DEADLOCK DETECTION IN HIGH LEVEL ARCHITECTURE FEDERATIONS USING AXIOMATIC DESIGN THEORY

Cengiz Togay[1], Veli Bicer[2], Ali H. Dogru[1]

[1] Middle East Technical University,
Department of Computer Engineering,
Ankara, Turkey
[2]Tepe Technology,
R&D Department, Turkey, Ankara

ctogay@ceng.metu.edu.tr (Cengiz Togay)

Abstract

In this study, we propose a method to translate the design matrices to Communicating Sequential Processes (CSP) codes in order to detect deadlocks in federations. Deadlock is an important problem to consider during the integration of systems. Both simulation and Component Oriented Software Engineering communities are considering the easier development of complex systems by integrating existing federates (components). Interfaces are the most important part of federate searching and standard interfaces do not include sufficient information about the internal behaviors of federates. Since interfaces should consist of design concepts, they should be created during design of federates. Axiomatic Design Theory (ADT) is a general design methodology that guides developers to decompose systems utilizing independence and information axioms. The Design matrix, which is a tool of axiomatic design, includes functional requirements, solutions, and dependencies among them. Since design matrices includes the internal behaviors of federates and it is a product of design, we applied the ADT to design Component Oriented Simulation systems. In our previous study, we proposed a method to find deadlocks using the design matrix. However, detection was left to the developer utilizing design matrices and it is very difficult to detect such deadlocks in complex systems. Deadlocks in federations can be figured out by utilizing Communicating Sequential Processes (CSP) formalism. We have used a CSP tool namely Failures-Divergence-Refinement (FDR2).

Keywords: Component Oriented, HLA, Deadlock Detection, and Communication Sequence Processes.

Presenting Author's biography

Cengiz Togay. He is doctoral candidate in the Middle East Technical University's Computer Engineering Department. He has worked on component-oriented simulations techniques, software engineering, Web Services, Semantic Web issues, and distributed systems. He obtained his MS in computer engineering from the Canakkale Onsekiz Mart University.

1 Introduction

Both simulation and Component Oriented Software Engineering (COSE) [1] communities try to easily build complex systems by combining existing components (federates) [2-4]. COSE approach is based on the integration of components through connection of their interfaces. Interfaces of federates are defined using one of the specific representations of the Object Model Template (OMT) [5] namely Simple Object Model (SOM). In our previous study, we represented SOMs as component interfaces [3]. Component interfaces do not have sufficient information to guide developers for ascertaining congruent components to build systems. The only published information about components is described in their interfaces (i.e. the signature). Their internal details and implementation mechanisms are not revealed. Therefore, developers have access only to interface of the components, some text documents describing the usage of components, and definitions such as the developer name, version number, etc. Component developer firms do not want to publish design artifacts because of confidentiality. Locating congruent components is one of the big problems of COSE and it requires machine readable design artifacts. The design artifacts should be prepared during design. Otherwise, developers can refrain to prepare them because of time and cost considerations. If the design artifacts are prepared after the implementation, information loss appears. Therefore, interfaces should be enriched with the required information in a machine readable form by considering confidentiality. In addition, this enrichment should be part of the design process.

Axiomatic Design Theory (ADT) [6] provides a broad and systematic approach to design systems through top-down decomposition. It proposes four domains, two design axioms, hierarchies, and zigzagging. These concepts are applied to the design matrix that is produced during the design process. The design matrix includes Functional Requirements (FRs) and Design Parameters (DPs) that represent the problem and solution spaces respectively and relationships among them. Axioms are used to define "Good Design". Components, their interface items (methods, events, and attributes), definition reasons for each of item, abstract definitions such as packages, data, function representations and connections among them can be represented in a design matrix in terms of COSE elements [7]. Therefore, components in COSE require an interface enrichment that design matrices provide. Also ADT provides guidance for decomposition. Based in ADT, if axioms of ADT applied, maintainable, cost effective, and modular design can be produced. The design matrix is a product of component oriented simulation development framework based on ADT [4]. COSE approaches are more efficient in mature domains. Congruent components are ascertained during the design of a system in mature domains. We proposed a method to ascertain components with evaluating their congruency in terms of interface conformance while considering dependency among interface items [4] and the information axiom [8]. COTS components (federates) often do not offer the exact functionality requisite by the system [9]. They integrate several services. Some services when selected, will require the incorporation of further other services due to dependencies. Since DMs provide dependency relations, DMs are helpful tools to locate components and component parts. Providing input-output dependency among component interfaces is also consistent with the "proof obligations" introduced in [10] for discovering interface and compositional inconsistencies.

