semantic calculus.

Compared with other Semantic Web solutions, the major advantages of the SLN are its simplicity and the nature of semantic self-organization: *any node can link to any semantic relevant node*. It supports the implementation of an autonomous and self-organized Semantic Web [19, 22].

Pons applies the semantic link technique to improve Web page prefetching [21, 22]. He proposes a prefetcher utilizing this semantic link information associated with the current Web page's hyperlink set to predict which Web objects to prefetch during the limited view time interval of the current Web page. The research result shows that the semantic link approach is effective in improving Web browser's cache-hit percentage while significantly lowering Web page rendering latency [18].

A SLN-Builder was implemented to facilitate the construction of the SLN [21]. It focuses on defining the relations on documents. An intelligent browser was also developed to enable users to foresee several steps ahead during browsing the semantically linked document network. But the builder and browser have three shortcomings: lack of metamodel, lack of graphical display, and lack of distributed cooperative building mechanism. A cooperative mechanism is very important for a SLN to evolve from small and simple into large and rich.

The Unified Modeling Language (UML) is a collection of diagram notations for software modeling. Although accurate definition of its semantics is impossible [12], its syntax is well-known and its meaning is well-understood in software community. Using UML to explain the SLN enables users in the software community to understand the SLN.

This paper proposes a SLN modeling language consisting of the algebraic definition for SLN, the SLN metamodel and the UML profile for SLN. The SLN metamodel specifies the primitives of modeling SLN. As the specific syntax, the UML profile for the SLN is defined for users who are familiar with the UML to use the SLN model to describe domain knowledge. A SLN builder implementing the SLN

modeling language is developed to help users easily define a SLN. A graphical SLN browser is also developed to visualize the SLNs. A distributed SLN building and sharing prototyping system is developed to realize a cooperative SLN application environment.

2. DESIGN PRINCIPLE AND SLN MODELING TECHNIQUES

The design of the SLN modeling language follows four principles:

(1) ***Extensibility***. The SLN model consists of two modules: the core module and the extension module. The core module consists of a set of semantic nodes, the semantic links between nodes, and the linking rules (SLN Rules). It can be extended by adding more reasoning and mapping features [24].

(2) ***Compatibility***. The SLN metamodel should comply with an existing metamodel specification like the MOF (Meta-Object Facility) to ensure the interoperability between the SLN and the other relevant techniques in the modeling architecture of the metamodel.

(3) ***Well-defined.*** The metamodel should be well-defined to guarantee the correctness of the SLN modeling language and the valid transformation from the logic model to its physical counterpart.

(4) ***Easy-to-use***. The specific syntax of the SLN modeling language should adopt a well-known style to enable users to easily use the language.

The UML modeling community provides two built-in extension mechanisms for specific modeling needs, the Meta Object Facility MOF and the UML profile. The former is a meta metamodel that can be used to define specific metamodels like the SLN metamodel, while the latter can be used for a "lightweight" extension by attaching stereotypes or tags to UML model structure and introducing semantics for specific modeling purpose [17].

A metamodel outlines the abstract syntax of a modeling language. It can be used to specify several specific syntaxes. The UML profile is eligible to define a specific syntax that is widely acceptable [5]. In the MOF metamodeling architecture, each model layer employs the same syntax style to provide a coherent meaning for designers.

Fig. 1 illustrates the SLN modeling technique. The SLN metamodel adopts the MOF as the basic metamodeling language to describe the core semantic elements in the SLN matamodel [1, 6, 8]. Just as the UseCase model in requirement modeling, the SLN metamodel demarcates the basic boundary of the SLN modeling. The UML profile for SLN describes the syntax of the SLN [2, 10]. The modeling tool implementing the modeling language facilitates the SLN model design. The transformation mechanism from the SLNs to the XML schemata enables the SLN browser to understand the UML profiles and enables users to intuitively observe the SLN structure.

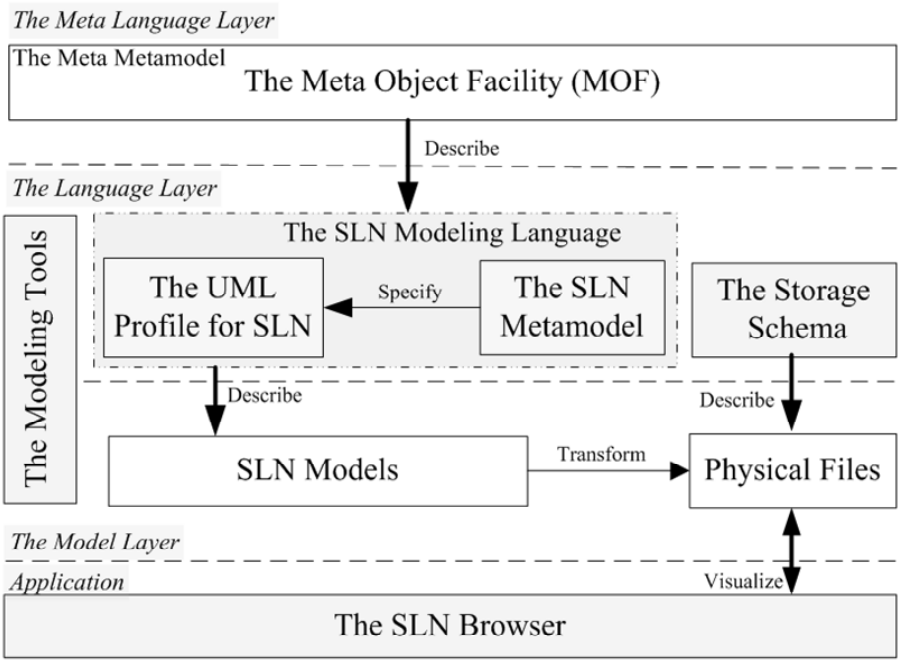


Fig. 1. Techniques on the SLN modeling language.

3. SLN MODELING LANGUAGE

A modeling language consists of a set of syntactic notations and the meaning of these notations.

3.1 The General Algebra Semantics of SLN

A Semantic Link Network consists of a semantic node set *SemanticNodes*, a semantic link set *SemanticLinks* and a reasoning rule set *SLNRules*, denoted as <*SemanticNodes*, *SemanticLinks*, *SLNRules*>. Any semantic link in the *SemanticLinks* is a binary relation between two semantic nodes in the *SemanticNodes*. For any three semantic nodes *A*, *B* and *C* in the semantic node set if there exist two semantic links $A—\alpha\rightarrow B$ and $B—\beta\rightarrow C$ in the semantic link set, and there exists a semantic link rule $X—\alpha\rightarrow Y$, $Y—\beta\rightarrow Z \Rightarrow X—\gamma\rightarrow Z$ (denoted as $\alpha\cdot\beta=\gamma$ in simple) in the *SLNRules*, then $A—\gamma\rightarrow C$ can be derived out and added to *SemanticLinks* [20]. Two semantic link networks can be merged into one by common nodes or by adding semantic links between nodes of different networks.

