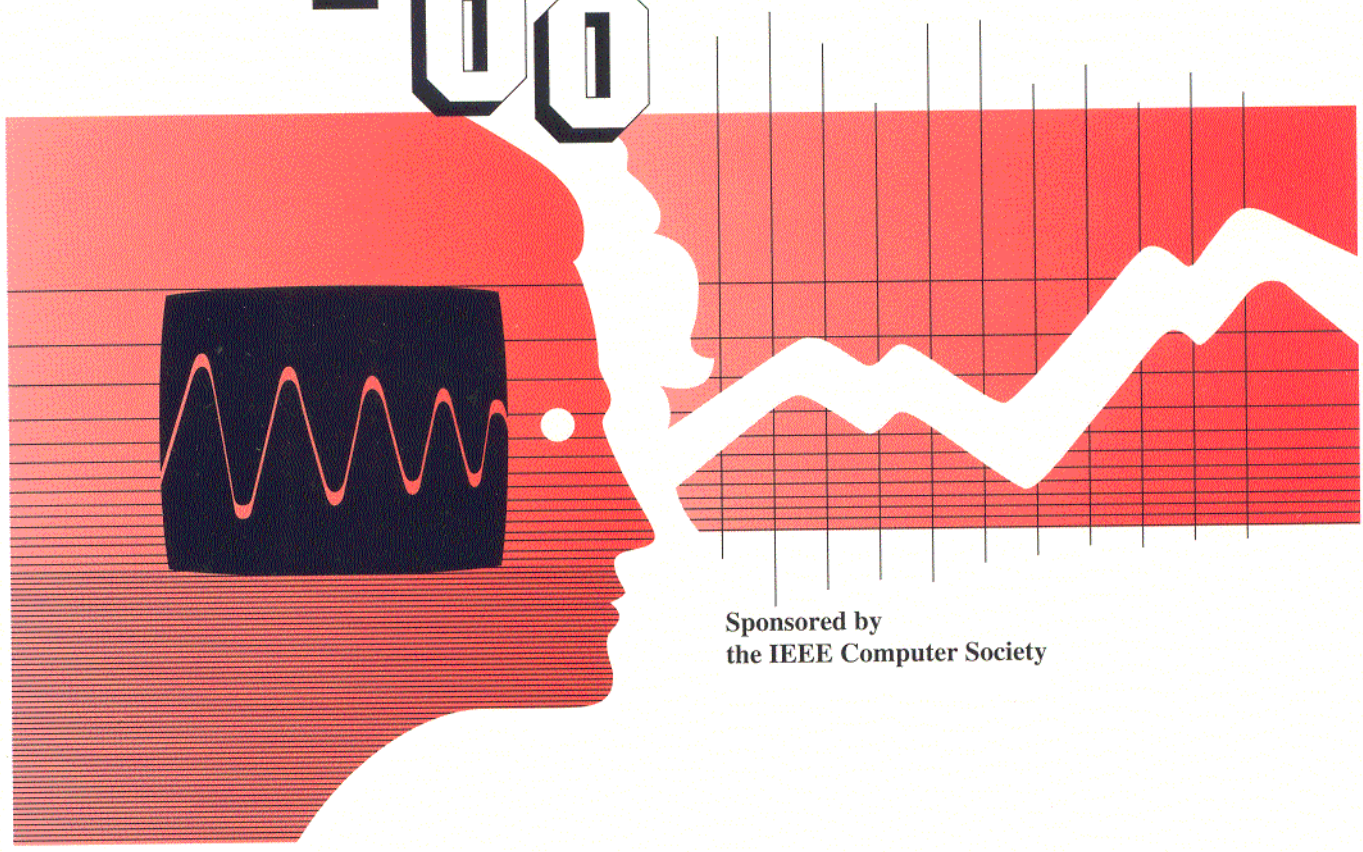
Proceedings

Twelfth International Conference on TOOLS WITH ARTIFICIAL INTELLIGENCE

ICTAI 00

13-15 November 2000
Vancouver, British Columbia, Canada



Sponsored by
the IEEE Computer Society

IEEE COMPUTER SOCIETY

Proceedings of the Twelfth International Conference on Tools with Artificial Intelligence, ICTAI 00, IEEE Computer Society Press
13-15 November 2000, Vancouver, British Columbia, Canada, ISBN 0-7695-0909-6, ISSN 1082-3409