When congruent federates are integrated, they should also be checked in terms of deadlock anomalies. Concurrently executing components can cause the unexpected run-time anomalies such as race conditions and deadlocks [11]. The undetected faults (dormant) during the formation a federation may lead to subsequent service failures [12]. Coupling is a sign of potential deadlocks and it can be extracted from the Design Structure Matrices (DSM) [13, 14]. Dependencies among federates can be represented using DSM. However, a DSM does not have the capability to describe the content of dependencies [15]. If federates are designed using COSE and ADT, design matrices of the federates can be used to check for potential deadlocks as figured out from the DSM and the design matrix [16]. We did not develop a specific technique for deadlock detection in our previous work [16], which is a very difficult task for complex systems.

A number of architecture description languages have been developed to describe concurrent systems such as WRIGHT [17], Rapide [18], and UniCon [19]. In Rapide developed system is checked in terms of user defined traces. Rapide does not verify all executions in the corresponding software system. We are proposing a method to check for deadlocks by referring to the design matrices and Communicating Sequential Processes (CSP) [20] representation. CSP is a process algebra and is supported by the Failures-Divergence-Refinement (FDR2) [21] tool. WRIGHT describes the system but the connector concept that defines the behaviors in WRIGHT is characterized using CSP [22]. Federates and their behaviors in design matrices are represented in CSP in terms of processes. Federations can be evaluated for formal checking for deadlock while utilizing FDR2 automatically. It should be noted that a deadlock in a system can also be a sign of a missing part. Each subscribed OMT item in a federate should be satisfied by a publisher federate otherwise a deadlock occurs with respect to CSP. Efficiency of decomposing system can have significant impact on decreasing resources for

verification [10, 23]. ADT guides with the hierarchically decomposition of designs that are not coupled. Therefore, ADT can be helpful to decrease the resources needed for verification of the system. In this study, we are proposing a method to translate design matrices of federates to CSP codes in order to detect deadlocks in federations and compatibilities among federates in terms of system requirements.

The rest of the paper is structured as follows. Section 2 provides a brief introduction to Axiomatic Design Theory, Section 3 describes CSP. Section 3 details the proposed method to translate design matrices to CSP representation with a case study.

2 Background

2.1 Axiomatic Design Theory

In order to design different types of systems such as machines, organizations or software, Axiomatic Design Theory (ADT) provides a framework in which the system is defined as an assembly of the subsystems, hardware, software, or people [6, 24]. These components mainly aim to operate together to accomplish a set of tasks. In the design process, a system is represented by different architectural elements such as the domains, hierarchies, or modules.

The design process is divided into four stages, namely customer domain, functional domain, physical domain, and process domain. The customer domain specifies the needs of the customers to be achieved by the system. These needs are then converted to the functional requirements (FRs) and the constraints (Cs) in the functional domain. By considering the FRs and the Cs, the design parameters (DPs) are provided in the physical domain. To realize the system by using the DPs, the process domain includes the process variables (PVs). Relations between these domains are expressed as "What" and "How" questions (e.g. what the customer wants (CN) is addressed by how it is accomplished (FR)).

In addition to specifying FRs and the corresponding DPs and PVs, the system design process continues further by hierarchical decompositions. By constructing FR, DP and PV hierarchies, the complexity of the system design process is divided into smaller components which can then be handled by different modules after the design is completed. However, the FRs and DPs cannot be decomposed independently by remaining in one domain. On the contrary, this process is done by zigzagging between the domains. The zigzagging is an important part of the axiomatic design to create hierarchies by enabling the parallel decomposition in all four domains. For example, once a FR1 is defined, the designer "zigs" to the physical domain to define its corresponding DP1. Then, the designer "zags" to the functional domain to decompose the FR1 into smaller FRs. This process continues until all the leaves of FRs are satisfied with the actual DPs.

