AN EXERCISE IN ONTOLOGY DRIVEN TRAJECTORY SIMULATION WITH MATLAB SIMULINK®

Umut DURAK
TUBITAK SAGE
PK 16 Mamak
06261 Ankara, Turkey
E-mail: udurak@sage.tubitak.gov.tr

Serdar GÜLER
METU Computer Engineering Dept.
06530 Ankara, TURKEY
E-mail: sguler@metu.edu.tr

Halit OĞUZTÜZÜN
METU Computer Engineering Dept.
06530 Ankara, Turkey
E-mail: oguztuzn@ceng.metu.edu.tr

S. Kemal İDER
METU Mechanical Engineering Dept.
06530 Ankara, Turkey
E-mail: kider@metu.edu.tr

KEYWORDS
Trajectory Simulation, Function-Oriented Design, Ontology-Driven Simulation, Domain Engineering, Model Driven Development.

ABSTRACT

We demonstrate an application of the ontology driven methodology to develop trajectory simulations in a function-oriented style. We adopt a model based approach to software development, guided by the domain engineering process, to promote knowledge and software reuse. MATLAB Simulink® block definitions have been generated from the function specifications in the Trajectory Simulation Ontology, called TSONT. MATLAB implementations of the blocks have been generated from the DAVE-ML definitions of the functions, which are incorporated in TSONT. Finally, the simulation has been put together by manually connecting the blocks.

INTRODUCTION

Trajectory simulations involve the mathematical models of the behavior of a munition and its subsystems during their operation. They compute the flight path and flight parameters, such as orientation, and angular rates, of a munition from the start to the end of its motion (US Department of Defense 1995). There is a wide span of trajectory simulations differing widely with respect to their performance and fidelity characteristics, from simple point-mass simulations to six-seven degrees of freedom hardware-in-the-loop missile simulations. The requirements of a given trajectory simulation are derived from the objectives of the intended user, who might be interested in the analysis, development, procurement and operation of some munition.

From our observations, common practice is to develop trajectory simulations for each and every application again and again. When the complexity of the modeled systems and requirements of the simulation application are considered, the risk of failure in such projects is considered to be high. Moreover, the expenditure of intellectual labor to study similar problems of the same domain is a waste. Another concern is assuring the quality of the product of each development effort. Bear in mind that verification in a trajectory simulation project requires a great deal of effort due to the demand for experts' time and flight data, which are both hard to obtain. Implementing a systematic software reuse will help make best use of past successful efforts. We propound an ontology based reuse infrastructure that will enable the trajectory simulation developer to reuse knowledge, design and code throughout the development cycle.

The ontology captures the essential knowledge in trajectory simulation domain and makes it available for reuse. Trajectory Simulation Ontology (TSONT) has been developed as the domain model of the reuse infrastructure (Durak et al. 2005, Durak et al. 2006a). Our ontology based reuse infrastructure uses domain engineering methodology, enhancing it with Model Driven Engineering (MDE) concepts. An earlier practice of this methodology was a six DOF object oriented trajectory simulation framework in MATLAB (Durak et al. 2006b). This model driven methodology helps us develop a trajectory simulation reuse infrastructure that aims to make domain knowledge and software reusable across a wide variety of trajectory simulation projects.

In this study, we utilize model transformation techniques to develop reusable code starting from the domain model, which is TSONT. A TSONT to Simulink® Blockset Definition conversion tool has been built. This tool was used to generate function block definitions. Then a Simulink® Blockset for Unguided Point Mass Trajectory Simulation, dubbed PANTHERA, was developed connecting these function block definitions.

Trajectory simulation domain involves mathematical models that account for some kind of behavior or some law. Capturing these models in a systematic way and representing them as an integrated part of the ontology

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

is an important concern. At this juncture, the DAVE-ML effort of NASA for the benefit of flight modeling and simulation community has been leveraged (Jackson et al. 2004).