The Semantic Link Network can be extended by including richer rules in the SLNRules to support inductive reasoning and analogical reasoning [24].

3.2 The SLN Metamodel

A metamodel at the abstraction level describes a specific modeling domain such that newly developed models are the instances of the metamodel. To define a metamodel for the SLN modeling language,

essential semantic elements should be defined and extended to satisfy the structure constrains of the modeling language. Fig. 2 is the MOF-based SLN metamodel.

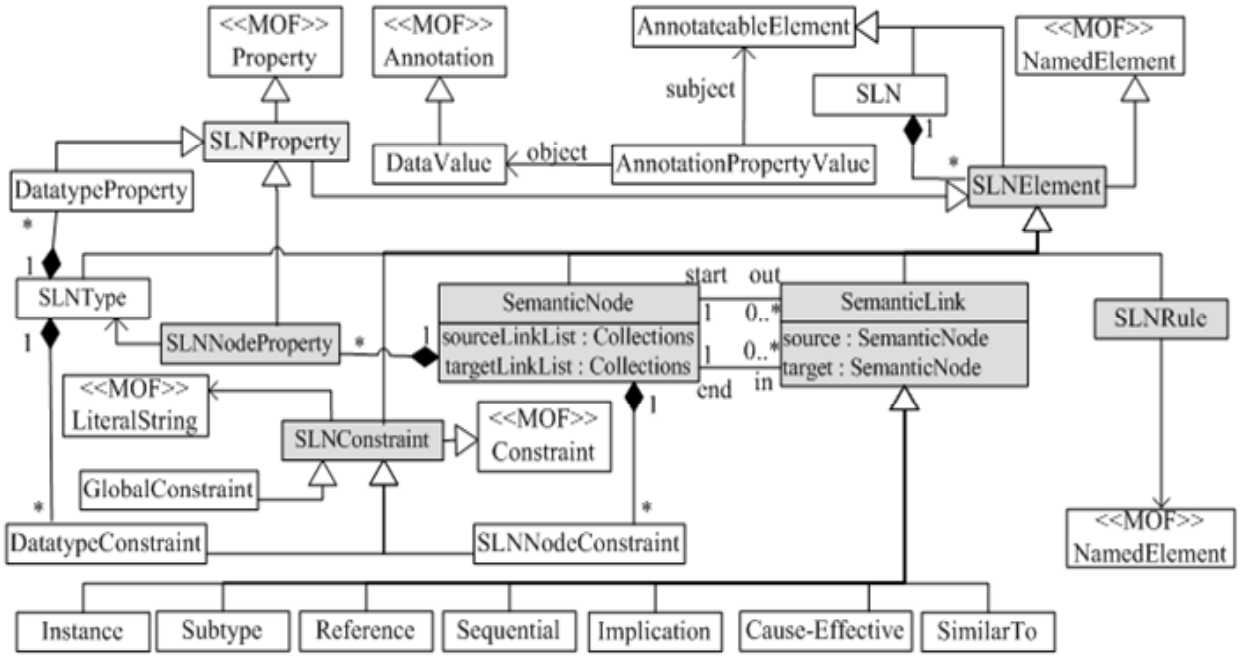


Fig.2. The MOF-based SLN metamodel.

The meaning of the SLN elements are illustrated as follows [19-24]:

SemanticNode. *A SemanticNode* describes the characteristics of the specific semantic nodes representing a type of resources. *SemanticNodes* are linked by *semanticLink*s. A *SemanticNode* owns several properties together with property constrains defined by *SLNNodeConstraint*s.

SemanticLink. A *SemanticLink* represents a binary association between two *SemanticNodes* as its source end and target end. With the extensibility of the SLN metamodel, users can define new semantic link types to extend the core metamodel.

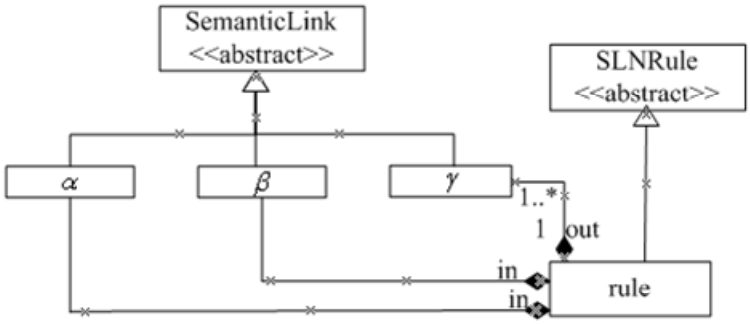
SLNProperty. A *SLNProperty* represents the properties of the *SemanticNode*. It inherits MOF element Property and owns two sub-elements: *DatatypeProperty* and *SLNNodeProperty*. A *SemanticNode* owns several *SLNNodeProperty*s. The *AnnotationPropertyValue* represents a property value. For an *AnnotationPropertyValue*, its *subject* is an *AnnotateableElement* and its *object* is a *DataValue* inherited from the MOF *Annotation*.

SLNType. A *SLNType* can have several *DatatypeProperty*s. It can be an integer, a string or a specific semantic node.

SLNConstraint. A *SLNConstraint* inherited from the MOF *Constraint* restricts SLN properties. It owns a *LiteralString* expression as its value. Three categories of constraints are *GlobalConstraint*,

DatatypeConstraint and *SLNNodeConstraint*. For example, the constrain forEachNode(node.sourceLinkList.size()+node.targetLinkList.size()>0) indicates that no isolated node exists in a SLN. *SLNNodeConstraint* specifies property type and values. For example, constrain node.targetLinkList.size()>0 means that the semantic node is accessible from at least one node.

SLNRule. A *SLNAlgebraRule* extends the *SLNRule* to capture the features of the algebra-based reasoning. Each rule owns several reasoning expressions according to the MOF *LiteralString*. Fig. 3 illustrates the formal expression of reasoning rules. For example, the algebra-based specific reasoning rule: r—*ce*→r1, r1—*imp*→r2 ⇒ r—*ce*→r2 can be represented as a *LiteralString ce*imp* ⇒ *ce*, where r, r1 and r2 represent semantic node, *ce* represents *cause-effective* link, and *imp* represents *implication* link. Users can extend the *SLNRule* according to domain requirements.