Two important axioms, Independence Axiom and Information Axiom, are introduced by ADT in order to obtain an effective design. The Independence Axiom states that the functional requirements should be independent from each other for an ideal design. In addition, Information Axiom states that the aim of a good design should be to minimize the information in the content. Through a design matrix, we can show how a set of FRs fulfills the set of CNs, how a set of DPs fulfill the set of FRs, and how the set of PV accomplish the set of DPs. According to the design matrix, a design can be in one of the following forms:

- Uncoupled Design: The design is in the ideal case. Each FR is satisfied by one DP so that a diagonal design matrix is produced.
- Decoupled Design: This is the most common form of the design. The design matrix is triangular in which the relationships are placed at only one of the sides of the diagonal in the design matrix.
- Coupled Design: The relationships are distributed on the design matrix, indicating a highly interdependent design.

Information axiom can also be applied to measure congruency among components in terms of probability of success [8]. ADT has been applied to software systems in various studies [6, 25-29], to COSE [7, 16] and to High Level Architecture (HLA) based simulation with provided enrichment in interfaces [3, 4]. Collaboration diagrams are applied to ADT to specify dependencies among components, methods, or attributes [7].

2.2 Communicating Sequential Processes

Communicating Sequential Processes (CSP) is a process algebra introduced by Hoare [20]. CSP is a language and is supported by the tools: Failures-Divergence-Refinement (FDR2) [21] for model checking and Process Behavior Explorer (ProBE) [30] for state machine based models. Wright [31, 32] an architecture description language uses CSP like notation to describe components' ports and roles. For instance, HLA Runtime Infrastructure (RTI) [33] is formalized using Wright to detect deadlocks and race conditions [11]. It should be noted that developed tools translate the Wright representation to CSP for utilizing the FDR tool. CSP can also be used for modeling complex service choreography for checking for deadlock among integrated services [34, 35]. In CSP, processes defined statically include a set of events. Events are atomic and provide synchronization among processes. They are used to define the behavior of processes. More than one process can be executed in a time in concurrent systems. This causes well known problems such as deadlocks. CSP theory and FDR2 are used for checking defined processes in terms of traces, stable failures, and failure-divergence models. In this article, we will concentrate on the

traces to check for the deadlock situation in the composed system using the FDR2 tool. CSP expressions that will be used in this article are listed in Tab. 1.

Tab. 1 CSP expressions used in this article

CSP Expression	Explanation
P[| A |] Q	P and Q processes are partial interleaved parallel composition. A is the set of the events. If A is empty then composition of P and Q behaves interleaved parallel.
P ||| Q	P and Q are interleaved parallel
e-> P	Event e performed first and then Process P is executed after an external trigger occurred.
SKIP	Successfully termination
STOP	Deadlock
Datatype x = a | b | c	Defines x datatype with a set of alternatives
Channel e	Defines event e
Channel e:x	Defines event e with x datatype
e ? a	Defines input on event e of an item defined during channel definition. As defined in datatype, instead of an item, b or c items can be used.
e ! a	Defines output on event e of an item. After this expression is performed, e?a expression in another process in waiting situation can be performed. Input and output expressions are used to provide synchronization.
Union	Unions the sets.

3 Translation from Design Matrix to CSP

In our previous studies [3, 7], we preferred to use COSEML notation [1, 36] as shown in Fig. 2 which is very similar to CORBA Component Model [37] to represent component interfaces in design matrices. Interfaces define the connection points to integrate components. During the integration only methods and component events are shared among components. In this study, a process concept in CSP corresponds to a partial or a whole component. Methods and component events are defined as events in terms of CSP and they are represented in a process. In COSEML notation, published interactions are located within "Method in", published events are found in "Event in", subscribed interactions are located within "Method out", and lastly subscribed events are placed within "Event out" as shown in Figures 1 and 2. Input and output definitions in COSEML notation are represented in CSP as shown in Tab. 2.

Tab. 2 COSEML and CSP representations

COSEML representation	CSP representation
Component	Process
Published method	Output event (e ! a)
Subscribed method	Input event (e ? a)
Published event	Output event (e ! a)
Subscribed event	Input event (e ? a)

