Integrated Design and Process Technology, **IDPT-2006**
Printed in the United States of America, June, 2006

COMPONENT ORIENTED SIMULATION DEVELOPMENT WITH AXIOMATIC DESIGN

Cengiz Togay, Ali H. Dogru

Computer Engineering Department, Middle East Technical University, Turkey.

Urcun J. Tanik, Gary J. Grimes,

Electrical and Computer Engineering Department, University of Alabama.

ABSTRACT

Component Oriented Software Engineering (COSE) tools generally deal with the composition of components using their interfaces. They operate at the level of component's interface and connect components by limited semantic guidance. These COSE approaches suffer from lack of standards and systematic documentation of component properties. A component interface is not detailed enough to define all interface items and relationships among them. However, Axiomatic Design matrix (AD) includes interface items and Functional Requirements (FRs). In this study, AD matrix notation is utilized for satisfying FRs defining interface items. Still some components cannot be integrated because of design mismatches in their mutual attributes and interactions. Therefore, a designer is left with two options: a) use of mature domain notion by COSE or, b) attempt to fix mismatches by wrappers or translators. In this study, we investigated the use of mature domains which provide interface standard using Object Model Template (OMT). OMT is a well known simulation standard used to solve interface problems among components (federate and/or federates). In this paper, we propose a method for rapid incremental prototyping of simulations from existing mature domain components by using Object Model Template (OMT), Component Oriented Software Engineering Model (COSEML), and Axiomatic Design Theory (ADT). We support our prototyping method with a representative simulation example.

INTRODUCTION

All industries attempt to reduce cost and time required to develop increasingly sophisticated products without sacrificing reliability. Many theories, algorithms, heuristics, and technologies have been developed to resolve complexity issues, such as Axiomatic Design Theory (ADT), the Theory of Inventive Problem Solving (TRIZ), Knowledge-based Engineering (KBE), and Semantic Web Technology (SWT). After a review of these topics, we will propose a method to create a simulation from existing domain components by using Object Model Template, Component Oriented Software Engineering Model (COSEML), and Axiomatic Design Theory (ADT)

ADT defining System Complexity

Developed at MIT, Axiomatic Design Theory (ADT) is an approach to design at the conceptual phase that proposes to resolve complexity issues early in the design process by applying two fundamental axioms: (1) Independence Axiom and (2) Information Axiom. Axiom 1 serves to produce a design matrix that can be uncoupled, coupled, or decoupled. An uncoupled design matrix can be represented as a matrix mapping each functional requirement (FR) to only one design parameter (DP), showing a one-to-one correspondence – considered an ideal design in a perfect world. A coupled design can be represented as a matrix mapping each FR to every single DP, creating a full design matrix depicting a design where each FR is influenced by every DP – considered the most complex design. A decoupled design is a design that can be represented as a triangular matrix, where every DP is not needed to fulfill each FR – considered more realistic since it is between both complexity extremes. Once Axiom 1 is applied, Axiom 2 is applied to select a design from the candidates produced, as represented with the matrices, according to information content. After determining the design range, defined as the range of system operations needed for a particular FR as fulfilled by a DP, and the system range, defined as the range for which the system can operate successfully, the probability for success for a that particular component can be computed. By taking into consideration all the components, the overall system complexity can be computed by determining the total information content (Suh, 2001).

Synergistic Innovation process with ADT and TRIZ

In order to reduce design complexity, innovative approaches have been developed independently of ADT,

such as the Theory of Inventive Problem Solving (TRIZ). Developed in Russia, TRIZ attempts to resolve this inherent tradeoff at the concept stage using a notion called contradiction matrix (TRIZ Web Site, 2006). This matrix serves to compare any given object's "improving feature" (the feature which the designer wishes to retain) with its "worsening feature" (the feature the designer does not wish to sacrifice). TRIZ has been combined with Axiomatic Design Theory when a trade-off needs to be resolved to decouple a complex matrix by introducing an innovative design parameter (DP) or component to replace the previous one, thereby reducing the overall information content.