SLNRule	α	β	γ
rule1	*ce*	*imp, st, sim, ins*	*ce*
rule2	*imp*	*st, ins, ce, ref*	*imp*
rule3	*st*	*ce*	*ce*
rule4	*st*	*imp*	*imp*
rule5	*st*	*ref*	*ref*
rule6	*st*	*ins*	*ins*
rule7	*ins*	*ce*	*ce*
rule8	*ins*	*imp*	*imp*
rule9	*ins*	*ref*	*ref*

Fig. 3. The metamodel of SLN rules.

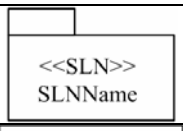
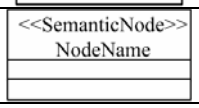
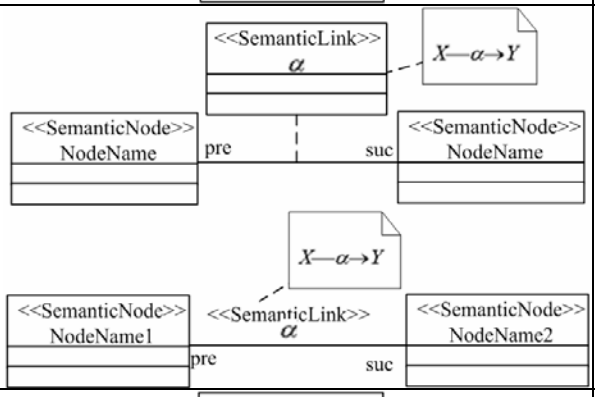
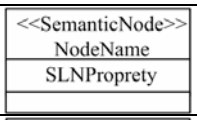
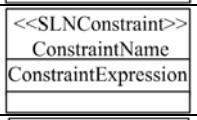
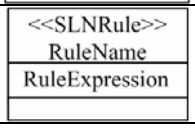
3.3 UML Profile for SLN

A UML profile provides a generic extension mechanism for building UML models in particular domains. UML profiles are based on additional stereotypes and tagged values that are applied to elements,

attributes, methods, links, and link ends. A UML profile is a collection of such extensions and restrictions for describing some particular modeling problem and facilitating modeling constructs in a domain.

The UML profile for SLN employs stereotype and tagged value to annotate SLN semantics in the SLN modeling syntax [7, 14]. Table 1 indicates the mapping between SLN features, UML features, and UML Profile features.

Table 1. The mapping between SLN features, UML profile features, and UML Profile features.

SLN features	**UML features**	**UML Profile features**	**UML Diagram Syntax**
Namespace	Package	<<SLN>>	
SemanticNode	Class	<<SemanticNode>>	
SemanticLink	Binary Direct Association	<<SemanticLink>>	
Property	Attribute	SLNProperty	
Constraint	Constraint	<<SLNConstraint>>	
Rule	Association Class	<<SLNRule>>	

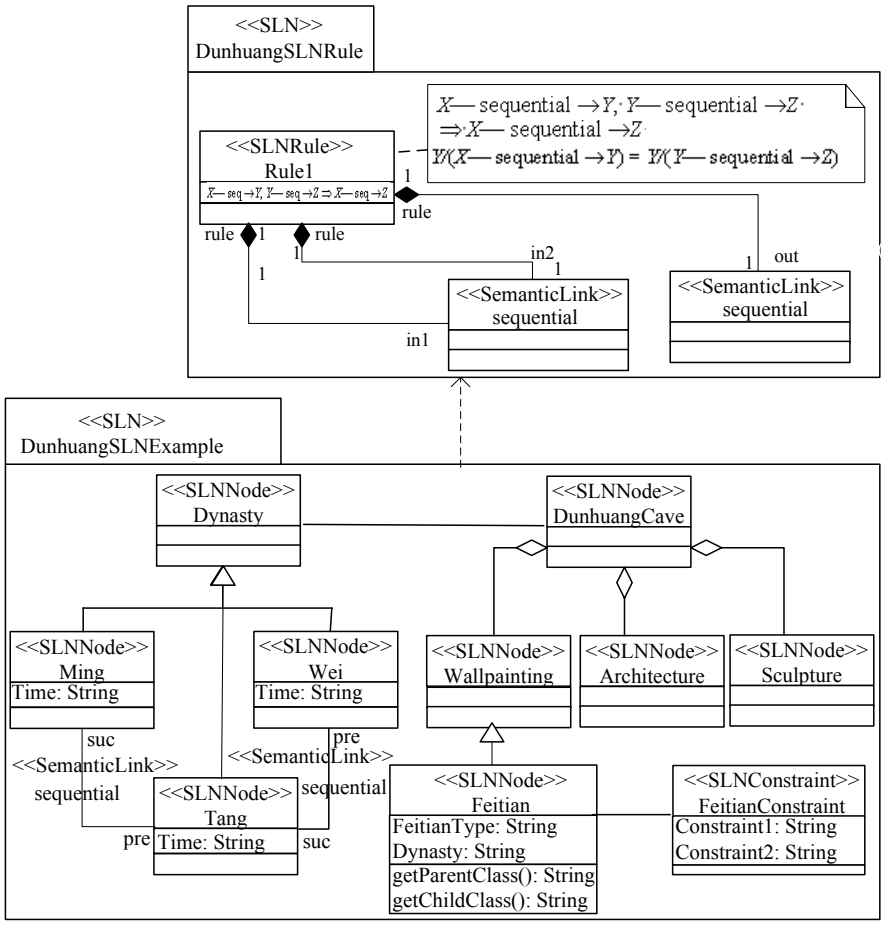


Fig.4 Dunhuang cave culture modeling with UML Profile for SLN.

We have used the UML profile for SLN to construct a Dunhuang cave culture SLN model as shown in Fig4. Two UML package with stereotype "<<SLN>>" were used to organize the SLN model *DunhuangSLNExample* and the inference rule set *DunhuangSLNRule* on this model. The package of the SLN model depends on the package of the rule set.

The package of the SLN model records dynasty information and Dunhuang cave information. The node *Dynasty* is the super type of *Ming*, *Tang* and *Wei* dynasities. The node *DunhuangCave* is the aggregation of *Wallpainting*, *Architecture* and *Sculpture*. The node *Feitian* is a subtype of *Wallpainting*, and it owns the constraint *FeitianConstraint* on its properties. The semantic nodes of SLN models are represented by the UML classes with the stereotype <<SLNNode>>. The SLN properties can be recorded as the properties of these classes. The semantic link "subtype" in SLN models is represented by the UML inheritance relation, while the other semantic links should be represented by the UML association with the stereotype <<SemanticLink>>. The <<SLNConstraint>> constraint is connected with the restricted <<SLNNode>> via the UML association.