A Distributed Multimedia Knowledge Based Environment for Modeling over the Internet

Stephen W. Ryan, Arvind K. Bansal[1], and T. Kapoor
Department of Mathematics and Computer Science
Kent State University, Kent, OH 44242 - 0001, USA
E-mail: {sryan, arvind, tkapoor}@mcs.kent.edu

Abstract

This paper describes a knowledge-based scalable multimedia environment for graph based modeling and the design of complex objects over the Internet. A complex object is modeled as a directed hierarchical graph with each sub-component abstracted as a node and the shared parameters between two components as an edge. The knowledge base archives and retrieves reusable components, and integrates multiple simultaneous distributed numeric simulations of components over the Internet. The tool exploits heterogeneous associative logic programming [1] and has been implemented using CORBA to exploit heterogeneity and user transparent distributed object-based computing. Java has been used for an architecture independent graphical user interface accessible via the World Wide Web. The design is visualized using XML extended to visualize graph-based multimedia objects over the Internet.

Keyword: CORBA**,** design, distributed computing, graph, Internet, knowledge base, logic program, modeling, multimedia, XML

1. Introduction

As the Internet becomes faster and more integrated, it has become a viable option to extend cluster based computing over a local area network to include the computers distributed over the Internet. Major advantages of the faster Internet are:

1. large knowledge bases can be distributed over the Internet in a user transparent way,
2. multiple computations can be done simultaneously over distributed resources, and
3. computing resources are significantly enhanced

Many applications can benefit from this enhancement in the Internet capability. One such application is a user-friendly distributed multimedia knowledge based tool to facilitate design of complex systems.

A complex system consists of many subsystems, which themselves are composed of even smaller subsystems. The design of a complex system involves modeling the individual subsystems. The simulation of individual subsystems has to be integrated to achieve the design of the complex system. This process can be undertaken using either a top-down or bottom-up approach. In the top-down approach, the designer outlines the overall system and then details the specifications of the various subsystems. In the bottom-up approach, the system is designed by assembling existing sub-component designs.

The design process is iterative: subsystems are repeatedly adjusted or redesigned until the components achieve the desired input/output parameter specifications. Previously designed components are reused with minor modifications to interface with different subsystems. A large repository of components of already designed subsystem is created and archived in a repository. Components are designed in two ways:

1. new engineering systems are simulated numerically using data from experimentation or from mathematical program "codes"; and
2. components designed previously are retrieved from the knowledge base using input/output parameters, interactively modified, and rearchived into the repository.

In this paper, we describe a knowledge based environment that

1. facilitates graph-based modeling to design objects using multimedia knowledge retrieval, and

[1] Corresponding Author

2. integrates various simulated subsystems on a distributed computing platform over the Internet.

This system aims to develop an user friendly environment for the analysis and design of aircraft engines by integrating existing numerical codes for engine components [2, 5, 9] to create an end-to-end engine simulation in realistic time. In this paper, we will use examples from NASA engine simulation project NPSS [5] to illustrate our system.

The software features a graph-based user interface for specifying and connecting components, CORBA-based communications [16] for distributing the simulation of the various subsystems across a network, and an XML [19] based visualization tool extended (in this work) to visualize a graph representation of multimedia objects.

The paper is organized as follows. Section 2 briefly describes the underlying theory of distributed knowledge base system over the Internet and basic definitions needed for a graph based representation of complex objects. Section 3 describes the overall model of the multimedia knowledge base system. Section 4 describes the details for graphical design of complex objects over the Internet. Section 5 describes the CORBA based implementation for the graph based design. Section 6 describes the implementation of multimedia capability in the knowledge base system. Section 7 describes the front end interface for the graph based design. Section 8 describes a grammar to extend XML for the graph based visualization of multimedia objects. Section 9 describes the related work. The last section concludes the work.