KBE as an approach to reducing system entropy and complexity

Knowledge-based engineering has been used to reduce design complexity by developing accessible rule-bases, enabling engineers to leverage best practices by storing heuristics in addition to quantitatively knowledge. Entropy, the state of disorganization, can be reduced and measured using innovative KBE techniques. For instance, the quantity of rules and the number of hierarchies, along with the number of connections between concepts can be considered when calculating total entropy, giving a means for automated calculation of entropy of a knowledge base. Also, ontologies, or specifications of a concept, can be provided by the domain expert and captured by the KBE system in the form of rules to form a knowledge base that can be built using an ontology framework. This type of framework allows the functional specifications of a part and its inter-relationships with other parts to be systematically and thoroughly expanded over time, as domain experts and intelligent agents populate the rule-base. Rule categories can include (1) rules that automatically formulate a part's specifications (2) rules that calculate engineering properties of a structure (3) rules that enable configuration selection according to limiting conditions (4) rules that optimize to improve on parameters such as cost, speed, and quality (5) rules that provide guidance on where and how to procure key information from the Internet and distributed databases (6) rules that provide analysis and second opinions (7) rules that impose the latest formats and standards to a design (8) rules that reveal design intent providing justification for an engineering decision and enabling design flexibility (9) rules that function as heuristics, providing fuzzy guidance on undefined problems (Tanik et al., 2005).

Semantic Web Technology as a driving force for complexity reduction

Semantic Web technology has the potential to revolutionize the design process through automated synthesis by enabling intelligent assimilation of these ontologies developed globally by individuals and/or firms. In other words, components can be "marked-up" in languages such as DAML, JESSKB, or OWL and uploaded to the Internet, fetched by intelligent agents searching for certain components to complete designs according to guiding rules provided by the designer. The process of storing and accessing millions of clusters of highly-structured, machine-processible, re-usable knowledge units, in the form of ontologies marked up in a semantic language, is a potential technology disruptor.

COSEML "Divide and Conquer" Approach

Another more commonly used approach to solving complex problems is the "divide and conquer" approach, which seeks to divide the problem into simpler parts, solve them, and integrate into a viable solution (Simon, 1969; Tanik, et al., 1991). In order to cope with the increasing complexity of a problem, Component Oriented Software Engineering (COSE) such as COSEML (Dogru, et al., March/April 2003) approaches represent a development methodology for assembling systems from reusable components by interface matching.

Using component technologies is a cost-effective way of constructing systems. In traditional development, system integration is often considered as a separate phase. In COSEML component integration is the centerpiece of the approach; thus, implementation has given way to integration as the focus of system construction. Because of this, integration is a key consideration in the decision whether to acquire, reuse, or build the components. If a reliable component is already developed, it can be used where it is needed. Hence, additional time or funds is not required for implementing this part of the solution.. But some challenges still exist that can be generally attributed to two factors (Jololian, et al., 2004):

- Multiple competing standards, as is the case with the multitude of component models in existence today (e.g. COM, CORBA, EJB, etc.).
- Lack of standards, as is often the case with separately designed components.

Although there are various types of component standards (e.g. COM, CORBA, EJB, etc.), these standards just provide the environment for the execution of components. Traditional components define the interface of components independently due to lack of standards. Because of these problems, COSE approaches assume that there are some mature domains that include components which are suitable for integration. Mature domains need to have a standard naming process understood by all users. For instance, though a Graphical User Interface (GUI) designer may appear to have a good understanding of a 'combo box', a semantic gap still exists causing a lack a consensus. Therefore, more semantics included in the component interface specification will aid in locating and integration of components (Beugnard, et al., 1999; Cicalese, et al., 1999; Dogru et al., March/April 2003).