As an example, composition of Federate 1 and Federate 2 are represented in a federation design matrix as depicted in Fig. 1. Corresponding COSEML representation is shown in Fig. 2. There are two federates namely Federate 1 and Federate 2 in this example federation. There are two interactions and four events of the Federate 1. They are organized based on the publish-subscribe mechanism. 'X's in design matrix represents the dependencies among FRs and DPs. Each FR is satisfied with at least one DP that is located in diagonal line. For instance, definition of interaction 1 is represented in FR1.1.1 and it is satisfied with DP1.1.1. Other 'X's in the same line represent the dependency of *Interaction 2* with others. Such as, to publish *Interaction 2*, DPs *Interaction 1* and *Event 1* are required as shown in Fig. 1. Dependency between federates are also depicted in the design matrix. For instance, *Interaction 1* in Federate 1 has subscribed to Federate 2 through *Interaction 1* in Federate 2. Similarly, Federate 2 has two interactions and two events. *Interaction 1* in Federate 2 is published if *event 1* is provided by another federate (Federate 1). It can be figured out that *Event 1* is published by the federate 1 without any dependency. Therefore there is no cycle between Federate 1 and Federate 2 in terms of *Interaction 1* and *Event 1*. Although for small designs cycles among OMT items can be figured out with human eye, we need tools. The design matrix tool provides us with such a capability. Our ADT tool can detect couplings that are signs of deadlocks that are represented in Fig. 1 as black boxes. Federates 1 and 2 are coupled in terms of DSM since they are sharing events and interactions to publish an item that other federate requires. Although ADT does not propose coupled designs based on the independence axiom, couplings can occur during integration especially between federates observable in the DSM. Coupling does necessarily cause deadlocks, it is only an indication [16]. In our example, the federation includes two decoupled federates and the federation is coupled. Although composition of federates are coupled, the federation is deadlock free.