2. Background

Logic programs expresses knowledge in a declarative and easily modifiable manner. A complex system is modeled using logical facts and rules [17].

Example 1

The following program illustrates the modeling of a part of an aircraft engine:

```
part_of(engine, turbofan).
part_of(turbofan, rotor).

contains(X, Z) :- part_of(X, Z).
contains(X, Z) :- part_of(X, Y), contains(Y, Z).
```

Information is retrieved from the given knowledge base, by posing a query of the form: *contains*(*engine, W*)? The query yields a set of values {*turbofan, rotor*}.

We have modeled multimedia knowledge over the internet using *distributed heterogeneous associative logic programming* [14, 15] which integrates associative computing, logic programming, and heterogeneous distributed computing for a scalable distributed knowledge retrieval system.

Associative computing [12] exploits content based matching to retrieve a subset data items by specifying possibly incomplete and unordered information related to a set of data items. The content based matching exploits data parallel search on the entire data set. A data parallel search executes efficiently on high performance parallel architectures and distributed ensemble of computers.

The *distributed heterogeneous associative logic programming system* [15] distributes multimedia knowledge on a heterogeneous cluster (including high performance processors) in a user-transparent way. A coordinating process ties multiple distributed multimedia knowledge bases together into a single seamless virtual knowledge base. An individual knowledge base resides in an *associative knowledge server*. Each *knowledge server* runs on a distinct computer node. Within each server, knowledge is stored in a tabular format. Data is represented as an indexed association of fields (optimized during compile time analysis) to facilitate search by content. Low-level details are available elsewhere [1].

After initialization of a system, a *knowledge base server process* loads the tabular data and the compiled rules, and reports to its coordinator with a list of the solvable goals. The coordinator builds a virtual knowledge base by combining the individual capabilities of all the servers. A coordinator then participates in even larger virtual knowledge bases by acting as a server (see Figure 1).

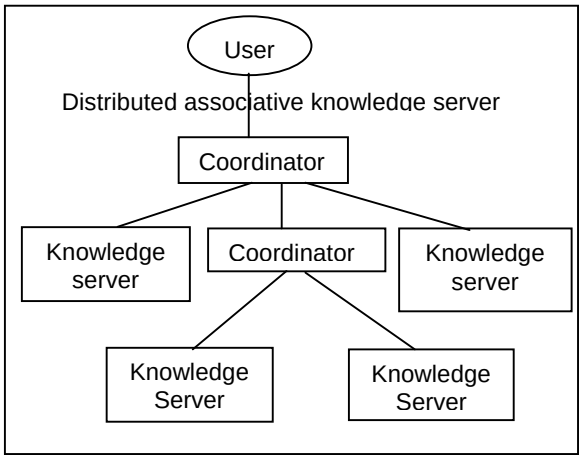


Figure 1. The distributed knowledge model

2.1. Definitions

A *hierarchical graph* (possibly directed) [4] has multiple layers of abstraction. Each level of abstraction joins a cluster of nodes (at the lower level of abstraction) having a common attribute/function into a single node. Each node at a higher level of abstraction is a simple node or

corresponds to a hierarchical graph at the lower level of abstraction. The edge between two nodes V_I and V_J at a higher level of abstraction represents one or more edges between the embedded subgraphs corresponding to the nodes V_I and V_J.

Example 2

Figure 2 illustrates a hierarchical graph with two levels of abstraction: Figure 2a gives the actual graph, and Figure 2b gives the abstracted graph. In Figure 2a, the nodes in the subgraph G_1 and the subgraph G_2 are connected through the edges E_1 and E_2. In Figure 2b, the subgraph G_1 is abstracted as the node V_1, the subgraph G_2 is abstracted as the node V_2, and the edge (V_1, V_2) is a composite edge abstracting the set of edges $\{E_1, E_2\}$.