In the package of SLN reasoning rules, a rule is represented by the class marked with <<SLNRule>>. The semantic links of SLN model are represented as the association classes and marked with the stereotype << SemanticLink>>. Several <<SemanticLink>> classes as the rule parameters are associated with a <<SLNRule>> class. A rule instance exists if a set of nodes and semantic links satisfies the precondition of a rule. There may be several rule instances in a SLN model for the same rule. In this example, "*Ming* —sequential→ *Tang*, *Tang*—sequential→ *Wei* ⇒ *Ming* —sequential→ *Wei*" is a rule instance of the rule "*X*—sequential→*Y*, *Y*—sequential→*Z* ⇒ *X*—sequential→*Z*".

4. TOOLS

4.1 The Modeling Tool

The aforementioned metamodel and UML profile are used to develop a modeling tool on the *eclipse* platform. The tool facilitate the SLN modeling process. Each component is implemented as a service. The model information and the graphical information are maintained in separate files. The tool supports synchronization of the SLN models and its physical countparts.

When the canvas is initialized, a root controller and two command stacks are assigned. One stack saves "undo" command, and the other saves "redo" command. Actions of users are translated into the corresponding requests and processed by the *editpolicy* module. Each model element owns its own model, controller and render. When the "create" command is triggered, the root controller initializes the corresponding controllers to render the model elements on the tool canvas according to the model definition. The SLN on Dunhuang culture shown in Fig.5 is established by this visual modeling tool.

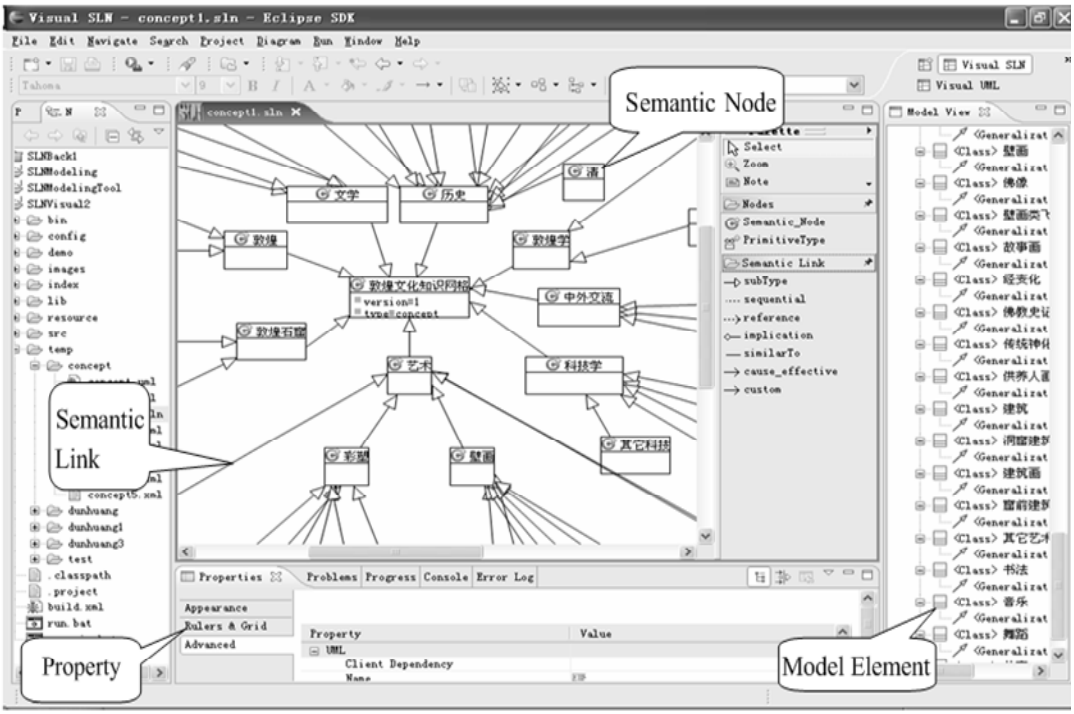


Fig. 5. The SLN modeling tool based on the SLN metamodel and UML profile.

The following is the XML schema used as the common format for the SLN tools.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="graph">
    <xs:complexType>
      <xs:all>
        <xs:element name="semantic_node">
          <xs:complexType>
            <xs:attribute name="id" type="xs:integer" use="required"/>
          <xs:attribute name="name"type="xs:string"use="required"/>
            <xs:attribute name="label" type="xs:string" use="optional"/>
            <xs:anyAttribute namespace="##any"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="semantic_link">
        <xs:complexType>
          <xs:attribute name="label" type="xs;string" use="required"/>
        <xs:attributename="source"type="xs:integer"use="required"/>
       <xs:attributename="target"type="xs:integer"use="required"/>
        <xs:anyAttribute namespace="##any"/>
        </xs:complexType>
        </xs:element>
      </xs:all>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Based on the validation mechanism of the SLN models, the modeling tool supports the transformation mechanism from the SLN model to the SLN XML file. The following algorithm checks the validity of a SLN model. If a SLN is valid, then the semantic nodes are unique and each semantic link has a source node and a target node.

Algorithm 1: validate the UML profile of SLN

```
Procedure validate()
{
    Initialize the umlfile of current SLN;
    Sparse the umlfile to get SLN resources;
    for ( each element N in resources )
        {
            if ( type(N) == ClassImp )
                {
                    Generate semantic node with nodeID and node information;
                    Add semantic node N to node_list;
                    Add semantic node N to semantic node set node_set;
                    for ( i=0; i < N.sourcelinklist.size()+N.targetlinklist.size(); i++)
                        add semantic node N to mapping_nodelist;
                }
        }
    // assure the names of semantic nodes are unique
    If ( size(node_list) <> size(node_set) ) return false;
    for ( each element L in resources )
        {
            If ( type(L) == AssociationImp )
                {
                    Get source node and target nodes of L;
                    Remove source node of mapping from mapping_nodelist;
                    Remove target node of mapping from mapping_nodelist;
                }
        }
    // assure the SLN is connected
    If ( size(mapping_nodelist() <> 0 )) return false;
    return true;
}
```

The following algorithm checks the validity of a SLN and stores the valid SLN into an XML file. The XML contains the semantic node and semantic links information.