DAVE-ML (Dynamic Aerospace Vehicle Exchange Markup Language) is a proposed standard for the interchange of aerospace dynamic models. It is aimed to provide a programming language independent representation of aerodynamics, mass/inertia, propulsion and guidance, navigation and control laws of a vehicle. DAVE-ML, which is XML-based, relies on MathML as a means to describe mathematical relations. MathML is an XML-based language for describing mathematics for machine to machine communication. We take advantage of DAVE-ML to incorporate mathematical models into our ontology TSONT.

DAVETools is a DAVE-ML to Simulink® translator developed at NASA. It constructs Simulink® block diagrams from a DAVE-ML file. To exercise the ontology to executable simulation blocks concept, while developing PANTHERA, some of the block codes was automatically generated by using the DAVETools.

ONTOLOGY BASED FUNCTION ORIENTED REUSE SCENERIO

Ontologies and Domain Engineering

Experience in software reuse suggests that the success of reuse is related to the use of artifacts in the context of a domain, where a domain is defined as the area in which an organization does business (Favaro, 1995). The term reuse infrastructure refers to the collection of artifacts that is made available to the software developer. Developing a reuse infrastructure for a problem domain is the essence of domain engineering. Domain engineering comprises three fundamental processes: domain analysis, infrastructure specification, and infrastructure implementation (Falbo et al. 2002).

Domain analysis is the identification, acquisition and evolution of relevant information on a problem domain to be reused in software specification and construction (Arango 1989). The outcome of domain analysis is the domain model, which, in our case, is the trajectory simulation ontology TSONT. Infrastructure specification is the selection and organization of reusable information in the model to fit the patterns of reuse in the environment of the developer. The infrastructure specification, together with the knowledge captured by the domain model, is input to the infrastructure implementation step, which produces and tests the specified components.

Trajectory Simulation Ontology

TSONT (Trajectory Simulation ONTology) has been constructed in OWL (Antoniou and van Harmelen 2004) to serve as the domain model of trajectory simulation reuse infrastructure. The structure of TSONT is devised to render concept to implementation mapping amenable for reuse by trajectory simulation developers. It captures the concepts of trajectory simulation domain in terms of classes, the taxonomy of these classes, and the composition relations of these classes, where a class is defined as an implicit collection of individuals that belong together because they share some properties (W3C 2006). The functions that are used to compute a trajectory are also specified and related to classes in TSONT. Furthermore, dependencies among the functions are stipulated.

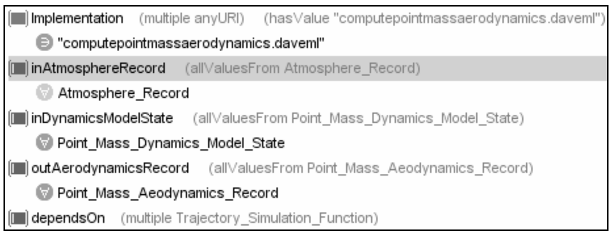
Implementation (multiple anyURI) (hasValue "computepointmassaerodynamics.daveml")
"computepointmassaerodynamics.daveml"
inAtmosphereRecord (allValuesFrom Atmosphere_Record)
Atmosphere_Record
inDynamicsModelState (allValuesFrom Point_Mass_Dynamics_Model_State)
Point_Mass_Dynamics_Model_State
outAerodynamicsRecord (allValuesFrom Point_Mass_Aeodynamics_Record)
Point_Mass_Aeodynamics_Record
dependsOn (multiple Trajectory_Simulation_Function)

Figure 1: Compute Aerodynamics Point Mass Function Definition from TSONT (screenshot from Protégé)

Figure 1 depicts a function definition from TSONT. Compute Aerodynamics Point Mass function is defined by its input and output variables in the ontology. Its implementation is given by referring to a DAVE-ML file. The parameter that starts with "in" associates the function with the input parameter which is also defined in TSONT with allValuesFrom restriction. The allValuesFrom restriction requires that for every instance of the class that has the specified property, the values of the property will be all members of the class indicated by the owl:allValuesFrom clause. This restriction entails that any implementation of Compute Aerodynamics functions for point mass trajectory simulations will require an atmosphere record. Same also applies to the properties that start with "out", which associate the function with its output parameters. The implementation datatype property refers to the DAVE-ML file that contains the algorithm for that function. Datatype properties define the relations between instances of classes and RDF literals and XML Schema datatypes.