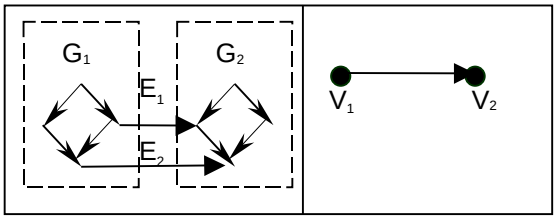


Figure 2a. A graph Figure 2b. Abstraction

3. Distributed modeling system

The overall system is given in Figure 3. The numerical simulation of engine components is done at the cluster of numeric nodes distributed over the Internet, and the virtual knowledge base is distributed over the Internet. The overall system works as follows:

The coordinator has a top level engine design program. To design a complete engine, the individual input/output parameters of the sub-components are computed. If a design of any sub-component is archived in the virtual knowledge base, then the design and the input/output specifications of the sub-component are displayed to the user. The user interactively modifies the design, simulates specific sub-component on a numeric node, and archives the new design in the knowledge base.

A user models a component using hierarchical graphs: each sub-component is a node representing a hierarchical graph at a lower level and each edge represents a set of interface parameters between the two sub-components. For example, an edge could represent a two dimensional table where the first column represents the values of the output from the source node, and the second column represents the input parameters at the sink node.

These graphs are compiled in a user-transparent manner to a set of logical rules at the user end, transmitted to the server end, and finally archived in the knowledge base.

A server sends the design as graphs represented in XML extended to visualize graphs. The use of extended XML provides a natural interface with other Internet based scripting languages.

4. Hierarchical graph based design

In this section we describe the design of an engine component using hierarchical graphs. The user defines the complex system as a set of hieracial graphs representing the overall system and subsystems. In the logic program representation, the nodes are mapped to predicates and edges map to shared variables between the corresponding predicates.

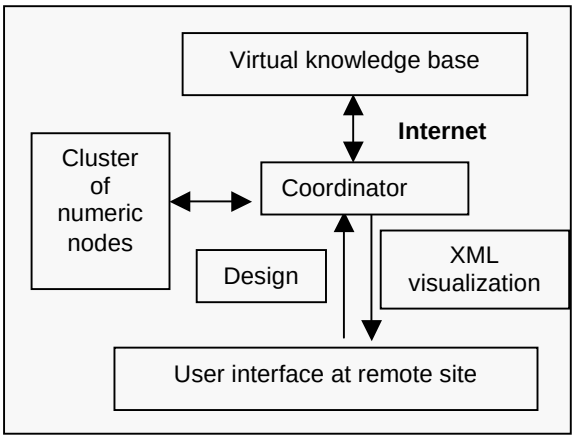


Figure 3. The overall design system

4.1. Designing combustion engine

Figure 4 is the top level of a hierarchical graph that abstracts the operation of an aircraft engine [13].

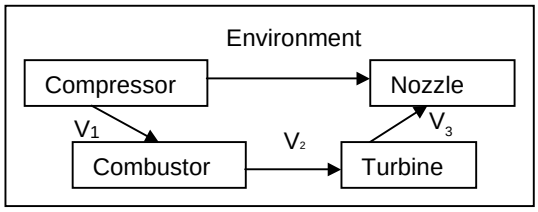


Figure 4. Graph-based design of an engine

In this graph, the *compressor* and the *nozzle* share a common variable *Environmen*t. The *compressor* and the *combustor* share a common variable V_1 which represents mixing volume V_1. A mixing volume is an abstraction to describe the interconnecting attributes between the components in a physical system. It includes values such as mass, temperature and pressure. The *turbine* shares mixing volume V_2 with the *combustor* and mixing volume V_3 with the *nozzle*. These sub-components are modeled in

more detail by expanding nodes *compressor*, *combustor*, *turbine* and *nozzle* into subgraphs.

