AIAA 2002- 4058

# Interactive, Secure Web-enabled Aircraft Engine Simulation Using XML Databinding Integration

Risheng Lin and Abdollah A. Afjeh
The University of Toledo
Toledo, Ohio

**38th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit**
7 - 10 July 2002
Indianapolis, Indiana

AIAA-2002-4058

# INTERACTIVE, SECURE WEB-ENABLED AIRCRAFT ENGINE SIMULATION USING XML DATABINDING INTEGRATION

Risheng Lin[*] and Abdollah A. Afjeh[†]
The University of Toledo
2801 West Bancroft Street
Toledo, Ohio 43606, USA

## ABSTRACT

This paper discusses the detailed design of an XML databinding framework for aircraft engine simulation. The framework provides an object interface to access and use engine data, while at the same time preserving the meaning of the original data. The Language independent representation of engine component data enables users to move around XML data using HTTP through disparate networks. The application of this framework is demonstrated via a web-based turbofan propulsion system simulation using the World Wide Web (WWW). A Java Servlet based web component architecture is used for rendering XML engine data into HTML format and dealing with input events from the user, which allows users to interact with simulation data from a web browser. The simulation data can also be saved to a local disk for archiving or to restart the simulation at a later time.

## INTRODUCTION

Computer programs capable of simulating the operation of aircraft engines are useful tools that can help reduce the time, cost and risk of product design and development and facilitate learning about the complex interactions between jet engine components. However, the strongly-coupled nature of the components' flow physics and the large number of operating and design parameters needed for simulation of the aircraft engine system present a challenge to developers who aim at designing an easy-to-use and effective engine simulation program for users. Most of the aircraft engine simulation software currently available have limitations primarily in the presentation of the simulation input and output data, due to the use of text-based interfaces, and the lack of data validation methods. As a result, engine simulation results could be overwhelming and difficult to interpret without a significant effort. Moreover, traditional simulation data are, in general, stored in proprietary data formats and constrained by hardware and operating system platform differences. Thus, developers are hindered in their efforts to synthesize simulation data in their design unless a clearly defined and interoperable data interface exists. The bottlenecks caused by data handling, heterogeneous computing environments and geographically separated design teams, continue to restrict the use of these tools [1].

Web-based simulation, due to its accessibility, convenience and emphasis on collaborative composition of simulation models, distributed heterogeneous execution, and dynamic multimedia documentation, has the potential to fundamentally alter the practice of simulation [2]. Presently, the majority of work in web-based simulation has centered on re-implementation of existing distributed and standalone simulation logics *within* Java Applets [3,4]. Applets are quite popular because they are supported by common browsers and are safe to execute on client computers. However, with the whole simulation code *tightly-bound* to an Applet, it may take a long time for the rich engine simulation code to load within a client's browser. In addition, it is often not efficient to execute complicated simulation logic at the client side, where a high performance computer is generally not available. Applets' security model, arguably one of its strengths, also creates obstacles for post-processing of simulation data beyond what applets provide since it inhibits creation of data files on the host machine.

This paper describes a web-based aircraft engine simulation system, called *X-Jgts*, through dynamic XML databinding framework which permits data communication with ease. XML [5], due to its structured, platform and language independent, highly extensible and web-enabled nature, has rapidly become an emerging standard to represent data between diverse applications. XML can represent both structured and unstructured data, along with its rich descriptive delimiters. By using XML to represent engine data in high performance propulsion system simulation, it is possible to faithfully model the structural elements of a chosen component in an interoperable fashion that is natural in their simulation context. Since HTTP (Hyper Text Transfer Protocol) already supports transmission

[*] Research Associate, Student Member AIAA, Department of Mechanical, Industrial and Manufacturing Engineering. E-mail: rlin@eng.utoledo.edu
[†] Professor and Chair, Department of Mechanical, Industrial and Manufacturing Engineering, Member AIAA. E-mail: aafjeh@eng.utoledo.edu

of plain text, XML data can be moved around readily using the HTTP through firewalls and disparate networks. Engine databinding through XML also provides simulation designers with a higher and more user-friendly API to work with underlying engine components repository and thus enables the components to communicate with each other effectively.

## ENGINE MODELS

This section provides an overview of engine analysis model that is used in our web-based simulation. Also presented is the designed engine data object model that will be used in engine databinding framework.

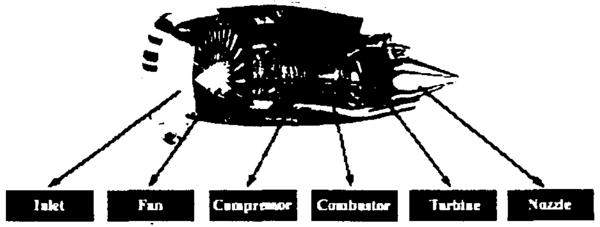
### Analysis Model

The mathematical model used to describe the operation of the gas turbine system in the current work is patterned after that presented in [6]. Here, the gas turbine system is decomposed into its individual basic components: inlet, compressor, combustor, turbine, nozzle, bleed duct connecting duct, and connecting shaft. Intercomponent mixing volumes are used to connect two successive components as well as define temperature and pressure at component boundaries. Operation of each of the components is described by the equations of aero-thermodynamics which are space-averaged to provide a lumped parameter model for each component. For dynamic (transient) gas turbine operation, the model includes the unsteady equations for fluid momentum in connecting ducts, inertia in rotating shafts, and mass and energy storage in intercomponent mixing volumes. A complete description of the model can be found in [7].

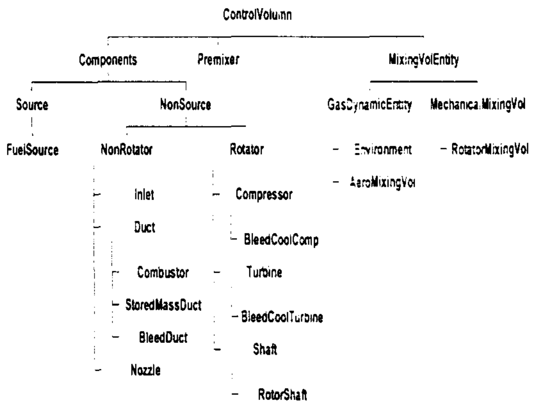
### Data Object Model

Based on the above engine analysis model, an "Engine Data Object" (EDO) model was designed to precisely define the intellectual content of engine component data, including a complete definition of engine data entities, attributes, relationships, and specification of local and global constraints on these entities.