Incorporating DAVE-ML in TSONT

Any function defined in TSONT has an implementation property that refers to a DAVE-ML file that documents the mathematical mapping of inputs to outputs of the function. Below is a part of the Update Dynamics Model State and Derivatives 3DOF DAVE-ML.

$$\ddot{y} = {(F_{Ay} + F_{Gy})} \Big/ {mass} \qquad (1)$$

```
<variableDef     name="yddot"     varID="yddot"
units="m/s2">
    <description>Acceleration in Y
    </description>
    <calculation>
        <math>
      <apply>
      <divide/>
         <apply>
            <plus/>
            <ci>FAY</ci>
            <ci>FGY</ci>
         </apply>
         <ci>mass</ci>
      </apply>
    </math>
    </calculation>
    <isOutput/>
  </variableDef>
```

DAVE-ML enables its user to express mathematical relations between input and output parameters using MathML (AIAA Simulation Standards Working Group, 2004). Above is the definition of an output parameter in DAVE-ML. The *yddot*, the second derivative of displacement along y-axis, is dependent on aerodynamic and gravitational forces in y-axis and mass, as described by the mathematical relation given in Equation (1).

Function Oriented Reuse Scenario

Function oriented programming relies on decomposing a system into a set of interacting functions with a centralized system state shared by these functions (Sommerville 1995). Although MATLAB Simulink® supports other paradigms as well (Lee 2003), in this study, we used it in a function oriented fashion and developed a function oriented blockset.
Being the domain model of the infrastructure, TSONT was our starting point. By using the tool (TSONT2SIM), the functions in the trajectory simulation domain are converted to Simulink® block definitions.

As shown in Figure 2, after the ontology to Simulink® transformation, which will be presented in the succeeding section, a set of blocks are selected from the collection of all function blocks. Design process focuses on connecting the selected function blocks to accomplish a task or build a new function in the functional hierarchy.

A block can be implemented either by automatically generating block code from DAVE-ML files or by coding it manually. In this study, we did both as we developed PANTHERA - Point Mass Unguided Trajectory Simulation MATLAB Simulink® Blockset. The PANTHERA reuse infrastructure was then used to develop simulations, such as TIGER and JAGUAR.

AUTOMATED TRANSFORMATION FROM TSONT TO SIMULINK®

MATLAB built in commands are used to generate Simulink® blocks from the ontology, which is in OWL. First, TSONT file is read into an OWLModel object of Protégé. Then, TSONT is parsed to extract the OWL classes that define the functions of the trajectory simulation domain. Protégé OWL API is used to parse the OWL file. As we parsed the ontology, we generated a MATLAB script that would construct the Simulink® blocks when interpreted (MathWorks 2007).

The Protégé-OWL API, offers many facilities to analyze the structure of the ontology (Knublauch 2006). It is being developed as an open-source Java library for OWL and RDF(S). It provides classes and methods to load and save OWL ontologies. It enables the programmer to query and manipulate OWL data models, and to perform reasoning based on Description Logic engines.

Every OWL class in the ontology has allValuesFrom restrictions on some of their properties. To specify the input and output ports of Embedded MATLAB functions these allValuesFrom restrictions are used. The properties that have allValuesFrom restrictions are