High Level Architecture and Object Model Template

An objective of this work is to investigate the problems associated with COSE approaches. Both simulation and COSE communities are conducting research associated with the technology that will make it possible to easily build complex systems by combining existing components (Bartholet, et al., 2004; Togay, et al., 2005a; Togay, et al., 2005b). Thus, we select High Level Architecture (HLA) based applications (simulations) as a practice area because of well-defined standards. HLA defines the component interface (Object Model Template (IEEE, 2000b) (OMT)) and communication standards (Runtime Infrastructure (RTI)(IEEE, 2000a)). Simulation based components (federates) use Object Model Template (OMT) for offering the interface. This type of definition is more powerful than the standard approach, because every object of the interface must be defined based on OMT standards. Although OMT provides some standards, independently developed components can use different object hierarchies and different names for the same object. So we assume that there are lots of mature domains. Each mature domain includes only one OMT hierarchy (objects, interactions, complex data types, etc.) and one or more components which use this OMT. Compatible components can be found using OMT (Togay et al., 2005b; Togay et al., 2005a). In (Togay et al., 2005b; Togay et al., 2005a), we assumed that some components have been developed before; this is a property of mature domains. Possibly some of the components are not developed; therefore, we need to define (design) the missing part of the application after specifying existing components. This design must be compatible with the components in the solution space. Thus, we need a design tool which will produce designs based on OMT classes. We selected Axiomatic Design Theory (ADT) (Suh, 2001) for this aim to create good design at conceptual level. Axiomatic design introduces an approach to solve the problem in a hierarchical way. Designer starts with the more general functional requirements (FRs) of a problem and systematically converges to more detailed requirement specifications. But at the same time, FRs define the abstract needs of a design, which is specified more precisely by real world implementations (Design Parameters (DPs)) to fulfill the corresponding FR (i.e. DPs in software, such as program code, algorithms satisfying FRs with abstract goals x, y, and z, respectively.). There are some examples of software design methodologies based on axiomatic design that are developed for Structured Analysis and Development Technique (SADT) (Do, et al., 1996), Object Oriented environments (Clapis, et al., 2000; Suh, 2001; Do, et al., June 2000; Do, et al., October 1999) and requirements management. But, so far there is no example or technique for component- based or oriented development with axiomatic design. Axiomatic design provides the documentation which is defined as a significant problem in (Garlan, et al., 1995). For instance, when components are designed using axiomatic design, all relations among attributes and interactions in case of FRs are documented. If we design the simulation using axiomatic design, then artifacts define all the relations among attributes and interactions with FRs.

OMT can be helpful to the developer to define the DPs in axiomatic design. At the end of the design, leaf-level DPs can be used to reach components which use or satisfy the DPs (OMT objects). Also in terms of component development, these DPs form the interface of components. Components can be designed using only ADT, but we will need a component representation. We use the COSEML tool to represent components and relations among them. Leaf level DPs of components are mapped to COSEML as component interface representations, automatically. COSEML includes a markup language. Therefore automatic operations can be done by different tools using markup documentation of component and/or application.

Method and Example

Mature Domain (MD) is an actionable domain knowledge that supports integration of suitable components (i.e SOM files, and OMT classes of domain and a design matrix (DM) of components). A MD expands in time with newly developed components. Components in a MD provide the specification to develop new ones. Therefore, compatible components that can be developed separately from the application depend on mature domain requirements. Also they can be developed while designing the application.

We selected the embedded cruise-control system to demonstrate our method. Cruise Control System controls the speed of a vehicle at a desired value as long as the speed remains uninterrupted by the pressing of the break pedal. If the system is switched on, pressing the “set” switch, it immediately begins maintaining the current speed unless it reacts to the actuating of switches in order to increase or decrease speed, or actuating of the accelerator pedal (Tanik et al., 1991).

Component Development

As mention above, components can be developed using a MD. We assume that there is only one component developed and located in the MD. In Figure 1, all component representations are demonstrated except OMT tables. OMT tables include more detailed information about components such as attribute types, data types etc. Developers are able to implement new component(s) based on these representations. For instance, Figure 1.a shows that Current Speed Calculator component that requires the Clock-Tick from outside. FRs of Clock-Tick in Figure 1.c may include more information about why the component needs this particular object. Actually, we can

see that Clock-Tick is used for calculating current speed in the same figure. Therefore, a developer can implement a Clock component to satisfy the Clock-Tick requirement of Current Speed Calculator component. When Clock component (see Figure 2) is developed, it is also added to the MD.