Algorithm 2: Store the SLN model with XML

```
Procedure transform()
{
    validate();
    Initialize the umlfile of current SLN;
    Sparse the umlfile to get SLN resources;
    for ( each element N in resources )
        {
            if ( type(N) == ClassImp )
                {
                    Generate semantic node with nodeID and node information;
                    Add semantic node N to node_list;
                }
        }
    for ( each element L in resources )
        {
```

```
            if ( type(L) == AssociationImp )
                {
                    Generate semantic link of L;
                    Insert the semantic link into link_list;
                }
        }
    Store semantic nodes and semantic links into the XML file;
}
```

The following algorithm constructs a tree in depth first order and shows the semantic relevant nodes and links.

Algorithm 3: Construct Tree in Depth First Order

```
Procedure treeConstruction(SemanticNode root)
{
    for(each node n    links to root)
        {
            add n to the node_list;
            add links between n and root to the link_list;
        }
    // assure the nodes and links construct a tree
    if ( size(node_list) - size(link_list) <> 1) return false;
    else
        {
            for ( each node K in node_list )
                {
                    if ( outdegree(K) <> 1 ) return false;
                    else
                        {
                            if ( indegree(K) == 0 ) return false;
                            else
                                {
                                    dfTreeEstablish(K);
                                }
                        }
                }
            return true;
        }
}

Procedure dfTreeEstablish(SemanticNode activeNode)
{
    for ( each link L in link_list )
        {
            if ( targetNodeOF(L) == activeNode )
                {
                    dfTreeEstablish ( sourceNodeOF(L) );
                    Remove semantic link L from link_list;
                }
        }
}
```

4.2 The SLN Browser

The SLN browser adopts a loosely coupled architecture for multiple types of display and interactions as shown in Fig. 6. This architecture consists of three layers according to the MVC (Model, Controller, and Viewer) design pattern. In this case, the SLN model represents the MVC Model. The MVC Controller translates the MVC Model into the runtime rendering, i.e., the MVC View. The display module of the SLN browser realizes the rendering of graphs. The User Interface (UI) module deals with the user interactions and forwards the interaction requests to the listeners of the SLN browser. These listeners associate with the MVC viewers and reply specific requests from the UI module in the run time.

The SLN browser adopts visualization technologies such as Degree-Of-Interest tree (DOI) and hyperbolic tree [2, 16]. The following algorithm constructs a tree structure used by DOI and hyperbolic tree layout according to the depth first order.

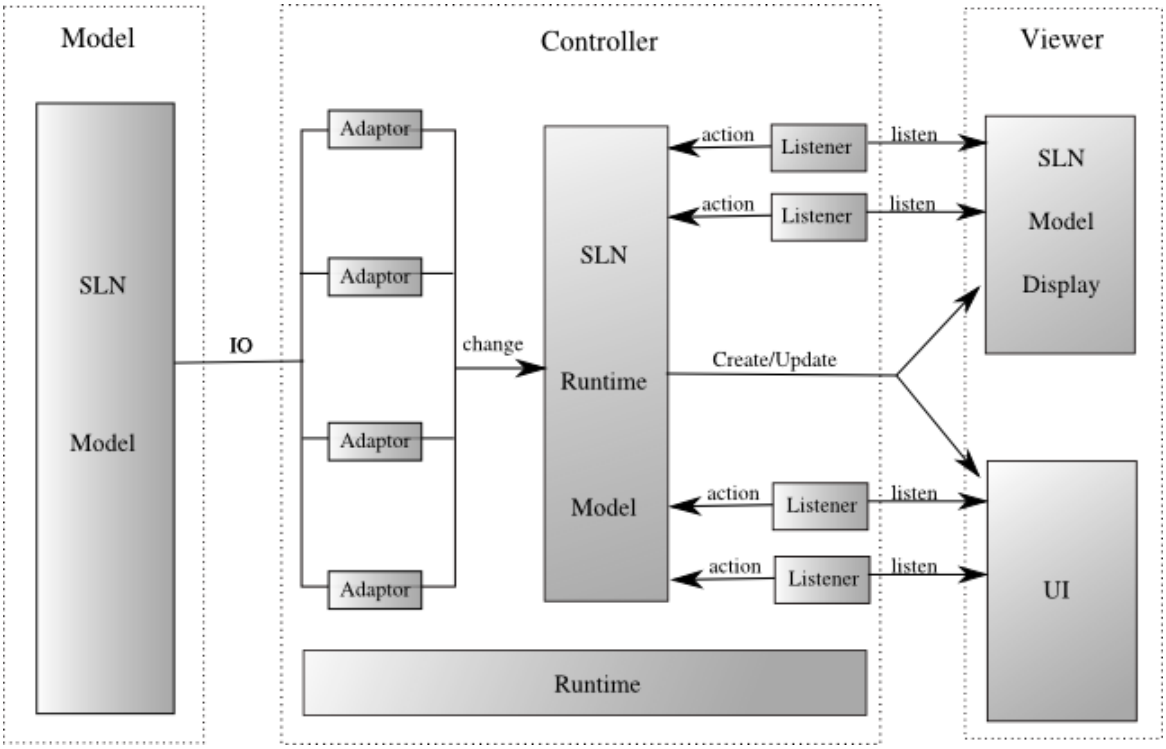


Fig. 6. The architecture of the SLN browser.

5. APPLICATION

The SLN modeling language and support tool have been applied to establish the semantic overlay on various Dunhuang cultural resources in forms of image, text, video and audio in the Dunhuang Cultural

Exhibition System. As shown in Fig.7, resource entities are located at the bottom, while the concept schema at the top layer annotates these resource instances. Here are 76 concepts and 126 resource instances. An additional link *belongTo* was introduced to extend the primitive links to help researchers study the content similarities and identifying the implied links between resources.