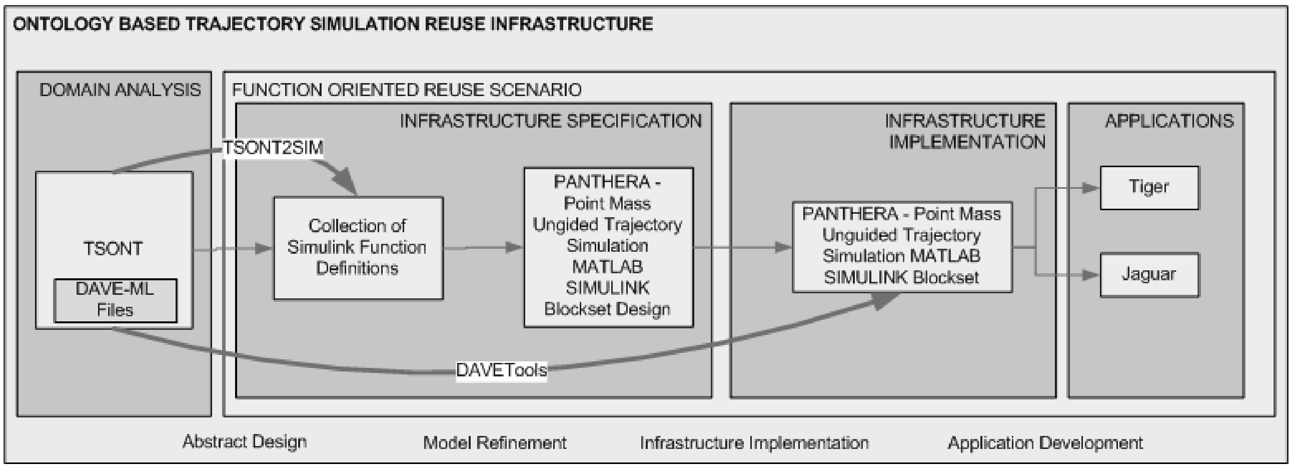
ONTOLOGY BASED TRAJECTORY SIMULATION REUSE INFRASTRUCTURE
DOMAIN ANALYSIS
FUNCTION ORIENTED REUSE SCENARIO
INFRASTRUCTURE SPECIFICATION
INFRASTRUCTURE IMPLEMENTATION
APPLICATIONS
TSONT2SIM
TSONT
DAVE-ML Files
Collection of Simulink Function Definitions
PANTHERA - Point Mass Ungided Trajectory Simulation MATLAB SIMULINK Blockset Design
PANTHERA - Point Mass Unguided Trajectory Simulation MATLAB SIMULINK Blockset
Tiger
Jaguar
DAVETools
Abstract Design
Model Refinement
Infrastructure Implementation
Application Development

Figure 2: Ontology Based Function Oriented Reuse Scenario

separated into two. The properties whose name starts with an "in" prefix become an input port and the ones whose name starts with an "out" become an output port.

Trajectory Simulation functions are organized in TSONT as an inheritance tree. The classes that correspond to Embedded MATLAB Functions are the leaf classes in the class tree of the TSONT. The root classes, from which the leaf classes are inherited from, are used to structure the subsystems in the blockset generated.

To generate MATLAB Simulink® blocks, a MATLAB script is formed. Basically, the *add_block* built-in function is used to add blocks, ports and subsystems into the systems. The *delete_block* and *delete_line* are used to delete the unnecessary components that are automatically generated by MATLAB for housekeeping purposes when a Subsystem or User Defined MATLAB Function is added to the system. The generation of the MDL file (MATLAB file format for Simulink/Stateflow diagrams), which is actually a collection of Simulink® blocks, is done by running the MATLAB script. A sample set of generated MATLAB Simulink® blocks and subsystems is shown in Figure 3.

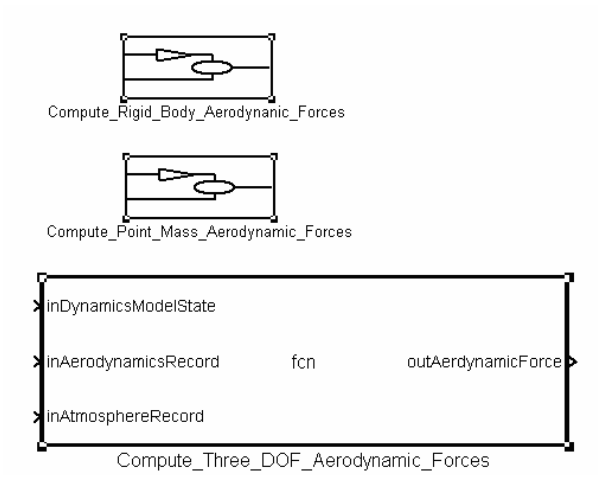
Compute_Rigid_Body_Aerodynamic_Forces
Compute_Point_Mass_Aerodynamic_Forces
inDynamicsModelState
inAerodynamicsRecord
fcn
outAerdynamicForce
inAtmosphereRecord
Compute_Three_DOF_Aerodynamic_Forces