Application Development

Another process for component development is to create components while application is being developed. We assume that eight components (Gas Tank, Throttle, Engine, Wheel, Current Speed Calculator, Clock, Throttle Setting Calculator, and Desired Speed Calculator) of Cruise Control domain are developed before and are present in the MD. In this paper, we only provide selected components and their respective representations due to page limitations, but COSEML representation of mature domain is given in Figure 3.

Application developer can see all components and their relations in the MD. Therefore, a developer can select components from a MD for simulation. MD may or may not be executable. In this example, MD components are executable because all components are satisfied by the publish-subscribe mechanism. Each object in components have at least one publisher and zero or more subscribers. Objects of Gas Tank are listed in Table 1. It shows that Gas Tank is ready to execute with supporter components. However, semantically some objects must be satisfied with a subscriber component such as Tank.Fill. This object must be called at the beginning of the application otherwise the system will not work because there is no gas in the tank. Also, some component must call (subscribe) the Brake, Accelerator, Resume interactions. Therefore, an application specific component must be developed. This component has all specification, interaction and attribute names, FRs, data types etc. to develop new emergent component(s). Customer may be concerned to see visually current speed and desired speed values. User Interface Emergent component attribute and interactions are depicted in Figure 4 and visual representation is given in Figure 5.

CONCLUSION

In this paper, we proposed a method to help COSE approaches using AD, COSEML, and OMT. AD connects the FRs with DPs (attributes and interactions) and represents a design matrix. COSEML is used to show abstract designs to support human understanding. Interface items of components in COSEML are directly related with design matrix of a component. OMT provides the interface standard for components.

ACKNOWLEDGEMENT

We sincerely appreciate intellectually stimulating discussions with our colleagues in Component Based Systems (CBS) research group, especially Ozgur Aktunc, Gayathri Sundar, and Rajani Sadasivam. We also appreciate useful comments provided by Dr. Murat M. Tanik.

REFERENCES

TRIZ Web Site, 2006, 21 February, from www.triz40.com.

Bartholet, R. G., Brogan, D. C., Reynolds, P. F. and Carnahan, J. C., 2004, "In Search of the Philosopher's Stone: Simulation Composability Versus Component-Based Software Design," *Proceedings of the 2004 Fall Simulation Interoperability Workshop*, Orlando,FL.

Beugnard, A., Jezequel, J.-M., Plouzeau, N. and Watkins, D., 1999, "Making Components Contract Aware," *IEEE Computer*, 32, 38-45.

Cicalese, C. D. T. and Rotenstreich, S., 1999, "Behavioral Specification of Distributed Software Component Interfaces," *IEEE Computer*, 32, 46-53.

Clapis, P. J. and Hintersteiner, J. D., 2000, "Enhancing Object Oriented Software Development Through Axiomatic Design," *First International Conference on Axiomatic Design*, Cambridge, MA.

Do, S. H. and Park, G. J., 1996, "Application of Design Axioms for Glass-Bulb Design and Software Development for Design Automation," *Third CIRP Workshop on Design and Implementation of Intelligent Manufacturing*, 119-126.

Do, S. H. and Suh, N. P., June 2000, "Object Oriented Software Design with Axiomatic Design," *Proceedings of ICAD2000 First International Conference on Axiomatic Design*.

Do, S. H. and Suh, N. P., October 1999, "Systematic OO Programming with Axiomatic Design", IEEE Computer, 32, 121-124.

Dogru, A. H. and Tanik, M. M., March/April 2003, "A Process Model for Component-Oriented Software Engineering", IEEE Software, 20, 34-41.

Garlan, D., Allen, R. and Ockerbloom, J., 1995, "Architectural Mismatch or Why it's hard to build systems out of existing parts," *Proceedings of the 17th International Conference on Software Engineering*, 179-185.

IEEE, 2000a. "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)- Federate Interface Specification," Institute of Electrical and Electronics Engineering, Inc., Standard-1516.1-2000, New York.