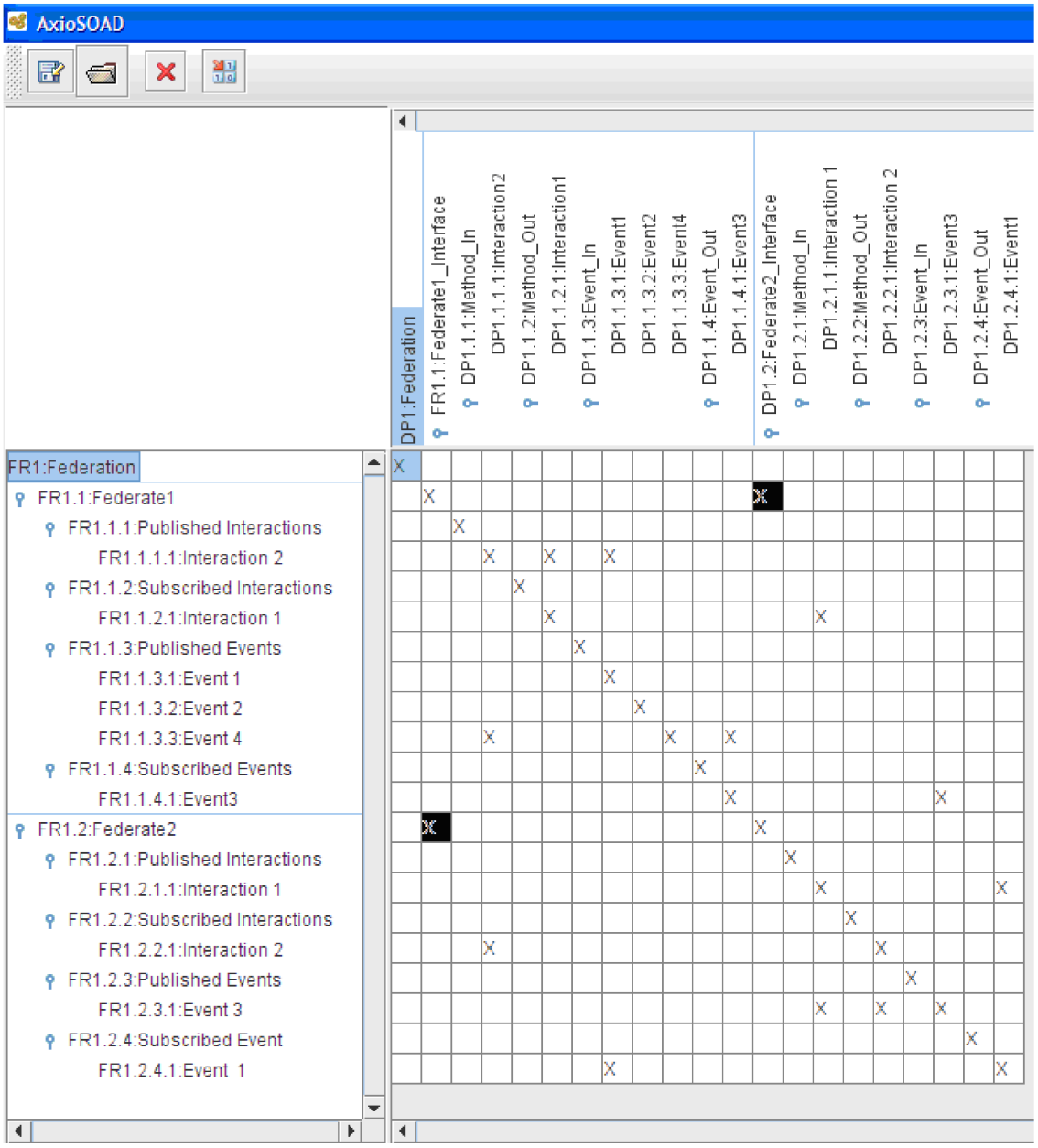

Fig. 1 A Federation consists of Federates 1 and 2

Federate interfaces are translated to CSP codes based on the following rules:

- Only published interactions and events are defined as processes. Each publisher must have at least one subscriber. If there is no subscriber of an OMT item then related process can be omitted.
- Input and output definitions are specified based on dependence relationships of published interactions or events. For instance, *Interaction 1* requires *Event 1* in Federate 2 and is represented as "Event1? e1 -> Interaction1! m1."
- A Federate is represented as a process that consists of one or more sub processes as shown in Tab. 3.
- A federation is also represented as a process and it is formed from one or more federate processes.
- Processes are composed based on shared methods or events among processes. If there is no shared item(s) than the "|||" term is used to connect processes. If there are, then "[| |]" is used.
- Shared items among processes are looked up from the design matrix.
- If there are events defined as input events (subscribe) and required output events (publish) from other federates, they must be considered during the federation process is forming.

ISBN 978-3-901608-32-2

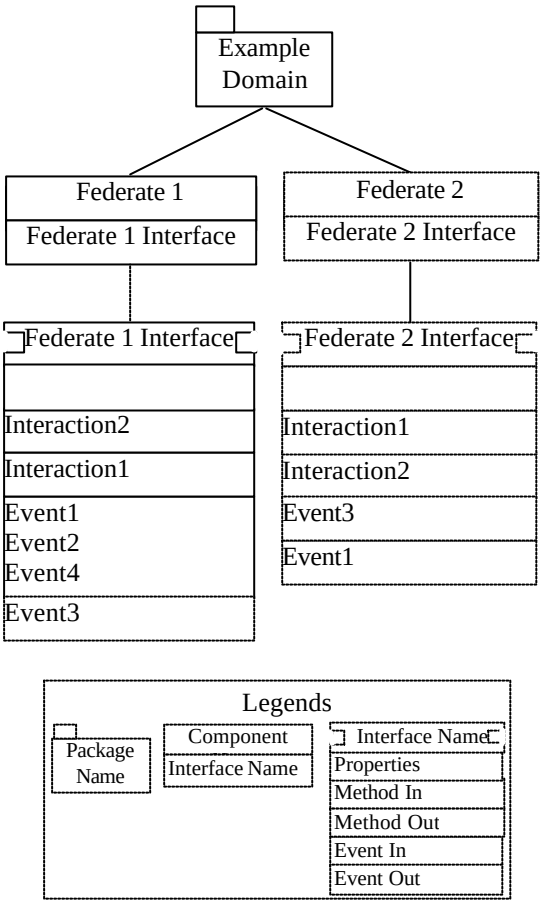

Fig. 2 COSEML representation of Federate 1 and Federate 2

CSP codes of Federate 1 and Federate 2 are listed in Tab. 3. Federate 1 has four published items therefore there are four sub processes. Only *Interaction 2* is shared between FEDERATE1_SUB1 and FEDERATE1_SUB2. Processes FEDERATE1_SUB3 and FEDERATE1_SUB4 can be executed asynchronously. FEDERATE1_SUB4 is omitted since there is no subscriber process. The Federate 2 has two published items therefore there are two sub processes. Only *Interaction 1* is shared between FEDERATE2_SUB1 and FEDERATE2_SUB2. When we try to execute one of these federates, FDR2 tool will notify us about the deadlock. We can conclude from this message that some interaction or events are not satisfied in the processes. Composition of the Federate 1 and Federate 2 forms a federation that is represented as FEDERATION in Tab. 3. *Interaction1*, *Interaction 2*, *Event 1*, and *Event 3* are shared between federates as listed in Tab. 3.