Figure 3 Examples of Automatically Generated MATLAB Blocks and Subsystems

PANTHERA

Simulink® Blockset for Unguided Point Mass Trajectory Simulation, dubbed PANTHERA, has been developed using Simulink® function block definitions automatically generated by our TSONT-to-Simulink® Transformation Tool. PANTHERA aims to be a reusable collection of Simulink® blocks for unguided point mass simulations. So a subset of Simulink® function block definitions is automatically generated. The implementation of blocks are either coded manually, or generated by DAVETools automatically provided that a DAVE-ML reference is available in TSONT. Since the constructs that are either manually coded or automatically generated are Simulink® blocks, we experienced no difficulty in integrating them. One of the generated blocks and its DAVETools generated implementation are presented below in Figure 4.

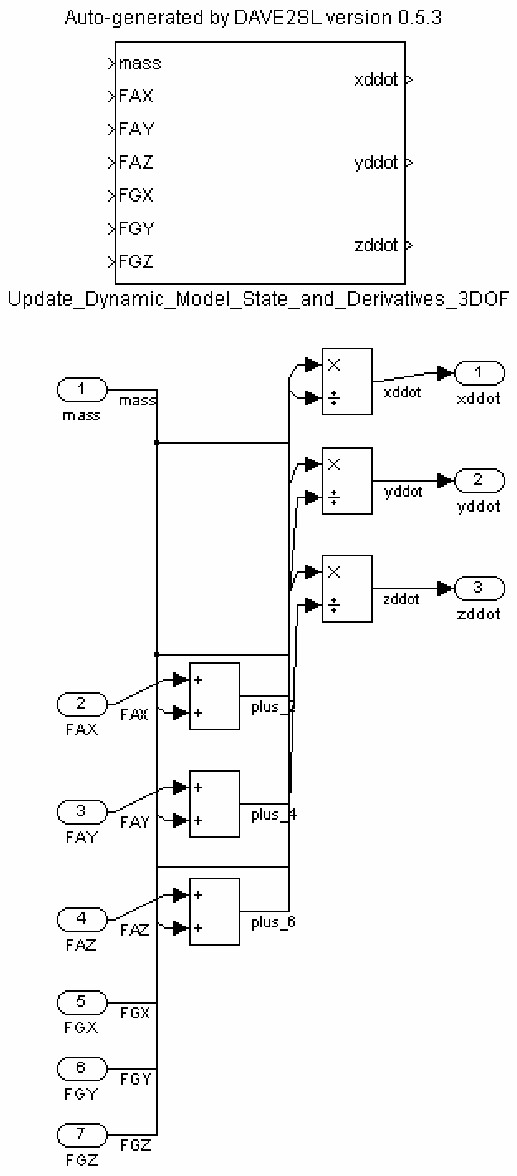

Figure 4 A Block Generated Using DAVETools

Three groups of subsystems have been developed for PANTHERA. The first group consists of subsystems that contain blocks to read simulation parameters. For example, Get Physicals is one of them. The second group contains blocks that carry out basic trajectory simulation computations, such as computing gravitational force. The last group contains subsystems for some high level functions that use the other two groups to accomplish their tasks. For example, Compute Phase Trajectory is one of them. Figure 5 depicts below PATHERA with the mentioned groups.