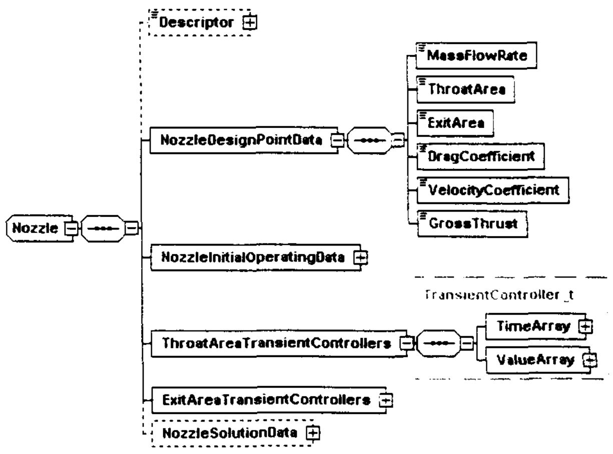
In order to effectively represent simulation data using XML, the engine system, shown in Figure 1(a), was first decomposed into individual basic components in a strict hierarchical manner in accordance with the XML topology. A set of data structures is then built in parallel with each engine component. An overall layout of a simplified data model is summarized in Figure 1(b). Each node in the model shown here is represented as an engine data object. The figure also indicates (informally) what data, if any, are encapsulated within each node object. For example, the *Nozzle* data object shown in Figure 1(c) gives information about a particular converging-diverging or converging-only nozzle in an engine simulation. The user-defined parameters of a nozzle include a set of nozzle design point data and nozzle initial operating data, such as mass flow rate, throat area, exit area, gross thrust. etc. Consequently, these data are designed as subchildren data objects in *Nozzle*. In addition, the nozzle throat and exit areas may be adjusted during the transient by a user-defined schedule; *ThroatAreaTransientControllers* and *ExitAreaTransientControllers* are designed for this



(a)



(b)



(c)

**Figure 1** (a) decomposition of engine component; (b) hierarchical engine data object model; (c) subchildren objects inside nozzle data object purpose. *NozzleSolution* object is used to store the solution datasets after a simulation, which itself contains other children data objects that are not shown here. An optional *Descriptor* object can also be included to describe nozzle operating status.

## ENGINE DATABINDING FRAMEWORK

Based on our data object model design, an Engine Data Binding (EDB) Framework has been implemented in Java to facilitate binding an engine data object into a data entity in XML-based engine data file. The framework makes it easy to convert between the engine data stored in XML file and their object representations, and facilitates the applications to access, modify and store any engine component data object. Figure 2 gives a schematic representation of all components in engine databinding framework. Engine databinding framework can also be run as a standalone application [8].

### Engine Schema

Engine schema establishes a bridge between XML-based engine data and its data object model. It associates each piece of the information defined in the data object model to a precise location in the XML structure. A set of engine schemas have been designed using XML Schema language [9] that specifies how the constituents of the engine data objects are mapped to an underlying XML-based engine data structure. The rules in the data model will guarantee that the schema description of engine data is syntactically correct and also follows the grammar defined within it.

Figure 3 shows a sample schema representation for the *Nozzle* and one of its children, *TransientController*, which is used to supply transient control parameters for throat and exit areas. Based on the Nozzle data model shown in Figure 1(c), the "Nozzle" schema defines all the data elements that are contained in a single nozzle data object. These elements are constrained by their corresponding complexTypes and simpleTypes and encapsulated in the *Nozzle* object. For example, *NozzleDesignPointData* defines all its permitted data variables, such as *MassFlowRate*, *ThroatArea* etc, and their corresponding data types, which are built-in double type. Also note that in the above Nozzle schema only *NozzleDesignPointData* element is explicitly defined, the rest of its element definitions use the "ref" attribute to tell the data parser in the engine simulation that the definition for these elements are defined in other schema files with the same target namespace (i.e, the default "engine" namespace in Fig.3) as nozzle. These 'ref'ed schema will be automatically included by schema parser during the run time. This kind of flexible design will guarantee that all the basic schema types can be reused. Moreover, it will allow for modular development and easy modification of engine schema as engine data object model evolves in the future.

### Schema Compiler

The engine schema compiler is designed to map an instance of an engine schema into the appropriate engine data object model. It *automatically* translates an engine-specific schema into a set of derived engine data object models (set of classes and types which represent the data) with appropriate access and mutation (i.e., get and set) methods that can be used to affect the underlying engine data files. Figure 4 shows an example of how a generated class should correspond to the nozzle schema defined in the previous section. With the "Nozzle" schema defined, attributes are "*compiled*" into simple Java types, usually primitives; element (along with its type information which specifies the content model) becomes engine data class, with

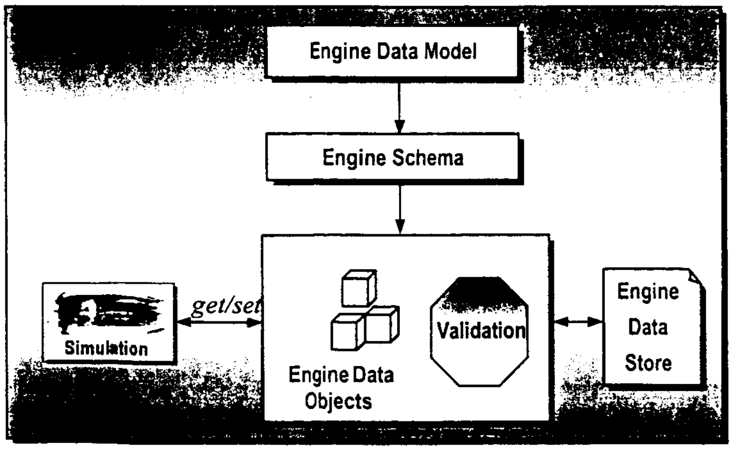


**Figure 2**. Engine databinding framework

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://mems1.ni.utoleco.edu/engine"
    xmlns.xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://mems1.ni.utoleco.edu/engine" elementFormDefault="qualified" version="1.0">
   <xsd:include schemaLocation="TransientController.xsd"/>
   <xsd:include schemaLocation="Descriptor.xsd"/>