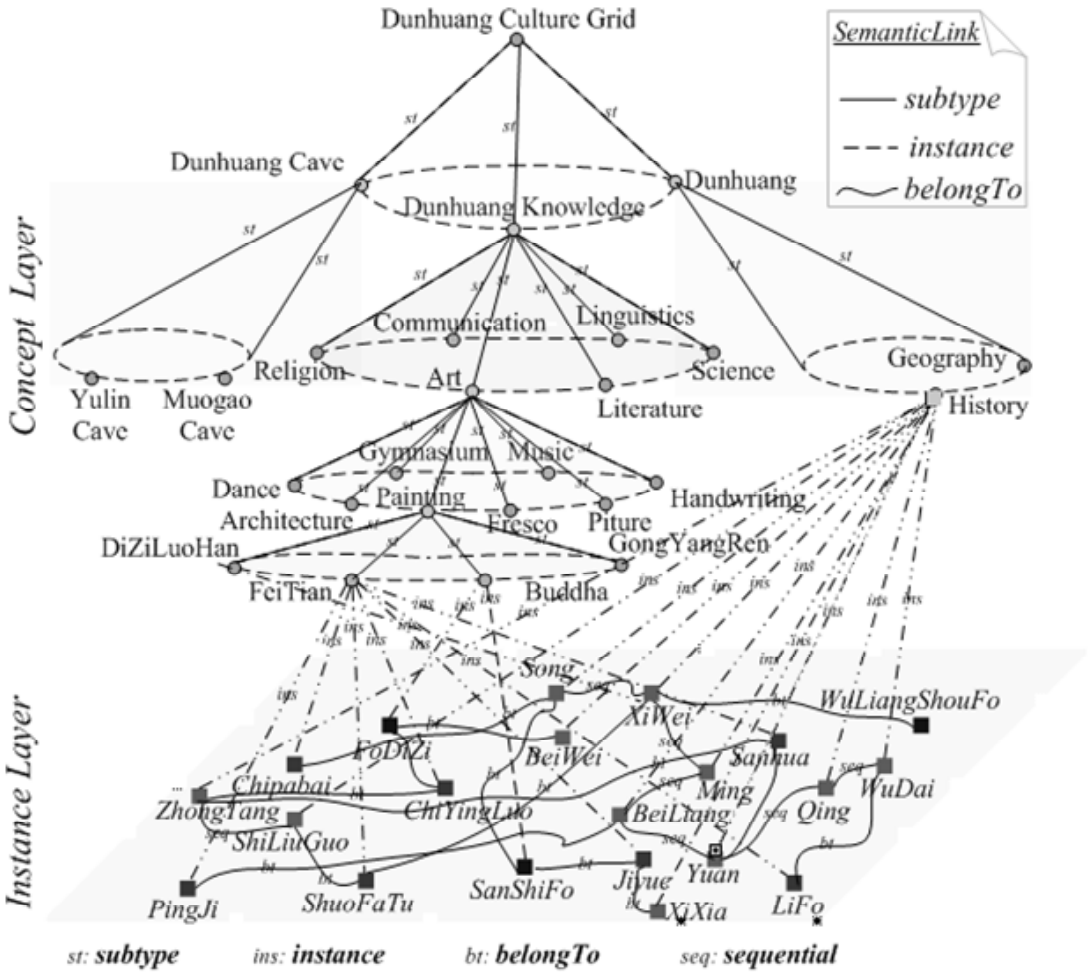


Fig. 7. Concept schema and instances of Dunhuang SLN.

The SLN browser loads the Dunhuang SLN segment file and provides the navigating and searching services for users. As shown in Fig.8, it displays the SLN in the main portion and the relevant content on the right column. The user can search by inputting keywords in the bottom line and selecting the scale of the SLN relevant to the input by moving the distance pointer. It suggests heuristic clues and verifies the rationality of suggestions according to semantic link reasoning. It enables users to see the result in form of a network of highlighted semantic linked resources.

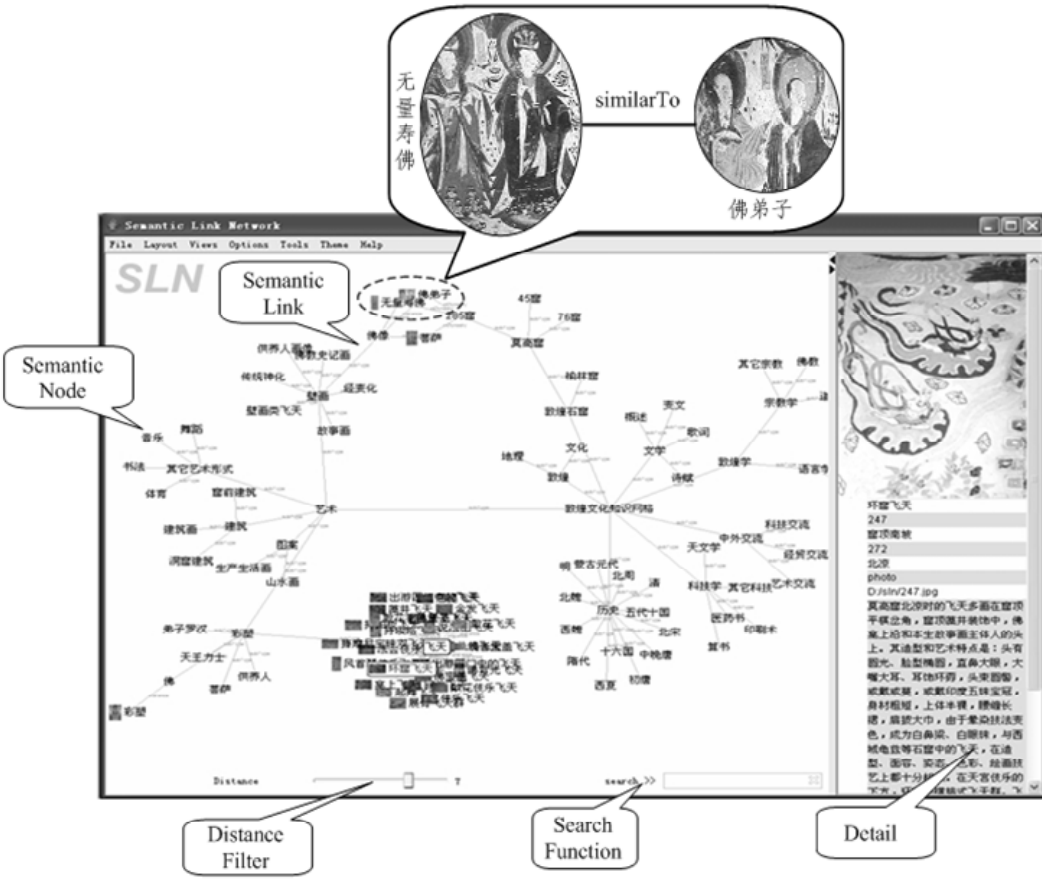


Fig. 8. The Dunhuang culture SLN shown by the SLN browser.

6. DISTRIBUTED SLN BUILDING AND SHARING MECHANISM

The establishment of a large-scale SLN is a social behavior, so it is very important to develop a mechanism to support the building and sharing of SLNs. Sharing extends individual ability of building SLN and helps interconnect different SLNs. Fig.9 presents a Client/Server networking infrastructure that facilitates the cooperative SLN development and sharing [20].