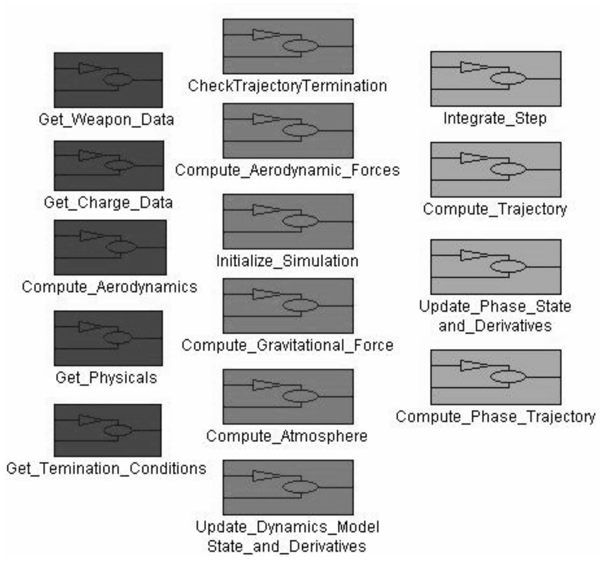
Get_Weapon_Data
Get_Charge_Data
Compute_Aerodynamics
Get_Physicals
Get_Temination_Conditions
CheckTrajectoryTermination
Compute_Aerodynamic_Forces
Initialize_Simulation
Compute_Gravitational_Force
Compute_Atmosphere
Update_Dynamics_Model State_and_Derivatives
Integrate_Step
Compute_Trajectory
Update_Phase_State and_Derivatives
Compute_Phase_Trajectory

Figure 5 PANTHERA Subsystems

Two different mortar simulations, dubbed TIGER and JAGUAR, have been developed using PANTHERA. Both are point mass trajectory simulations based on 81mm mortar data. They differ by means of the blocks used to construct them. While TIGER was developed using the upper most block Compute Trajectory, sub functional blocks were used to develop JAGUAR. Figure 6 below is the result of a sample run of TIGER. With the development of TIGER and JAGUAR this exercise leads us to a process starting from ontology to executable simulation employing model transformation tools and technologies.

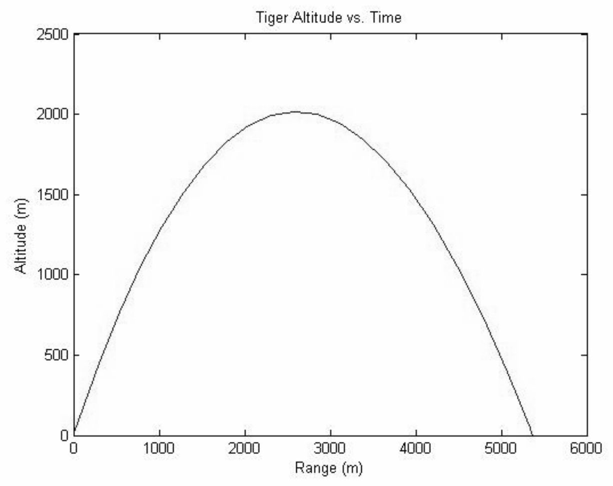

Figure 6 Altitude vs. Range from a Sample Tiger Run

CONCLUSION

This exercise aimed to demonstrate the applicability of the ontology driven, model-based methodology to develop trajectory simulations in a function-oriented style. To generate MATLAB Simulink® block definitions from the function specifications in TSONT, a transformation tool has been built. Outputs of this tool are used to develop a Simulink® blockset for point mass trajectory simulations. On the application engineering front, two trajectory simulations have been built using this blockset. Thus, domain knowledge captured in an ontology can be transformed to an executable simulation by making use of some automated means.

The challenge of this exercise was to make the concepts of model driven engineering work for a simulation domain in the context of a reuse infrastructure. We make use of Protégé OWL API to render the OWL constructs and create Simulink® constructs. While developing this tool, meta-level matching of OWL and Simulink® constructs are hard coded. So the tool that was built for this exercise was specific to TSONT. Building a tool that will enable to define transformations from any ontology to Simulink® blocks can be regarded as the next challenge, which should be doable using state-of-the-art metaprogramming tools. These efforts are believed to build up mature methodologies to produce simulations from domain models.

REFERENCES

AIAA Simulation Standards Working Group, 2004. "Dynamic Aerospace Vehicle Exchange Markup Language (DAVE-ML) Reference, Version 1.7b1". http://daveml.nasa.gov/DTDs/1p7b1/DAVEML_ref.pdf

Antoniou, G. and van Harmelen, F., 2004. *Web Ontology Language: OWL. Handbook on Ontologies*. International Handbooks on Information Systems, Springer.