   <!-- ComplexType Nozzle_t is designed to constraint Nozzle -->
   <xsd:complexType name="Nozzle_t">
       <xsd:sequence>
           <xsd:element name="Descriptor" type="Descriptor_t" minOccurs="0"/>
           <xsd:element name="NozzleDesignPointData">
               <xsd:complexType>
                   <xsd:attribute name="MassFlowRate" type="xsd:double"/>
                   <xsd:attribute name="ThroatArea" type="xsd:double"/>
                   <xsd:attribute name="ExitArea" type="xsd:double"/>
                   <xsd:attribute name="DragCoefficient" type="xsd:double"/>
                   <xsd:attribute name="VelocityCoefficient" type="xsd:double"/>
                   <xsd:attribute name="GrossThrust" type="xsd:double"/>
               </xsd:complexType>
           </xsd:element>
           <!-- NozzleInitialOperatingData element is similarily designed
               and ommitted here for simplicity-->
           <xsd:element name="ThroatAreaTransCntl"
                   type="TransientCntl_t"/>
           <xsd:element name="ExitAreaTransCntl"
                   type="TransientCntl_t"/>
           <!—All NozzleSolutionData elements and ommitted for simplicity-->
       </xsd:sequence>
       <xsd:attribute name="Name" type="xsd:string" use="required"/>
   </xsd:complexType>
</xsd:schema>
```

```
<?xml version="1.0"?>
<xsd:schema xmlns xsd="http://www.w3.org/2001/XMLSchema"
     elementFormDefault="qualified" version="1.0">

     <!-- TransientController complexType -->
     <xsd:complexType name="TransientCntl_t">
          <xsd:sequence>
               <xsd:element name="TimeArray" type="doubleDatalist"/>
               <xsd:element name="ValueArray" type="doubleDatalist"/>
          </xsd:sequence>
          <xsd:attribute name="name" type="xsd:string" use="optional"/>
     </xsd:complexType>

     <xsd:simpleType name="doubleDatalist">
          <xsd:list itemType="xsd:double"/>
     </xsd:simpleType>
</xsd:schema>
```

**Figure 3**. Engine schema representation of *Nozzle* and *TransientControl* data object model

```
//– all the Java import statements here

public class Nozzle implements java.io.Serializable {
   private String _name;
   private Descriptor descriptor;
   private DesignPointData _nozzleDesignPointData;
   private InitOperatingData _nozzleInitOperatingData;
   private ThroatAreaTransCntl _throatAreaTransCntl;
   private ExitAreaTransCntl _exitAreaTransCntl;
   private NozzleSolutionData _nozzleSolutionData;

   public Nozzle() {
      super();
   }
   public String getName() {
      return this._name;
   }
   public void setName(String name) {
      this._name = name;
   }
   public ExitAreaTransCntl getExitAreaTransCntl() {
      return this._exitAreaTransCntl;
   }
   public void setExitAreaTransCntl(ExitAreaTransCntl  exitAreaTransCntl) {
      this._exitAreaTransCntl = exitAreaTransCntl;
   }