The system translates a graph-based representation of a modeled system into a set of logical rules. The graph in Figure 4 is converted into the following logic rule.

```
engine(Environment):-
    compressor(Environment, V1),
    combustor(V1, V2),
    turbine(V2, V3),
    nozzle(V3, Environment).
```

4.2. Translating hierarchical graphs

Hierarchical graphs have been transformed into a set of logical rules using an object-oriented approach (see Figure 5).

```
Algortihm compile_graph
Input:      A heirarchcial graph
Output: A set of logical rules

{   let the set of logical rules P be •;
    for each graph in the graph set
        for each vertex in the graph
        {   Set S = " ";
            S += vertex.LPhead( );
            if there is an embedded subgraph
            {   S += ":-" ;
                S += graph.LPtail( );
                S += "." ;}
            P = P ∪ S;}
        return P;}

vertex.LPhead( )
{   S += node-label;S += "(";
    for each edge incident upon the node
    {   S += edge-label;
        if this is not the last edge
            S += "," ; }
    S += ")" ;}
    return S;}

graph.LPtail( ):
    {   S += " " ;
        for each node in the graph
                { S += "," ; S += vertex.LPhead( ) ;}
        return S;}
```

Figure 5. Compiling hierarchical graph

The implementation uses three classes: *graph-class*, *vertex-class*, and *edge-class*. The *graph-class* is a triple of the form (*graph-id,*, a *list of vertices, a list of edges*). The *vertex-class* is a pair of the form (*a list of incident edges, a reference to a subgraph representing the corresponding sub-component*). The *edge class* is a pair consisting of references to the two connecting vertices.

To parse a graph, every vertex is represented as a logical rule. A vertex-label forms a predicate name, edge-labels of the incident edges form the arguments of the predicate, and the mapping of an embedded subgraph representing a subsystem makes the right hand side of the rule. A vertex not mapped to any embedded subgraph, maps to a logical fact.

A method *vertex.LPhead*( ) constructs the predicate from a vertex object. A method *graph.LPtail*() builds predicates recursively corresponding to the embedded subgraphs. The method *graph.Lptail*( ) calls *vertex.LPhead*( ) for each vertex in a subgraph to create the set of corresponding logical rules.

5. Distributed simulation

The graph representation naturally exploits distributed simulation: embedded subgraphs of a hierarchical graph reside, and are simulated on separate host computers. Meta-information such as the specific server or class of servers used to solve a subgoal are stored in the properties dialog of the various nodes in the graph.

5.1 CORBA based implementation

The current system is implemented using CORBA as the middleware. CORBA provides user transparency and heterogeneity for distributing servers across the network. In a CORBA based system, all potential servers define their capabilities using a standard Interface Definition Language (IDL), and publish their availability to an Object Request Broker (ORB). This interface, which encapsulates the operation of a knowledge server, is flexible enough to access other types of servers. This flexibility facilitates the use of a large library of pre-existing CORBA-compliant server objects without a need to recompile or modify the objects. A simple code stub translates between the interface of a CORBA object and the standard server interface. The naming and lookup facilities of the ORB are used to find the appropriate server object regardless of the server type. This enables the use of various types of modules, including numerical engineering codes that model physical subsystems such as the aircraft engine described in Figure 4.

Figure 6 illustrates a sample IDL and typical capabilities of a knowledge server.

```
module knowledge_server {
    interface KnowledgeServer
    {   short load_program(in string file);
        short load_code(in string code);
        short solve(in string goal);
        short more();
        short next();
        string get_bindings();
    };
```

Figure 6. A sample IDL for a knowledge sever

The invocation of the method *load_program* causes a server to load a compiled knowledge base from a file. The method *solve* submits a query to the server. The method *get_bindings* retrieves a set of solutions. The methods *more* and *next* retrieve additional solutions by controlling backtracking in the knowledge servers.

6. Multimedia knowledge capability

Multimedia knowledge is represented in the knowledge base by mixing the uniform resource locators (URL) of the HTML file, video file, or the audio as arguments to a fact in the knowledge base. For example, we can code different types of jet engines as a set of facts, and every different type of engine becomes a unique index for the picture of the engine as illustrated below:

```
jetengine(ge90).        jetengine(pw2000).

enginePicture(ge90,"http://www.geae.com/lrgcom/
        ge90 /images/ge90_cutout.gif").
enginePicture(pw2000,"http://www.prattwhitney.com/
        images/engines/enginegallery/lg.f117.cut.jpg").
```

We illustrate the use of the knowledge retrieval using a small multimedia knowledge base about jet engine manufacturers and their engines. The engine companies have web sites on the Internet that have information about their products, including images of the engines themselves. This system can access this distributed multimedia knowledge by creating a knowledge base server and querying the knowledge transparently.