Arango, G., 1989. “Domain Analysis: From Art to Engineering Discipline”. *In Proceedings of 5th International Workshop on Software Specification and Design*, Pittsburgh, PA.

Durak, U., Mahmutyazicioglu, G. and Oguztuzun, H., 2005. “Domain Analysis for Reusable Trajectory Simulation”. *Euro SIW’05*, Toulouse, France, 303-312.

Durak, U., Oguztuzun, H. and İder, K. 2006, “An Ontology for Trajectory Simulation”. *Proceedings of the 2006 Winter Simulation Conference*, Monterey, CA, USA.

Durak, U., Oguztuzun, H. and İder, K. 2006. “An Ontology Based Trajectory Simulation Framework”. Submitted for publication.

Falbo, R.A., Guizzardi,G. and Duarte, K.C., 2002. “An Ontological Approach to Domain Engineering”. *International Conference on Software Engineering and Knowledge Engineering*, Ischia, Italy.

Favaro.J. 1995. “Technical Report on Reuse”. *European Software Institute*.

Jackson, E., Hildreth, B., York, B. and Cleveland, W. 2004. “Evaluation of a Candidate Flight Dynamics Model Simulation Standard Exchange Format”. *AIAA Modeling and Simulation Technologies Conference and Exhibit, Providence*, Rhode Island.

Knublauch, H., 2006. *Protégé-OWL API Programmer’s Guide*,http://protege.stanford.edu/plugins/owl/api/guide.html

Lee, E.A.. 2003. “Model-Driven Development - From Object-Oriented Design to Actor-Oriented Design”. *Workshop on Software Engineering for Embedded*

Systems: From Requirements to Implementation, Chicago.

MathWorks Inc., 2007. *Simulink® User's Guide*, http://www.mathworks.com/access/helpdesk/help/toolbox/simulink/ug/?bqchgnk.html

Sommerville, I., 1995. *Software Engineering*. Addison Wesley Longman Publishing, Redwood City, CA, USA.

U.S. Department of Defense 1976. *Missile Flight Simulation, Part One Surface-to-Air Missiles*. MIL-HDBK 1211.

W3C, 2004. *OWL Web Ontology Language Overview*. http://www.w3.org/TR/2004/REC-owl-features-20040210/.

AUTHOR BIOGRAPHIES

UMUT DURAK is a Head Researcher in Defense Industries Research and Development Institute of Scientific and Technological Research Council of Turkey (TUBITAK-SAGE). He has been involved in modeling and simulation of dynamic systems. He obtained his BS and MS from Mechanical Engineering Department of Middle East Technical University (METU), Ankara, Turkey. He is currently a Ph.D. student in the same department. His e-mail address is umut.durak@sage.tubitak.gov.tr.

SERDAR GÜLER is a Research Assistant in Biology Department at the Middle East Technical University (METU), Ankara, Turkey. He obtained his BS from Food Engineering Department of Middle East Technical University. He is currently MS student in Software Engineering and Biotechnology departments in the same university. His e-mail address is sguler@metu.edu.tr

HALIT OGUZTUZUN is an associate professor in the Department of Computer Engineering at the Middle East Technical University (METU), Ankara, Turkey. He obtained his BS and MS degrees from METU in 1982 and 1984, and PhD from University of Iowa, Iowa City, IA, USA in 1991. His current research interests include distributed simulation and model-driven engineering. His e-mail address is oguztuzn@ceng.metu.edu.tr.

S. KEMAL İDER is a professor in the Mechanical Engineering Department at the Middle East Technical University (METU), Ankara, Turkey. He obtained his BS and MS degrees in Mechanical Engineering from METU, both in 1976. He received MS degree in Economics in 1979 and PhD degree in Mechanical Engineering in 1988 from the University of Illinois at Chicago. His research interests include multibody dynamics and control of flexible systems. His e-mail address is kider@metu.edu.tr.