   //– the same with all other types and are omitted here

   public boolean validate()
          throws EngineValidationException   {
       try {
               Validator validator = new Validator();
               validator.validate(this); }
       catch (EngineValidationException vex) {
             return false;
       }
       return true;
   }
   public void marshal(java.io.Writer out)
          throws MarshalException, EngineValidationException {
      Marshaller.marshal(this, out);
   }
   public static Nozzle unmarshal(java.io.Reader reader)
          throws MarshalException, EngineValidationException {
      return (Nozzle)Unmarshaller.unmarshal(Nozzle.class, reader);
   }
}
```

**Figure 4**. *Nozzle* data class generated by schema compiler process

generated data types and properties encapsulated in it. The generated class provides pairs of accessor (*get*) and mutator (*set*) methods for all the properties defined in engine schema, which closely follows the JavaBean Design Pattern [10].

In addition, the engine schema compiler can generate the data 'validation' class code so as to enforce the constraints expressed in the schema. The code generated by the valid schema translation will check that incoming engine data files are 'legal' with respect to the constraints defined in schema, thereby ensuring that only valid XML-based engine data files are produced by the marshalling process.

The generated Java classes also include a set of *marshal*, and *unmarshal* methods that can be used to "translate" engine application data from/to engine data objects automatically. These are achieved through an underlying Marshalling Framework design.

Marshalling Framework

The marshalling framework supports the transportation (*unmarshal*) of XML-based engine data into "graphs" of interrelated instances of objects that are generated by engine schema complier and, in addition, converting (*marshal*) such graphs back into engine data stored in XML documents. The marshal method works by taking a desired *Writer* object as argument and then returning an XML element representation of that object. If the object contains references to other engine data objects, then recursion can be used, using the same method. The same applies to unmarshaling process where a general *Reader* is used. When the engine data are correctly unmarshaled, each element node in the XML file becomes an instance of the data class that was generated by engine schema compiler, i.e. engine data object. Then, the engine simulation components can use the corresponding methods, along with a set of *mutator* and *accesor* methods, to work with the engine data in the underlying data file. The end result is engine data binding.

## SIMULATION ARCHITECTURE

*X-Jgts* is a web-based, interactive, graphical, numerical gas turbine simulator which can be used for the quick, efficient construction and analysis of arbitrary gas turbine systems. It also provides a systematic, meaningful data presentation and secured data operation scheme with the support of a built-in data binding framework. Figure 5 illustrates the overall simulation architecture described in this paper, as well as its major components and the interactions between web client and simulation server.

### Web Client

In *X-Jgts* system, the client user interface is delivered through a web browser. The web browser is a universal user interface that is responsible for presenting engine simulation data, issuing requests to the simulation web server, and handling any results generated at the request of the user. *X-Jgts* uses both dynamically generated HTML and Swing-based Java Applet to properly present user-friendly data; in particular, HTML is used to display simulation results, while Swing-based Applet is used for graphic data display. The platform-independent nature of HTML and Java Applet enables the engine simulation to be widely conducted from heterogeneous, networked computers.