IEEE, 2000b. "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Object Model Template (OMT) Specification," Institute of Electrical and Electronics Engineering, Standard-1516.2-2000, New York.

Jololian, L. K., Ngatchou, J. C. and Seker, R., 2004, "A Component Integration Meta-Framework using Smart Adapters," *IEEE Proceedings of the 2004 International Symposium on Information and Communication Technologies ACM*, 90, 128-133.

Simon, H. A., 1969, "The Science of the Artificial," The MIT Press.

Suh, N. P., 2001, "Axiomatic Design: Advantages and Applications," Oxford University Press.

Tanik, M. M. and Chan, E. S., 1991, "Fundamentals of Computing for Software Engineers," Van Nostrand Reinhold.

Tanik, U. J., Grimes, G. J., Varadraj Gurupur and Sherman, C. J., 2005, "An Intelligent Design Framework Proposal Leveraging Axiomatic Design and the Semantic Web," *Journal of Integrated Design and Process Science*, 9, 41-53.

Togay, C. and Dogru, A. H., 2005a. "A Framework for Component Integration Using Axiomatic Design and Object Model Template for Simulation Applications," Department of Electrical and Computer Engineering University of Alabama, Technical Report- 2005-11-ECE-001, Birmingham, Alabama.

Togay, C. and Dogru, A. H., 2005b, "Infrastructure Design for HLA Based Automated Federation Development," *The Eighth World Conference on Integrated Design and Process Technology*, 698-704.

FIGURES AND TABLES

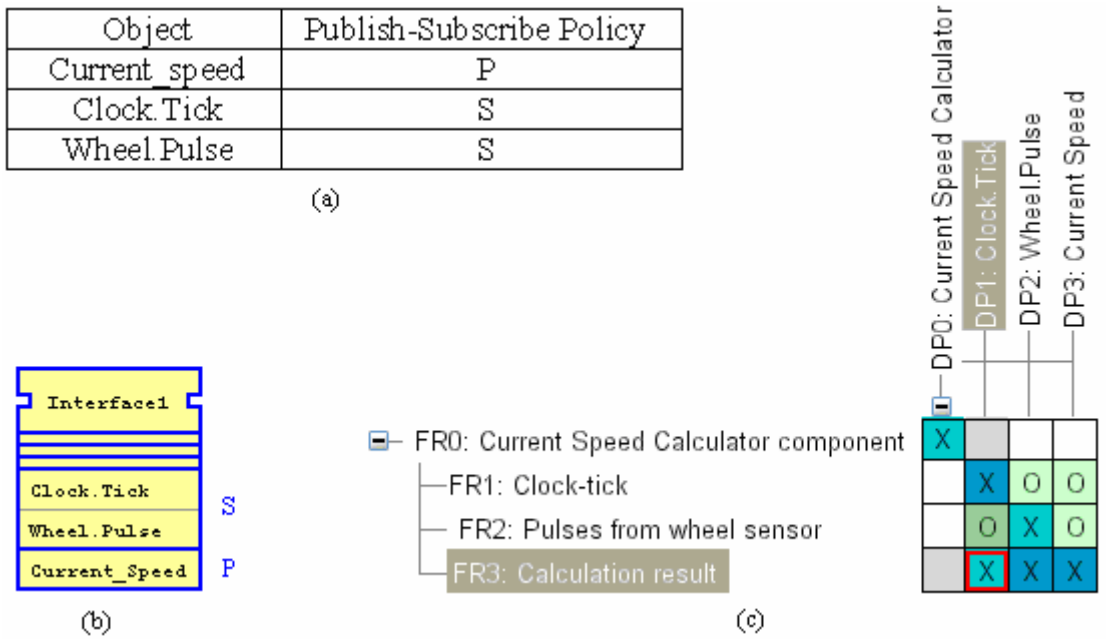

Figure 1: Current Speed Calculator component representations: a) Publish-Subscribe Policy of component objects, b) COSEML Component representation c) Design Matrix of component