We tested the executable CSP codes in Tab. 3 and obtained a deadlock free federation. Although, coupling is available between the Federate 1 and the Federate 2 as shown in the design matrix, we can conclude that all OMT items which are required by a federate, are satisfied by a federate in the federation, and dependency relations between OMT items are not allowed to form cycles.

Tab. 3 CSP representations of Federate 1 and Federate 2 in a federation

datatype D_i1= i1, D_i2= i2, D_e1= e1, D_e2= e2, D_e3= e3, D_e4= e4

channel Interaction1:D_i1, Interaction2:D_i2, Event1:D_e1, Event2:D_e2, Event3:D_e3, Event4:D_e4

----------Federate 1----------

FEDERATE1_SUB1 = Event1?e1 -> Interaction1?i1 -> Interaction2!i2 -> FEDERATE1_SUB1

FEDERATE1_SUB2 = Interaction2?i2 -> Event3?e3 -> Event4!e4 -> FEDERATE1_SUB2

FEDERATE1_SUB3 = Event1!e1 -> FEDERATE1_SUB3

--FEDERATE1_SUB4 = Event2!e2 -> --FEDERATE1_SUB4 there is no subscriber

FEDERATE1 = (FEDERATE1_SUB1 [|{|Interaction2|}|] FEDERATE1_SUB2) || FEDERATE1_SUB3 *--||| FEDERATE1_SUB4*

----------Federate 2----------

FEDERATE2_SUB1 = Event1?e1 -> Interaction1!i1 ->FEDERATE2_SUB1

FEDERATE2_SUB2 = Interaction1?i1 -> Interaction2?i2 -> Event3!e3 -> FEDERATE2_SUB2

FEDERATE2 = (FEDERATE2_SUB1 [|{|Interaction1|}|] FEDERATE2_SUB2)

----------Federation----------

FEDERATION=(FEDERATE1[| union(union(union ({|Event1|}, {|Interaction1|}), {|Interaction2|}), {|Event3|})|] FEDERATE2)

4 Conclusion

In federations, deadlocks can appear because of shared OMT objects among federates. Dependency relationships among objects can be used to detect deadlocks. Design matrices include the definition reasons of the OMT objects, and their dependencies with other OMT objects. The design matrices are the product of a framework based on Axiomatic Design Theory. In our previous study [16], we proposed a method to find deadlocks utilizing design matrices for small problem sets. This framework is used to design federates and federations. During design, coupling in the design matrix is an indication for possible deadlock situations; however, deadlock will not necessarily occur. CSP and supporting tools are used to check defined processes in terms of traces, stable failures, and failure-divergences models. Since human does not detect deadlocks in complex systems, we propose a method to translate design matrices to CSP for utilizing FDR2 tool to automatically check for deadlocks, in this article.

5 References

[1] A. H. Dogru and M. M. Tanik, "A Process Model for Component-Oriented Software Engineering," in *IEEE Software*, vol. 20, 2003, pp. 34-41.

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

[3] C. Togay, A. H. Dogru, U. J. Tanik, and G. J. Grimes, "Component Oriented Simulation Development With Axiomatic Design," presented at The Ninth World Conference on Integrated Design and Process Technology, San Diego, CA, 2006.

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

[5] IEEE Std. 1516.2-2000, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Object Model Template (OMT) Specification," 2000.

[6] N. P. Suh, *Axiomatic Design: Advantages and Applications*. New York: Oxford University Press, 2001.

[7] C. Togay and A. H. Dogru, "Component Oriented Design Based on Axiomatic Design Theory and COSEML," *Proceedings of ISCIS, Lecture Notes in Computer Science*, vol. 4263/2006, pp. 1072-1079, 2006.

[8] C. Togay, O. Aktunc, M. M. Tanik, and A. H. Dogru, "Measurement of Component Congruity for Composition Based on Axiomatic Design," presented at The Ninth World Conference on Integrated Design and Process Technology, San Diego, CA, 2006.

[9] L. Iribarne, J. M. Troya, and A. Vallecillo, " Selecting software components with multiple interfaces," presented at Euromicro Conference, 2002.

[10] M. Rangarajan, P. Alexander, and N. B. Abu-Ghazaleh, "Using automatable proof obligations for component-based design checking," presented at IEEE Conference and Workshop on Engineering of Computer-Based Systems, 1999.