As a general rule for web-based simulation, application logic should not be implemented on the browser. Complex simulation logics that are tightly built into Applets are normally inefficient to execute due to the fact that client side users generally lack powerful computing resource. In addition, it may take quite a long time for a client's browser to load. Therefore, the browser, HTML, and Swing Applets designed in *X-Jgts* are used strictly for delivering the user interface and view into the engine simulation. The user requests are made either from the front-end Applet or HTML code to perform designate tasks remotely in the simulation web server.

### Simulation Server

Engine simulation server is a dynamic extension of a Web server and the heart of any web interactions. It uses HTTP as protocol for communication and consists of static resources, such as the front end simulation Applet, as well as dynamic web pages (HTML) that are generated by different engine web components hosted in the server. The web server listens for incoming requests and then services the requests as they come in. Once the server receives a simulation request, it then springs into action. Depending on the type of request, the web server might look for a web page, or execute a web component on the server. Either way, it will return some kind of results to the web client.

In *X-Jgts*, engine web components are sets of

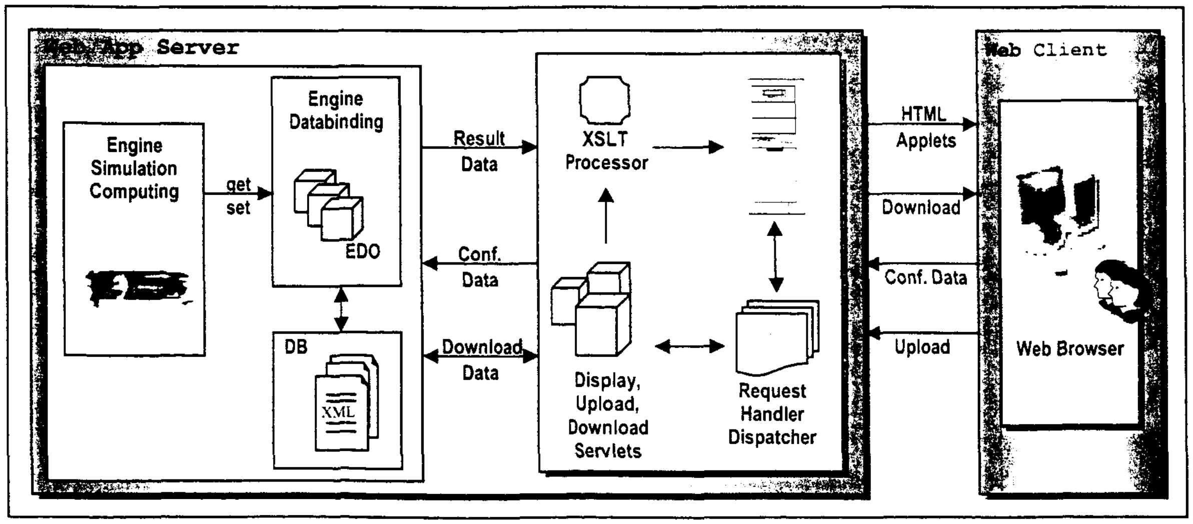


**Figure 5**. Web-based simulation architecture in *X-Jgts*

simulation task-related Servlets [11] or JavaSever Pages [12]. Servlet/JSP provides a platform-independent means of extending a web server's capabilities. When a user issues a request for a specific Servlet, the server will simply use a separate thread and then process the individual request. This has a positive impact on performance.

Engine web components are running in the Tomcat [13] Web container to dynamically process various simulation requests and construct responses. The web container provides services such as request dispatching, security, concurrency, and life-cycle management. Based on different task-related services, engine web components may invoke other web resources directly through embedded URLs that point to other web components while it is executing, or indirectly by forwarding a request to another resource using *RequestDispatcher*. There are four main services currently available in the engine simulation server.

*Simulation Web Component*

Engine simulation service is a core web component that provides a transient, space-averaged, aero- and thermo-dynamic gas turbine analysis for a web client based on the engine analysis model. Besides that, the simulation web component includes the built-in engine databinding support and an underlying XML-based engine database repository to store simulation data (Figure 5). During the engine simulation, the verification logics that are automatically generated by engine schema compiler can be applied inside the simulation so that the users' inputs and simulation outputs could be checked. Engine components can also conveniently manipulate the engine data with a set of *accessor* and *mutator* methods devised from databinding framework. When a simulation completes, engine components can readily *marshal* sets of engine object data into the underlying data repository for storage and *unmarshal* them back to engine data objects later when data manipulation is necessary. This feature gives a very useful and natural way for the storage of any engine data object and provides the engine simulation with unambiguous, meaningful and interpretable representation of engine data sets. The engine simulation service can also generate simulation graphs and transcript data dynamically and send them to the front-end Applet for display.

*File Download Web Component*

*X-Jgts* allows users to save their simulation results to the local file system so that users can redisplay their simulation result or restart simulation at a later time. This is achieved internally by the file-download service. Due to security reasons, current web browsers prohibit the front-end simulation Applet from directly writing data files on the host that is executing it. Nevertheless, Applets can usually make network connections to the host they came from. In *X-Jgts*, whenever a user wants to download a complete simulation result or engine configuration file, the front-end Applet will make a request to file-download service resided on the simulation web server, locate the corresponding case file from database repository and then generate a download response to the user. By setting the HTTP `Content-Disposition` response header as *attachment*, Web browser at client side will pop up a "save as" box to let user save simulation result.