Users can download the SLN modeling tool from the host server by searching the software list in the central directory server and use it to define SLNs, store them at the client and upload the defined SLNs to the host servers and register in the directory server for sharing with others. Users can also download the SLN browsers from the host server in the same way and use it to browse local SLNs. Any newly uploaded SLN and software need to be registered in the server. For large-scale applications, the central resource lists can be distributed onto several servers. Moreover, the SLN servers can be organized in a fully decentralized manner [25].

With the upload and download behaviors, the global and local SLNs become richer and richer. The SLN browser has the following functions:

(1) *The editing and reasoning functions*. When several SLNs are loaded in the SLN builder, SLNs with common nodes are merged. Therefore new semantic links can be derived out according to the semantic linking rules and added to the merged SLN. Users can edit the composed SLN according to their requirements.

(2) *The searching and recommendation functions*. The SLN browser will analyze the description of the nodes and find relevant SLNs in the resource list and recommend an appropriate one to the user by matching the context of the node (the neighboring concepts and the connected semantic links) and the SLNs. The user can store the recommended SLNs at the client and link them to the local SLNs.

Ontology is the basis of connecting SLNs. Fig. 10 shows the interface of an OWL-based SLN builder. Users create SLNs with the builder at the client side. The loaded SLNs are displayed in the builder's window. SLNs designed by different people can be merged according to the ontology and saved as a new SLN. Node information is listed in the right portion. The node and the semantic link information can be edited by users. Nodes can be labeled with concepts in domain ontology. If different ontologies are used, mapping between ontologies are needed [15]. After the definition of the SLNs, users can upload them onto a SLN server to share with others.

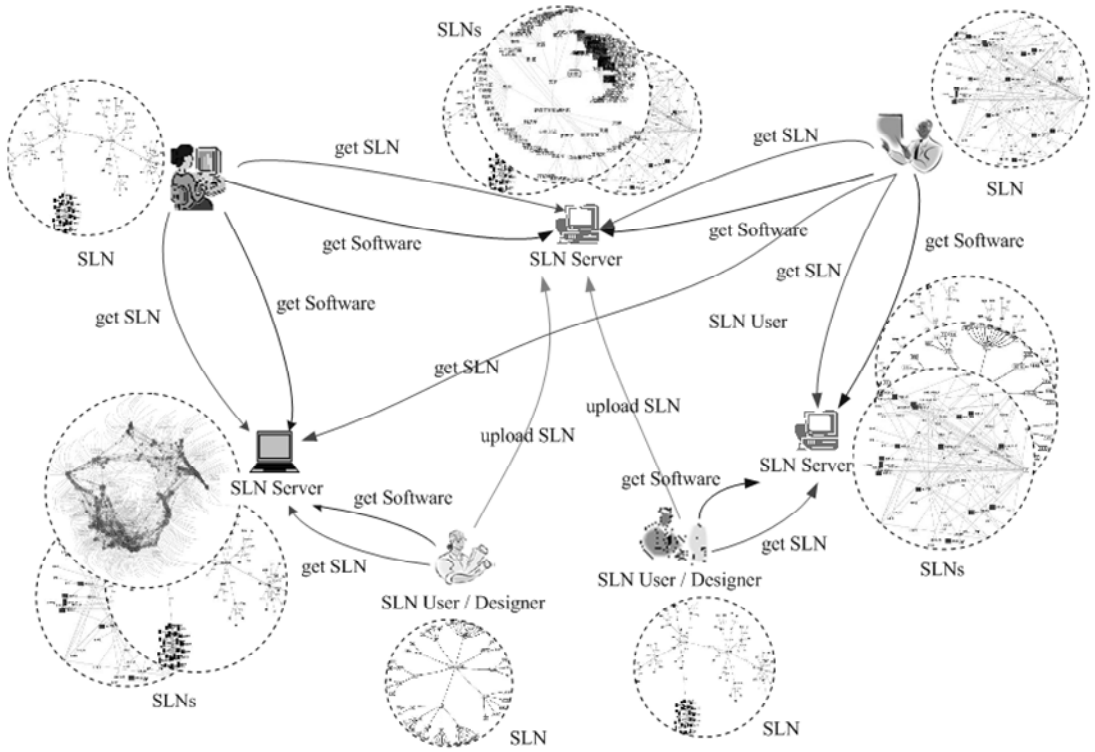


Fig.9. The SLN resource sharing infrastructure.