[11] R. J. Allen, D. Garlan, and J. Ivers, "Formal modeling and analysis of the HLA component integration standard," presented at 6th ACM SIGSOFT international symposium on Foundations of software engineering, 1998.

[12] A. Avizienis, B. Randell, and C. Lanwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, pp. 11-33, 2004.

[13] D. Steward, *System Analysis and Management: Structure, Strategy and Design*. New York: Petrocelli Books, 1981.

[14] T. R. Browning, "Applying the design structure matrix to system decomposition and integration problems: a review and new directions," *IEEE Transactions on Engineering Management*, vol. 48, pp. 292-306, 2001.

[15] Q. Dong and D. E. Whitney, "Designing A Requirement Driven Product Development Process," presented at 13th International Conference on Design Theory and Methodology, Pittsburgh, PA, 2001.

[16] C. Togay, G. Sundar, and A. H. Dogru, "Detection of Component Composition Mismatch with Axiomatic Design," presented at IEEE Southern Conference, Memphis, TN, 2006.

[17] R. Alan and D. Garlan, "The WRIGHT architectural specification language," Carnegie Mellon University, School of Computer Science, Technical Report CMU-CS-96-TBD, 1996.

[18] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann, "Specification analysis of system architecture using rapide," *IEEE Transactions on Sofware Engineering*, vol. 21, pp. 336-355, 1995.

[19] M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik, "Abstractions for Software Architecture and Tools to Support Them," *EEE Transactions on Software Engineering*, vol. 21, pp. 314 - 335 1995.

[20] C. A. R. Hoare, "Communicating Sequential Processes," *Communication of the ACM*, vol. 21, pp. 666-677, 1978.

[21] Formal Systems Ltd., "Failures-Divergence-Refinement: FDR2 User Manual," 2003.

[22] R. Allen and D. Garlan, "Formalizing architectural connection," in *Proceedings of the 16th International Conference on Software Engineering*, 1994, pp. 71-80.

[23] J. M. Cobleigh, G. S. Avrunin, and L. A. Clarke, "Breaking up is hard to do: an investigation of decomposition for assume-guarantee reasoning," presented at Proceedings of the 2006 international symposium on Software testing and analysis, 2006.

[24] N. P. Suh, "Axiomatic Design Theory for Systems," *Research in Engineering Design*, vol. 10, pp. 189-209, 1998.

[25] S. H. Do and G. J. Park, "Application of Design Axioms for Glass-Bulb Design and Software Development for Design Automation," presented at Third CIRP Workshop on Design and Implementation of Intelligent Manufacturing, Tokyo, Japan, 1996.

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

[27] S. H. Do and N. P. Suh, "Object Oriented Software Design with Axiomatic Design," presented at Proceedings of ICAD2000 First International Conference on Axiomatic Design, Cambridge, June 2000.

[28] S. H. Do and N. P. Suh, "Systematic OO Programming with Axiomatic Design," in *IEEE Computer*, vol. 32, October 1999, pp. 121-124.

[29] B. Gumus and A. Ertas, "Requirement Management and Axiomatic Design," presented at Integrated Design and Process Technology Symposium, 2004.

[30] Formal Systems Ltd., "Process Behavior Explorer: Probe User Manual," 2003.

[31] R. J. Allen and D. Garlen, "A Formal Approach to Software Architecture," Carnegie Mellon University, Ph.D. Thesis CMU-CS-97-144, 1997.

[32] R. Allen and D. Garlen, "A formal basis for architectural connection," *ACM Transactions on Software Engineering and Methodology*, vol. 6, pp. 213-249, 1997.

[33] IEEE Std. 1516.1-2000, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)- Federate Interface Specification," 2000.

[34] W. L. Yeung, J. Wang, and W. Dong, "Verifying Choreographic Descriptions of Web Services Based on CSP," presented at The IEEE Services Computing Workshops, 2006.

[35] W. L. Yeung, " Mapping WS-CDL and BPEL into CSP for Behavioral Specification and Verification of Web Services," presented at Web Services ECOWS '06, 2006.

[36] A. H. Dogru, "Component Oriented Software Engineering Modeling Language: COSEML," presented at Computer Engineering Department, Middle East Technical University, Turkey, TR 99-3, 1999.

[37] CORBA, "CORBA Components Specification 3.0," 2002.