*File Upload Web Component*

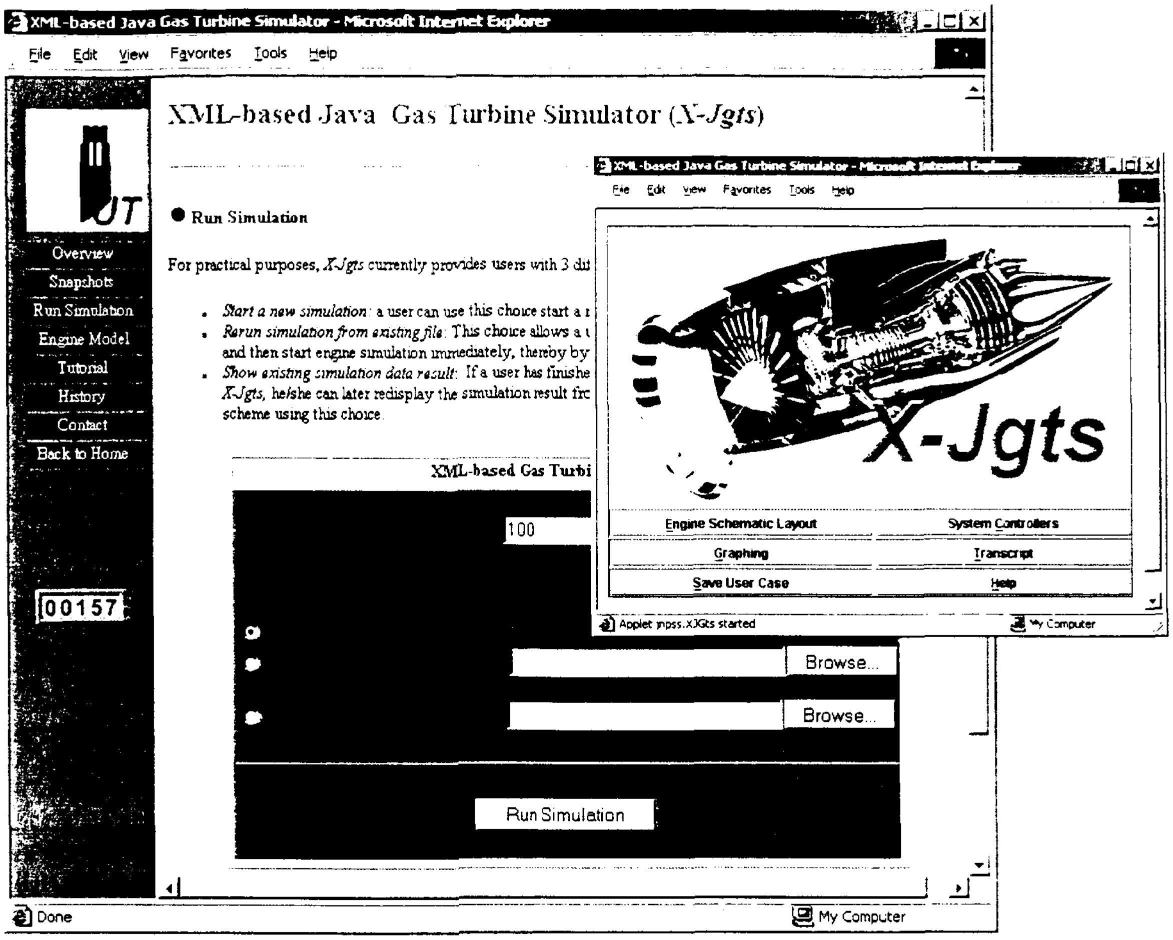
At times users have a requirement to upload a file from their local file system to the web server for display of engine simulation result in a more meaningful way. *X-Jgts* web components include a Servlet that can receive a file upload using its input stream. When a file is sent via a browser, it is embedded in a single POST request with *multipart/form-data* [14] encoding type. The file upload Servlet will take in the part of this multipart data stream, reassembled and encoded on the server, and then dispatch the processing results to display service, where dynamically generated engine data file in HTML format are sent to client's browser for display.

*Display Web Component*

Since engine data are stored in XML file format, it is easier to apply certain transformation logic such that simulation results can be displayed in a more friendly way within the user's browser. XSLT [15] provides a way to transform the engine data without cluttering up the web components code with HTML. When the simulation server receives a display request, the build-in XSLT processor knows how to parse engine component-specific XSLT style sheets and apply transformations. Best of all, a clean separation between engine data, presentation, and simulation logic allows changes to be made to the look and feel of a web site without altering the simulation code. Because XML-based engine data can be transformed into many different formats, it can also achieve portability across a variety of browsers and other devices.

## DEMONSTRATION

Based on the designed data object model, databinding architecture, and simulation architecture, a web-based engine simulation has been implemented that internally uses *Onyx* [16] as the engine simulation logic. Onyx is an object-oriented framework for propulsion system simulation. Figure 6 shows the XML-based Java Gas Turbine Simulator, *X-Jgts*, being accessed from an Internet Explorer browser.



XML-based Java Gas Turbine Simulator (*X-Jgts*)

● Run Simulation

For practical purposes, *X-Jgts* currently provides users with 3 di

- *Start a new simulation*: a user can use this choice start a
- *Rerun simulation from existing file*: This choice allows a
  and then start engine simulation immediately, thereby by
- *Show existing simulation data result*: If a user has finishe
  *X-Jgts*, he/she can later redisplay the simulation result fr
  scheme using this choice.

XML-based Gas Turbi

**Figure 6**. XML-based Java Gas Turbine Simulator accessed from a Web browser

For practical purposes, *X-Jgts* currently provides users with 3 different kinds of simulation services. A simulation identifier (ID) is required to perform each service.

Start a new simulation

A user can use this choice to start a new engine simulation in interactive construction mode. After the user enters a simulation ID, and starts to perform the simulation, the Swing-based Applet interface (Figure 6) will appear. From there the user can access the various main windows of the simulation system: *Engine Schematic Layout*, *System Control Dialog*, *Graphing*, *Transcript*, or *Save User Case*.

Before each simulation is run, the user must provide each individual engine component with initial simulation configuration data from the designed *Engine Schematic Layout Dialog* (see Figure 7). An engine model is developed by building an engine component schematic graphically as *Icons* (e.g., BleedDuct, Nozzle, VariableCompressor, etc.) and connecting them together. In the diagram, the arrowheaded connecting lines represent both the directional flow path for fluid through the engine, and the structural connections along which mechanical energy is transmitted. The user can define the operational characteristics for the component (i.e., the component name, design- and initial-operating point data, etc.) in the engine component's dialog window (Figure 8). The *System Control Dialog* (Figure 9) provides controls for the overall operation of the simulation. The steady-state numerical solver is used to balance the gas turbine equations at the initial operating point as was defined by the user: while transient solvers are used for dynamic engine performance analysis. When the necessary data input for simulation configuration is finished, the simulation can have the option to start simulation immediately or download the configuration file and run it later.