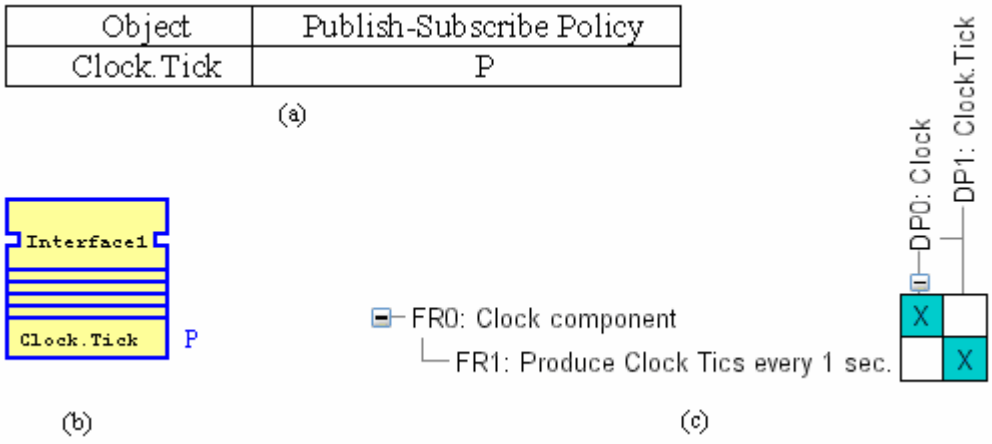

Figure 2: Clock component representations: a) Publish-Subscribe Policy of component object, b) COSEML Component representation c) Design Matrix of component