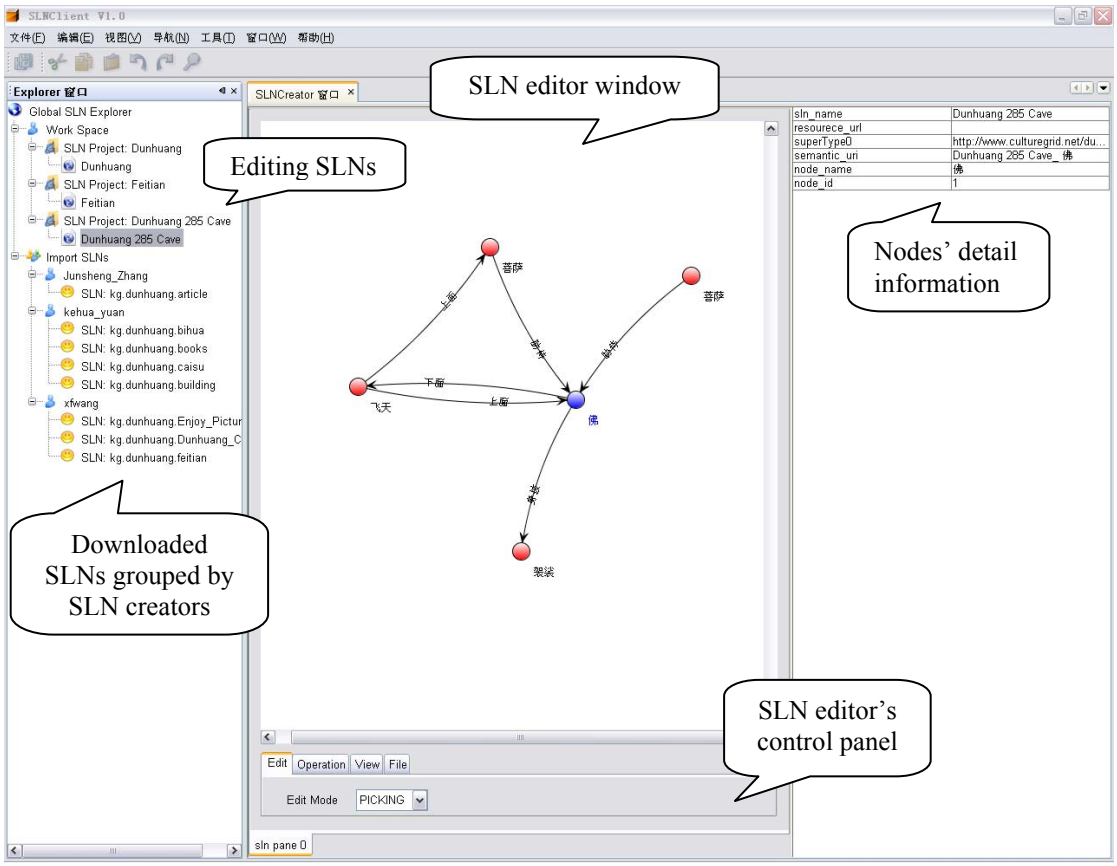


Fig 10. The interface for building and sharing SLNs.

7. CONCLUSION

The proposed modeling language consists of the SLN algebraic definition, the SLN metamodel and the UML profile for SLN. The metamodel specifies the primitives of the SLN modeling. The UML profile enables users familiar with UML to represent domain knowledge by using SLN. The advantage is that people with basic UML knowledge can use SLN without knowing extra notation. Given a set of domain-specific semantic link and the linking rules, the modeling language supports the definition of any domain-specific SLN. The SLN builder implements the modeling language and facilitates the definition of SLN. The graphical SLN browser enables users to intuitively observe the semantic relations. The distributed SLN building and sharing mechanisms suggests a solution to realize a Semantic Linked Web.

ACKNOWLEDGEMENTS

The research work is supported by the National Basic Research Program (973 project no.2003CB317001), the International Cooperation Project of Ministry of Science and Technology of China (Grant no.

2006DFA11970), and EU 6th Framework Project GREDIA (IST-FP6-034363).

REFERENCES

[1] C. Atkinson, and T. Kühne, Model-Driven Development, *A Metamodeling Foundation, IEEE Software,* 20(5) (2003) 36-41.
[2] K. Baclawski, et al. Extending the Unified Modeling Language for Ontology Development, *International Journal Software and Systems Modeling,* 1(2) (2002) 142-156.
[3] T. Berners-Lee, et al., A Framework for Web Science, *Foundations and Trends in Web Science*, 1(1)(2006)1-130.
[4] S. Card and D. Nation, Degree-of-interest trees: a component of an attention-reactive user interface, *Advanced Visual Interface*, (2002) 231-245.
[5] S. Cranefield, M. Purvis: UML as An Ontology Modelling Language, the *16th Intl. Joint Conf. on Artificial Intelligence*, (1999) 46-53.
[6] S. Cranefield, Networked Knowledge Representation and Exchange Using UML and RDF, *Journal of Digital information,* 1(8) (2001).
[7] S. Cranefield, UML and the Semantic Web, *the International Semantic Web Working Symposium*, Palo Alto, (2001) 113-130.
[8] V. Deursen, et al. Domain-Specific Languages: An Annotated Bibliography*, ACM SIGPLAN Notices,* 35(6) (2000) 26-36.
[9] K. Duddy, UML2 Must Enable A Family of Languages, *Communications of the ACM*, 45(11) (2002) 73-75.
[10] A. Felfernig, et al. UML as Domain Specific Language for the Construction of Knowledge Based Configurations Systems, *International Journal on Software Engineering and Knowledge Engineering*, 10(4) (2000) 449–470.
[11] D. Gašević, et al. Applying MDA Standards in Ontological Engineering, *International Conference on Information Technology*, (2003) 193-196.
[12] D. Harel and B. Rumpe, Meaningful Modeling: What's the Semantics of "Semantics"?, Computer, 36 (10) (2004) 64-71.
[13] K.H.Mak. Jeffrey, et al. Precise Modeling of Design Patterns in UML, *the 26th International Conference on Software Engineering,* (2004) 252-261.
[14] C. Jonker, et al. A Visual Language for the Domain Knowledge of Desire, *Workshop on Knowledge Acquisition, Modeling and Management*, (1998) 18-23.
[15] Y. Kalfoglou and M. Schorlemmer, Ontology mapping: the State of the Art, *The Knowledge Engineering Review*, 18 (2003) 1-31.
[16] J. Lamping and R. Rao, The Hyperbolic Browser: A Focus + Context Technique for Visualizing Large Hierarchies, *Journal of Visual Languages and Computing*, 7(1) (1996) 33-55.
[17] S. A. McIlraith et al. Visual Modeling of OWL DL Ontologies Using UML, *International Semantic Web Conference*, LNCS 3298, (2004) 198-213.
[18] A.P.Pons, Object Prefetching Using Semantic Links, *The Data Base for Advances in Information Systems*, 1(37)(2006)97-109.
[19] H. Zhuge, Active E-Document Framework ADF: Model and Tool, *Information and Management*, 41(1) (2003) 87-97.
[20] H. Zhuge, The Knowledge Grid. World Scientific, Singapore, 2004.
[21] H. Zhuge and R. Jia, Semantic Link Network Builder and Intelligent Browser, *Concurrency and Computation: Practice and Experience*, 16 (14) (2004) 1453 -1476.
[22] H. Zhuge, Retrieve Images by Understanding Semantic Links and Clustering Image Fragments, *Journal of Systems and Software*, 73 (3) (2004) 455-466.
[23] H. Zhuge, Semantic Component Networking: Toward the Synergy of Static Reuse and Dynamic

Clustering of Resources in the Knowledge Grid, *Journal of Systems and Software*, 79 (2006) 1469-1482.

[24] H. Zhuge, Autonomous Semantic Link Networking Model for the Knowledge Grid, *Concurrency and Computation, Practice and Experience*, (2007) 1065-1085.

[25] H. Zhuge and X. Li, Peer-to-Peer in Metric Space and Semantic Space, *IEEE Transactions on Knowledge and Data Engineering*, 6(19) (2007) 759-771.

[26] H. Zhuge, The Open and Autonomous Interconnection Semantics. Keynote at ICEC 2006, Proceedings of the 8th International Conference on Electronic Commerce, Fredericton, New Brunswick, Canada, Aug.13-16, 2006. pp.105-115. ACM Press.

Technical Report of Knowledge Grid Research Center, KGRC-2007-03, June, 2007, www.knowledgegrid.net/TR