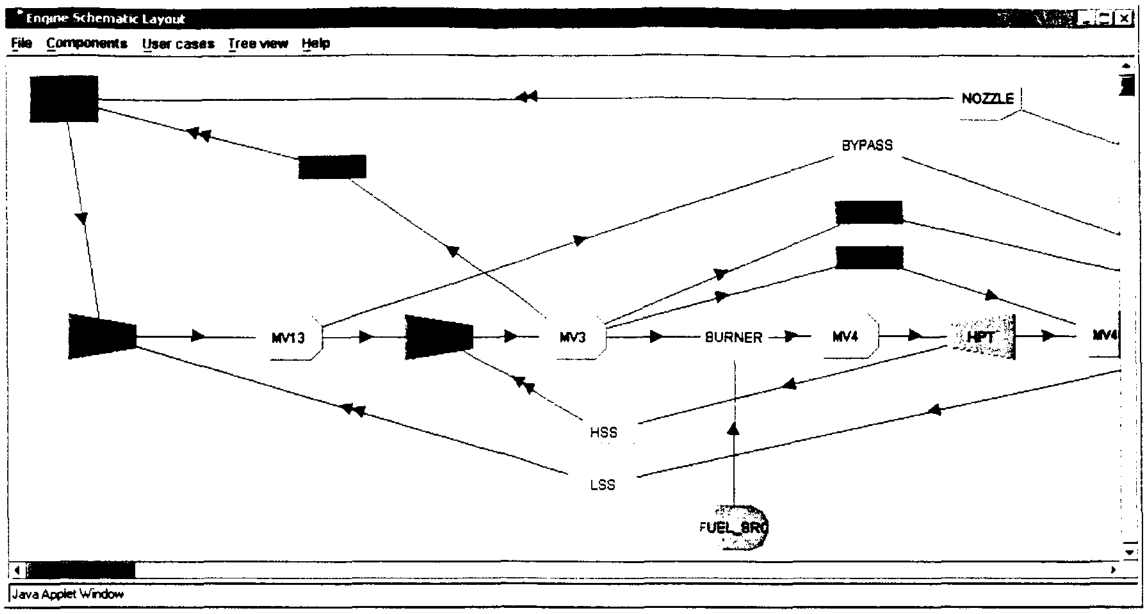
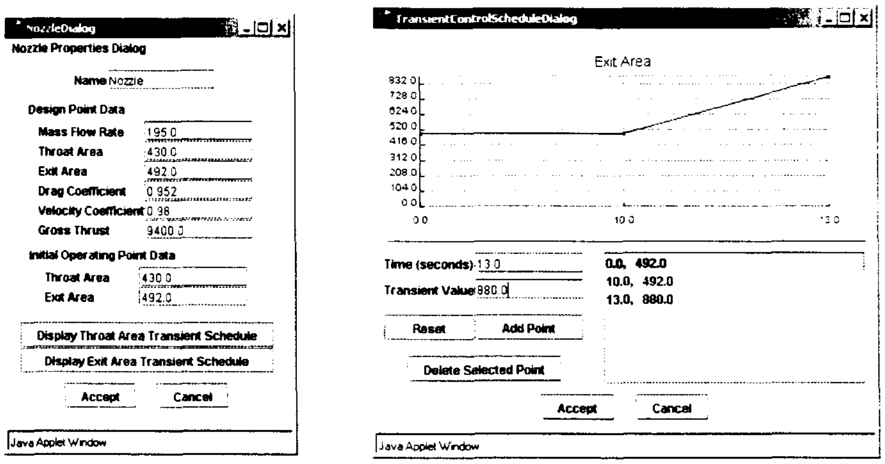


**Figure 7**. Engine schematic layout dialog



Nozzle Properties Dialog

| | |
|---|---|
| Name | Nozzle |

Design Point Data

| | |
|---|---|
| Mass Flow Rate | 195.0 |
| Throat Area | 430.0 |
| Exit Area | 492.0 |
| Drag Coefficient | 0.952 |
| Velocity Coefficient | 0.98 |
| Gross Thrust | 9400.0 |

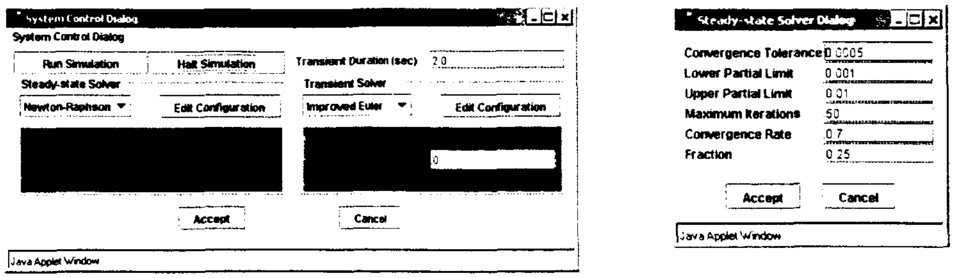
Initial Operating Point Data

| | |
|---|---|
| Throat Area | 430.0 |
| Exit Area | 492.0 |

| | |
|---|---|
| Time (seconds) | 13.0 |
| Transient Value | 880.0 |

0.0, 492.0
10.0, 492.0
13.0, 880.0

**Figure 8**. Dialogs used to set engine component (Nozzle) operational characteristics



System Control Dialog

| | |
|---|---|
| Transient Duration (sec) | 2.0 |
| Steady-state Solver | Newton-Raphson |
| Transient Solver | Improved Euler |

| | |
|---|---|
| Convergence Tolerance | 0.0005 |
| Lower Partial Limit | 0.001 |
| Upper Partial Limit | 0.01 |
| Maximum Iterations | 50 |
| Convergence Rate | 0.7 |
| Fraction | 0.25 |

**Figure 9**. Engine simulation system control dialog

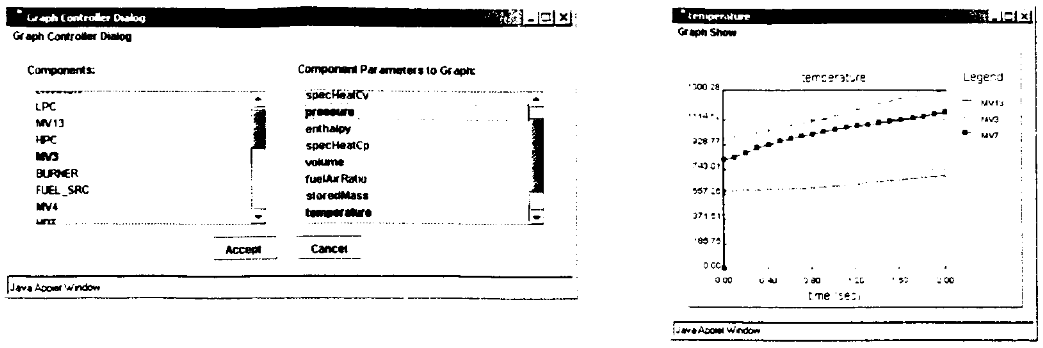


**Figure 10**. Graphically display engine component parameters

Once a simulation begins, the engine configuration data will be encoded in XML format and sent over the Internet to the web simulation server. When the server receives the engine configuration file, it then automatically dispatches the file to the simulation web component, where engine databinding and simulation logic are performed. At the same time, the user can select from *Graph Control Dialog* (Figure 10) to plot a number of specified parameters for any of the components currently displayed in the *Engine Schematic Layout* window. The user may also view simulation status reports, using the *Transcript* button shown in Figure 6, that are sent from simulation web server during the simulation. Once the simulation is completed, the simulation web component will marshal all engine data objects into an engine data file designated by its simulation ID, and store it into the database repository. Finally, the user can use *Save User Case* button to download the complete solution of the simulation case for later use.

Rerun simulation from an existing file

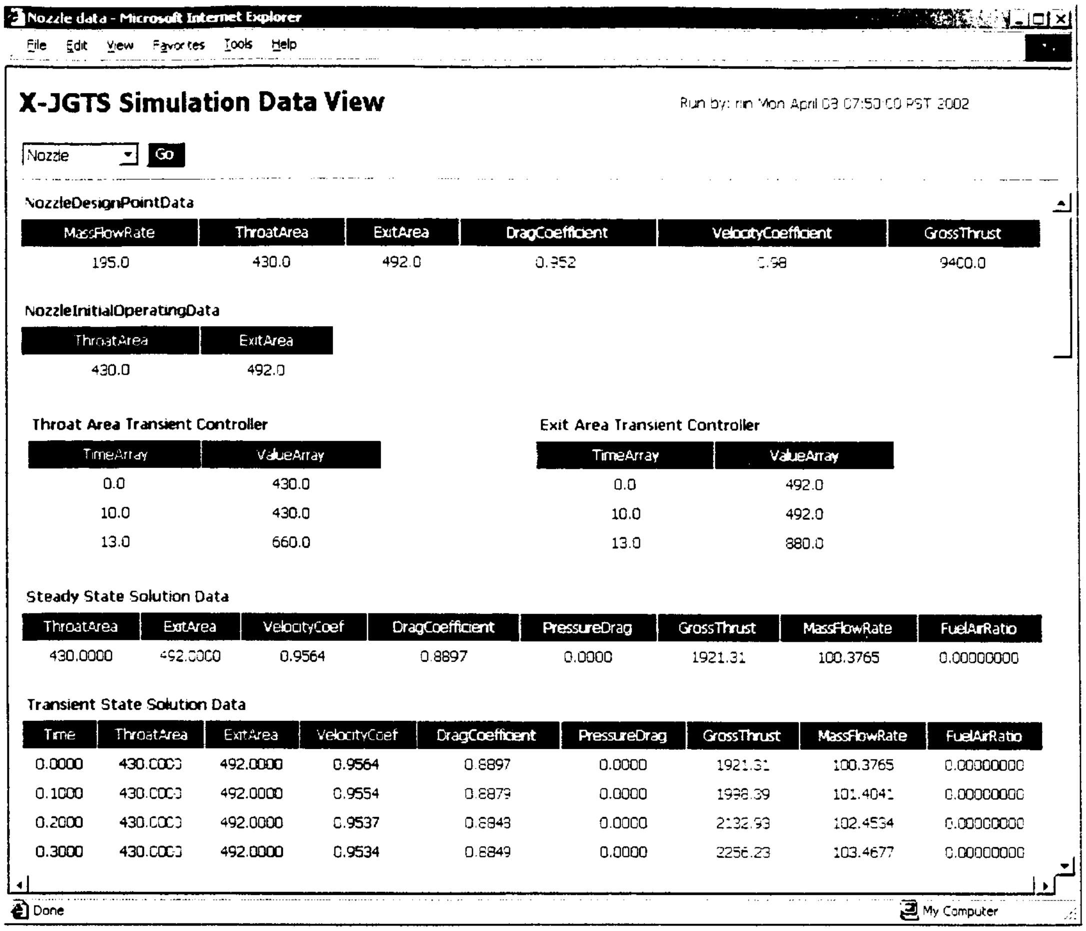
*X-Jgts* also provides a service for users to directly input engine simulation configurations from a file, which allows bypassing the engine construction procedures. Part of a sample configuration file is shown in Figure 11. When a user uploads the configuration file from a web browser (Figure 6), all the defined simulation parameters will be immediately available from *Engine Schematic Layout Dialog* and *System Control Dialog*. Users can then use *User cases* menu in *Engine Schematic Layout* to verify these configurations. Users can also edit these data using the above two dialogs. In this case, the updated configuration file will be sent to the server to run the simulation.

Show existing simulation data results

If a user has finished an engine simulation case and saved the simulation data using *X-Jgts,* he/she can later redisplay the simulation results in a web browser with a more meaningful data presentation scheme using this service. In this case, when the web simulation server receives an engine simulation case file uploaded from the user's web browser (Figure 6), it will internally use *Display Web Component* (combined with sets of pre-designed XSLT style sheets) to dynamically generate HTML code for display within the user's browser. Figure 12 shows the nozzle data file from an example simulation case. The user can choose different engine components to display from the drop-down list at the top of the web page.