The knowledge base contains two types of facts and one rule. Facts of the form *makes_engine*(*Maker*, *Engine*) associate the various engines in the knowledge base with the manufacturers that make them. Facts of the form *enginePicture*(*Engine, URL*) mark the location on the Internet where the image of an engine is archived. The rule

```
engineFrom(Maker, Picture) :-
    makesEngine(Maker,Engine),  enginePicture(Engine,
    Picture).
```

matches the manufacturers in the multimedia knowledge base with images of the engines on the Internet.

Figure 7 illustrates the result of executing a query against such a knowledge base. In this example, the query *engine_from*(*pratt, X*)? retrieves and displays a series of images of engines from the internet in a user transparent manner.

7. User interface implementation

The graphs and graphical user interfaces have been implemented using the Java programming language [3] due to the following reasons:

1. Java is an architecture independent language that provides a portable graphical user interface, supports visualization of multimedia objects, and accesses resources on the Internet.
2. The use of Java allows the user interface code to be mobile. By constraining the functionality (file system access for example), the front end can be loaded on demand as an applet from any site with a Java-enabled web browser.

Figure 7. A multimedia query using the Internet

The structure of the Java front end is illustrated in Figure 8. The graphical user interface (GUI) contains a typical windows-based interface for viewing the graph and text. There are menus to abstract a hierarchical graph to higher level, to get low level details of a hierarchical graph, to compile hierarchical graphs into the corresponding logical rules, and to launch processes to solve a query. The user is able to draw a hierarchical graph interactively using the front end. A hierarchical graph is abstracted or detailed by referring to the unique identifier of the subgraph corresponding to a node at a higher level of abstraction.

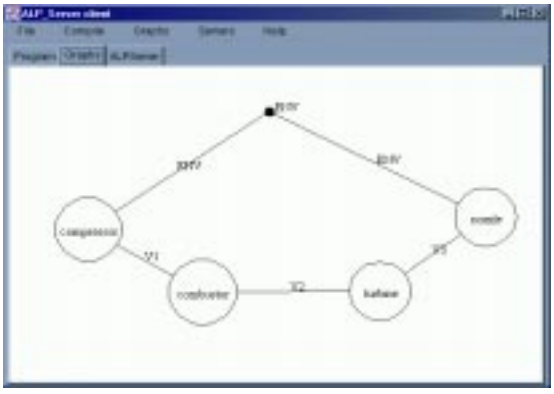


Figure 8. The user interface

Beneath the user interface there are modules to convert the graphs to logic programs, to parse query, to compile a logic program to tabular representation and abstract instruction code, and to communicate with the knowledge servers. A schematics is given in Figure 9.