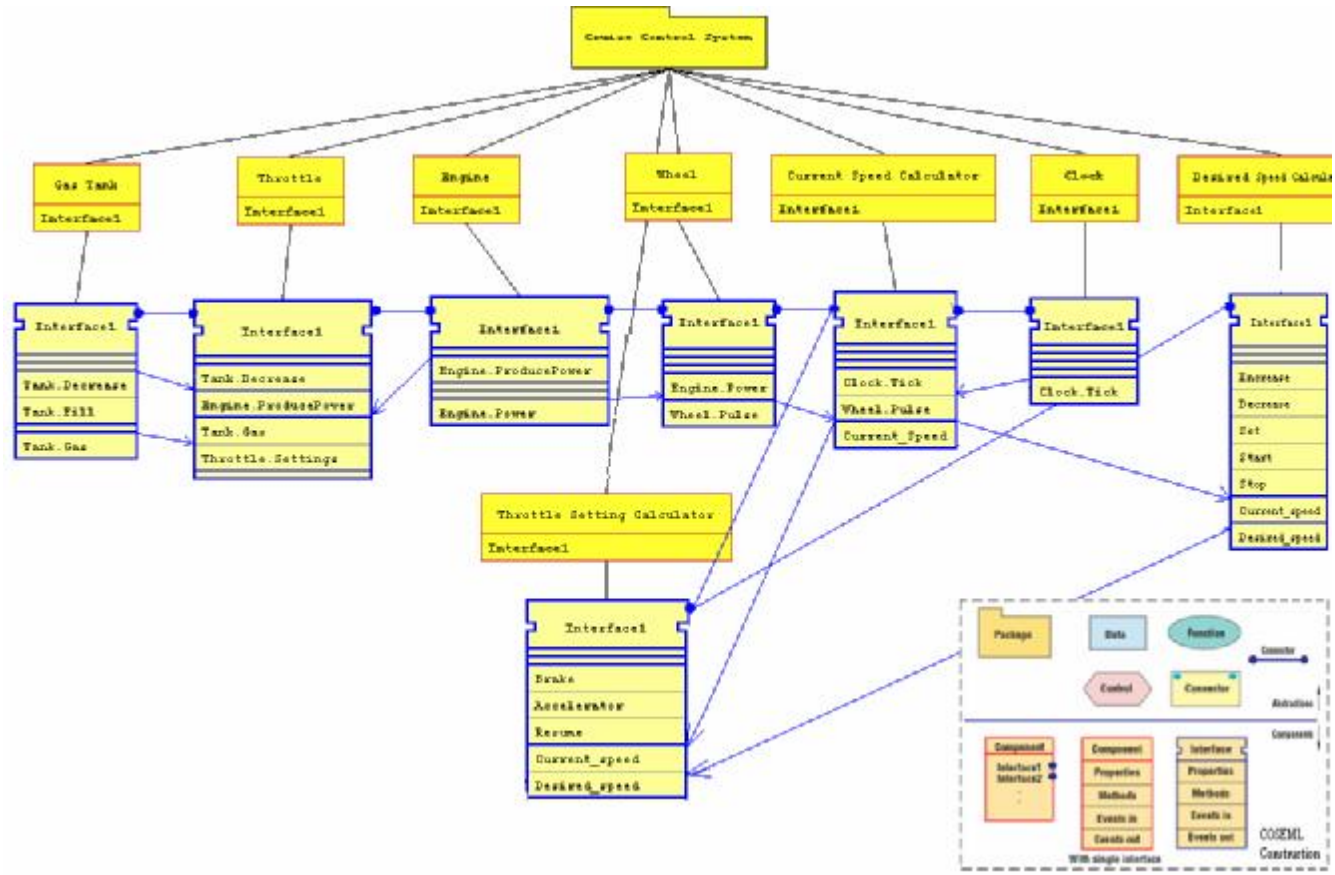

Figure 3: All components in Cruise Control Domain with COSEML Representation

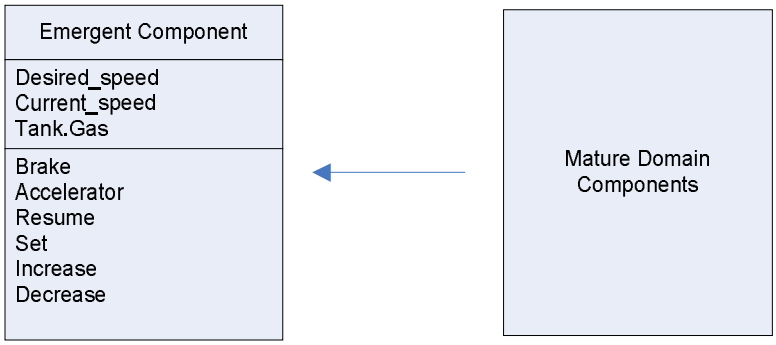

Figure 4: User Interface Emergent Component

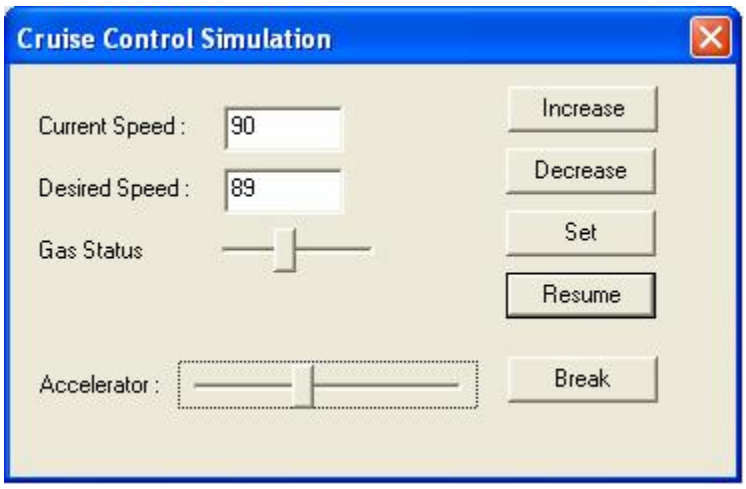

Figure 5: Cruise Control Simulation User Interface Component (Federate)

Table 1: Gas Tank Publish-Subscribe Check Table

Object Name	Component Name	Publish-Subscribe	Result
Tank.Gas	Gas Tank	P	OK
	Throttle	S	
Tank.Fill	Gas Tank	P	OK
Tank.Decrease	Gas Tank	P	OK
	Throttle	S	