```
<?xml version="1.0"?>
<EngineRoot xmlns="http://mems1.ni.utoledo.edu/engine">
      <EngineLibVersion>1</EngineLibVersion>
      <EngineBase BaseName="XJGTS">
         <Configuration>
            <SteadyStateSolver SloverName="NewtonRaphsonSolver"
                     ErrorTolerance="5.0E-4" ConvergenceRate="0.7"
                     InterationToFailure="50" PerturbationSize="0.25"
                     LowerPartialLimit="0.0010" UpperPartialLimit="0.01"/>
            <TransientSolver SloverName="ImprovedEuler"
                     ErrorTolerance="5.0E-4" ConvergenceRate="0.7"
                     InterationToFailure="50" DeltaTime="0.2"
                     FinalTime="2.0" PerturbationSize="0.25"
                     LowerPartialLimit="0.0010" UpperPartialLimit="0.01" />
            <Connectors>
               <Connector from="Environment" to="LPC" isFeedback="false"/>
               <Connector from="LPC" to="MV13" isFeedback="false"/>
               <Connector from="MV13" to="HPC" isFeedback="false"/>
               <!-- other connectors are defined in a similar mannar -->
             </Connectors>
        </Configuration>
        <EngineModel>
            <Components>
              <!—only Nozzle is illustrated here, the same with all other components -->
              <NonSource>
                <NonRatator>
                   <Nozzle Name="Nozzle">
                      <NozzleDesignPointData MassFlowRate="195.0"
                         ThroatArea="430.0" ExitArea="492.0" DragCoefficient="0.952"
                         VelocityCoefficient="0.98" GrossThrust="9400.0"/>
                      <!—the same with NozzleInitPointData-->
                      <ThroatAreaTransientControllers name="Throat Area Transient Controller">
                         <TimeArray> 0.0 10.0 13.0 </TimeArray>
                         <ValueArray> 430.0 430.0 660.0 </ValueArray>
                      </ThroatAreaTransientControllers>
                      <!— the  same with ExitAreaTransientControllers -->
                   </Nozzle>
                </NonRatator>
              </NonSource>
           </Components>
        </EngineModel>
      </EngineBase>
</EngineRoot>
```

**Figure 11**. Engine simulation configuration file specified in XML file format


Nozzle data - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Nozzle
Go
Done
My Computer

X-JGTS Simulation Data View

Run by: rin Mon April 03 07:50:00 PST 2002

NozzleDesignPointData

| MassFlowRate | ThroatArea | ExitArea | DragCoefficient | VelocityCoefficient | GrossThrust |
|---|---|---|---|---|---|
| 195.0 | 430.0 | 492.0 | 0.952 | 0.98 | 9400.0 |

NozzleInitialOperatingData

| ThroatArea | ExitArea |
|---|---|
| 430.0 | 492.0 |

Throat Area Transient Controller

| TimeArray | ValueArray |
|---|---|
| 0.0 | 430.0 |
| 10.0 | 430.0 |
| 13.0 | 660.0 |

Exit Area Transient Controller

| TimeArray | ValueArray |
|---|---|
| 0.0 | 492.0 |
| 10.0 | 492.0 |
| 13.0 | 880.0 |

Steady State Solution Data

| ThroatArea | ExitArea | VelocityCoef | DragCoefficient | PressureDrag | GrossThrust | MassFlowRate | FuelAirRatio |
|---|---|---|---|---|---|---|---|
| 430.0000 | 492.0000 | 0.9564 | 0.8897 | 0.0000 | 1921.31 | 100.3765 | 0.00000000 |

Transient State Solution Data

| Time | ThroatArea | ExitArea | VelocityCoef | DragCoefficient | PressureDrag | GrossThrust | MassFlowRate | FuelAirRatio |
|---|---|---|---|---|---|---|---|---|
| 0.0000 | 430.0000 | 492.0000 | 0.9564 | 0.8897 | 0.0000 | 1921.31 | 100.3765 | 0.00000000 |
| 0.1000 | 430.0000 | 492.0000 | 0.9554 | 0.8879 | 0.0000 | 1998.39 | 101.4041 | 0.00000000 |
| 0.2000 | 430.0000 | 492.0000 | 0.9537 | 0.8848 | 0.0000 | 2132.93 | 102.4534 | 0.00000000 |
| 0.3000 | 430.0000 | 492.0000 | 0.9534 | 0.8849 | 0.0000 | 2256.23 | 103.4677 | 0.00000000 |

**Figure 12**. Nozzle simulation data displayed within a user's web browser

## CONCLUSION

In this work, an XML-based dynamic databinding framework for use in engine simulation has been discussed. By dynamic data binding, the framework provides an object interface to access and use engine data, transparently mapping simulation data in engine components as engine data objects. The framework also enables the separation of engine simulation logic from its persistence logic, such that the engine simulation codes and the underlying data persistence codes can be developed independently.

Since engine component data in the binding process are stored in an XML document, they not only bypass the requirement to have a standard binary encoding or storage format, but also provide the meaning of the data through its tag representation. Furthermore, it is completely natural to move around XML engine data using HTTP through disparate networks.

This paper also describes a Web-based engine simulation system, *X-Jgts*, which internally uses engine databinding framework. The simulation system couples a front-end graphical user interface, developed using the Java Swing API, and various Java Servlet-based web components from engine simulation server to service user's requests. The designed web components include remote simulation service, dynamic data display service in HTML format, and file download and upload services which allow a user to save data for later use in a more secure way. All these services are readily available via the built-in databinding framework support and the use of XML to describe engine data. The combined package provides analytical, graphical and data management tools which allow users to construct and control dynamic gas turbine simulations by manipulating graphical objects from a variety of heterogeneous computer platforms through the use of Java-enabled world-wide web browsers.

The method developed in this paper is generic and may readily be used for other simulation applications requiring intensive data exchange. Using this approach, developers are enabled to design better aircraft engine simulation codes via a systematic and more meaningful data representation scheme and a built-in data validation method.

## ACKNOWLEDGMENTS

The authors gratefully acknowledge partial financial support from NASA Grant NAG-1-2244 under the direction of Mr. Wayne K. Gerdes, NASA Langley Research Center. They also would like to express their appreciation to Dr. John A. Reed, the University of Toledo, for providing parts of *Onyx* [16] simulation code to test this work.

## REFERENCE

[1] Reed, J. A., Follen, G. J. and Afjeh, A. A., *Improving the Aircraft Design Process using Web-based Modeling and Simulation*, ACM Transactions on Modeling and Computer Simulation, Vol. 10, No. 1, 2000, pp. 58-83

[2] Fishwick, P. A., Hill, D. R. C. and Smith, R., Eds., *Proceedings of the 1998 International Conference on Web-Based Modeling and Simulation*, SCS Simulation Series, Vol. 30, (1998).

[3] Reed, J. A. and Afjeh, A. A., *A Java-based Interactive Graphical Gas Turbine Propulsion System Simulator*, AIAA paper 97-0233, 35th Aerospace Sciences Meeting and Exhibit, Reno NV

[4] EngineSim Beta Version 1.5b, NASA Glenn Learning Technologies Project. http://www.grc.nasa.gov/WWW/K-12/airplane/ngnsim.html

[5] Extensible Markup Language (XML) 1.0, W3C Recommendation, Feb. 1998

[6] Daniele, C. J., Krosel, S. M., Szuch, J. R., and Westerkamp, E. J., "Digital Computer Program for Generating Dynamic Engine Models (DIGTEM)," NASA TM-83446, 1983.

[7] Reed, J. A., "Development of an interactive graphical propulsion system simulator," Master of Science Thesis, The University of Toledo, Toledo, Ohio, August 1993.

[8] Lin, R. and Afjeh, A. A., *A Dynamic Data Binding Framework for High Performance Object-Oriented Propulsion System Simulation*, 2002 Advanced Simulation Technologies Conference, High Performance Computing Symposium, April 2002

[9] "XML Schema Part 0: Primer, W3C Recommendation," 2 May 2001, http://www.w3.org/TR/xmlschema-0

[10] Watson, M., "Creating Java Beans: Components for Distributed Applications," Morgon Kaufmann Publishers, Sept. 1997

[11] Sun Microsoft System, Java Servlet Specification at: http://java.sun.com/products/servlet/index.html

[12] Sun Microsoft System, Java Server Page Specification at: http://java.sun.com/products/jsp/index.html

[13] The Jakarta Project at: http://jakarta.apache.org

[14] File Upload Specification RFC1867 http://www.ietf.org/rfc/rfc1867.txt

[15] XSL Transformation W3C Recommendation version 1.0, November, 1999. http://www.w3.org/TR/xslt,

[16] Reed, J.A., 1998. "Onyx: An Object-Oriented Framework for Computational Simulation of Gas Turbine Systems," Ph.D. Dissertation, The University of Toledo