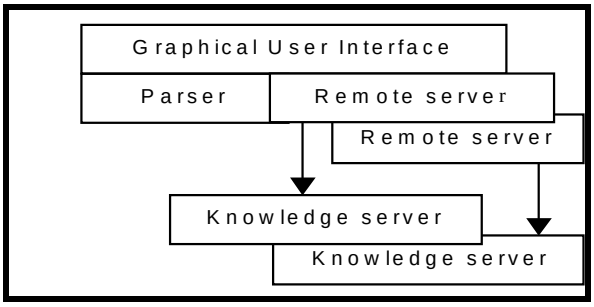


Figure 9. The structure of Java based front end

8. Visualization of hierarchical graphs

In this section, we describe a grammar to extend XML [19] for graph based visualization over the Internet. An object is represented as a set of one or more graphs. Each graph has zero or more attributes and one or more edges. The graph attribute could be *graph-id, default color of edges, default color of nodes, default zoom size* of nodes. *Graph-id* provides the reference from a node to the corresponding subgraph embedded at the lower level of abstraction. By placing graph-id within a node, the node is expanded (when needed) to the corresponding subgraph. Each edge has a label. On each edge or a node, a multimedia object is superimposed by including type:<Type> attribute where <Type> is defined an audio clip, video clip, image, user-defined object, a graph , or a multi dimensional table.

```
<Object>::= {<G>}+
<G> ::= ‘<’graph  {<Graph-Attribute>}*  ‘>’ {<E >}+
          ‘</ ‘graph ‘>’
<E> ::= ‘<’edge  {<Edge-Attribute>}*’>’< N> <N>
         ‘</’Edge’>’
<N> ::= ‘<’node  {<Node-Attribute>}* ’>’ ‘</’node’>’
<Graph-Attribute> ::= graph_id:<Number> | edge_color:
          <Color>| node_color:<Color> | size: <Number>
<Edge-Attributes> ::=  edge_id:<Number> | color:
          <Color> |  label: <Text> | type:<Type> |
          edge_dir: <Dir>
<Node-Attributes> ::=  node_id:<Number> | type:<Type:> |
          label= <String> | x:<Number> | y:<Number> |
          color: <Color> | size:< Number> | style: <Style>
<Direction>::= directed | undirected
<Type>::= video | image | view |audio| graph | table |
           user_defined |
<Style> ::= solid | circle
<Number>::= <Digit>+
```

Figure 10. A grammar to visualize graphs in XML

8.1 An example of XML based graph

Figure 11 illustrates an example of the extended XML representation of the graph presented in Figure 8.

```
<?xml version ="1.0" encoding="UTF-8"
standalone="no"?>
<!DOCTYPE graph SYSTEM "graph.dtd">
<graph graph_id="0" edge_color="black"
node_color="black">
<edge edge_id ="1" label="ENV" edge_dir="directed">
<node node_id="1" label="ENV" style="solid" size="5"
x="300" y="100" > </node>
<node node_id="2" label=”compressor” style="circle"
size="35" x="100" y="200"> </node>
</edge>

<edge edge_id ="2" label="V1" edge_dir="directed">
<node node_id="2" label=”compressor”> </node>
<node node_id="3" label=”combustor” style="circle"
size="35" x="200" y="315">  </node>
</edge>

<edge edge_id ="3" label="V2" edge_dir="directed">
<node node_id="3" label=”combustor” > </node>
<node node_id="4" label=”turbine” style="circle" size="35"
x="400" y="315"> </node>
</edge>

<edge edge_id ="4" label="V3" edge_dir="directed">
<node node_id="4" label=”turbine” > </node>
<node node_id="5" label= ”nozzle” style="circle" size="35"
x="500" y="200"> </node>
</edge>

<edge edge_id ="5" label=  “ENV” edge_dir="directed">
<node node_id="5" label= “nozzle” > </node>
<node node_id="1" label= “ENV”> </node>
</edge>
</graph>
```

Figure 11. A graph visualization in extended XML

This text is transmitted over the Internet to visualize the graph at the user’s end. At the user’s end the extended XML code is parsed [11] and displayed using a JAVA interface.

9. Related works

As previously mentioned, NASA’s NPSS project is developing a distributed system for the analysis, design and simulation of aircraft engines. The approach is to reuse the existing base of engineering codes by standardizing on the CORBA distributed object protocol and making those existing codes interoperable. The work described in this paper is complementary to NPSS project since we are using knowledge based system to enhance component reuse by archiving previously designed subsystems and to integrate multiple subsystems using symbolic reasoning. Our system can easily be interfaced

with the NPSS software library due to the use of CORBA protocol for distributed execution.

10. Conclusions

In this paper, we have described a distributed knowledge based environment for the design a complex object by reusing previously designed components and by integrating numerically simulated subsystems over the Internet. The system exploited associative logic programming for content based search, ease of modifiability of knowledge base, and symbolic reasoning. We also described a graph based representation and visualization of the complex object. We extended XML to visualize graph based multimedia objects over the Internet.

We are extending our system to provide interfaces to already developed libraries of numerical codes in NPSS project to simulate an aircraft combustion engine [5].

Acknowledgments

This research was supported in part by NASA Glenn Research Center through a NASA Grant and Ohio Board of Regent's Ph. D. enhancement grant. We thank Gregory Follen and NASA research team for useful discussions and the continued support of the project.

References

[1] A. K. Bansal, "A Framework of Heterogeneous Associative Logic Programming," *International Journal of Artificial Intelligence Tools, Vol. 4, Nos. 1 & 2*, (1995), pp. 33 - 53.

[2] R. Claus, G. J. Follen, R. Haimes, and W. Jones, *CAPRI - Computational Analysis Programming Interface/ A CAD infra-structure for Aerospace Analysis and Design Simulations*, NASA HPCC/CAS Workshop/NASA Ames Research Center, February 2000.

[3] J. Gosling, B. Joy, and G. Steele, "The Java Language Specification," *Addision-Wesley*, also see http://www.javasoft.com

[4] I. Herman, G. Melançon, and M. S. Marshall, " Graph Visualisation and Navigation in Information Visualization," http://www.cwi.nl/InfoVisu/Survey/StarGraphVisuInInfoVis .html #16592

[5] P. T. Homer and B. Schlichting, "Using Schooner to support distribution and heterogeneity in the Numerical Propulsion System Simulation Project," *Concurrency Practice and Experience*, *Vol. 6(4)*, 1994, pp. 271 – 287

[6] G. Iazeolla and A. D'Ambrogio, "A Web-Based Environment for the Reuse of Simulation Models," *SCS Western MultiConference on Computer Simulation*, San Diego, (1998)

[7] R. McNab. and F. W. Howell, "Using Java for Discrete Event Simulation," *Proceedings of the Twelfth UK Computer and Telecommunications Performance Engineering Workshop (UKPEW)*, University of Edinburgh (1996), pp. , 219 - 228

[8] A. L. Evans et. el., "HPCC NPSS Introduction," http://hpcc lerc.nasa.gov/hpcc2/npssintro.shtml

[9] G. J. Follen and M. auBuchon, *Numerical Zooming between a NPSS Engine System Simulation and a 1-Dimensional High Compressor Analysis Code*, NASA HPCC/CAS Workshop/NASA Ames Research Center, February 2000.

[10] V. Ogle and M. StoneBraker, "Chabot: Retrieval from a Relational Database of Images," *IEEE Computer* 29, (1995), pp. 18 - 22.

[11] R. Pfeifer, ":Parsing XML Using Java," *IBM XML Technology Group Report*," http://www-4.ibm.com/software/developer/education/tutorial-prog/ parsing. html

[12] J. L. Potter, Associative Computing, *Plenum Publishers*, New York, (1992).

[13] J. A. Reed and A. A. Afjeh, "A Java-based Interactive Graphical Gas Turbine Propulsion System Simulator," AIAA paper 97-0233, *35th Aerospace Sciences Meeting and Exhibit*, Reno NV (1997), pp. 1 - 9.

[14]S. Ryan S. and A. K. Bansal, "A Scalable Heterogeneous Associative Logic Programming System," *Proceedings of the Ninth International Conference on Tools with Artificial Intelligence*, Newport Beach, California, (1997), pp. 37 – 44.

[15] S. Ryan and A. K. Bansal, "A Scalable Distributed Associative Multimedia Knowledge Base System for the Internet," *Proceedings of the Eighth International Conference on Intelligent Systems*, Denver, Colorado, (1999), pp. 1 - 6.

[16] J. Siegel, CORBA Fundamentals and Programming, *John Wiley & Sons*, (1996)

[17] L. S. Sterling, and E. Y. Shapiro, The Art of Prolog, *MIT Press*, (1994).

[18] P. Tarau, "Jinni: a Lightweight Java-based Logic Engine for Internet Programming," *Proceedings of JICSLP'98 Implementation of LP languages Workshop*, (1998), pp. 1-15.

[19]Extensible Markup Language (XML), Version 1.0 , *W3C Recommendation* (1998), http://www.w3.org /TR/1998/REC -xml-19980210

[20] D. H. D. Warren, "An Abstract Prolog Instruction Set," *Technical Report 309*, SRI International, (1983).