Component Replacement Strategies for Information Systems Reengineering

Malleswara Talla, Ph.D.
John Molson Business School, Concordia University, Montreal, Canada

Raul Valverde, Ph.D.
John Molson Business School, Concordia University, Montreal, Canada

ABSTRACT
Recent trend in systems architecture and design is component-based. A system is designed as a set of mutually supporting components that provide the intended services. The requirements models such as business type models and use case models are often used for deriving the targeted component-based architecture. The component interfaces are derived via sequence diagrams, collaboration diagrams and context diagrams. As the business model evolves, it becomes vital that the system also needs to match the business evolution whether it involves changing business rule set or growth in volume of business transactions. Timely reengineering of systems is profitable to any organization. The systems reengineering can be conducted in a pragmatic manner via component by component or a selected set of components; it becomes manageable and cost-effective to maintain the system and to train only a smaller sample of affected users. This paper offers a methodology for system reengineering via component replacement and model-view-control framework for component refinement and evolution in order to achieve a reengineered system that reflects upon the latest requirements in business domain.

Keywords
Information System, Reengineering, Business Type model, Use Case, Component-based model, Interface model, Context model, MVC framework.

1. INTRODUCTION
Contemporary methodology for software systems engineering is component-based. Current enterprise systems modeling often use technologies such as Common Object Request Broker Architecture (CORBA), Component Object Model (COM), Easy Java Simulations (EJS), etc. An information system as a *component-based model* is a set of mutually supporting software components that provide the intended services. A component provides a service as a modular piece of a software system that communicates with other components via interfaces. A component model unambiguously specified interfaces and enables interactions among the participating components [10]. Component-based systems architecture allows a pragmatic system development or system reengineering as only few components can be focused. The system *reengineering* is in fact analyzing and adapting the system to current business rules and performance requirements. When component interfaces are not well documented, reverse engineering is the only tool to understand and remodel the system components [2]. The component models leave enough scope for adaptation to changing needs and business rules [3].

The component-based systems modeling provides an opportunity to expedite the systems development via Open Source or Commercial Off-The-Shelf (COTS) components that are readily available [4]. The component based software modeling allows an incremental system specification that can adapt to a distributed component information systems [5]. An *information system* should efficiently evaluate and store information for better decisions in an organization, considering its suppliers, customers, and other collaborating organizations. The component-based software models can support real time decisions via component selection at runtime [6]. Similarly, the component based systems modeling allows an efficient system design, development, implementation, and subsequent reengineering. The component based methodology allows a legacy information system to be migrated to a modern distributed information system via reengineering [11].

This paper proposes few component replacement strategies and methodology for system reengineering an information system, taking into account the complexity of components. The following sections detail the methodology and component reengineering strategies to achieve targeted reengineering of an information system.

2. COMPONENT-BASED ARCHITECTURE
A component is a piece of software that offers a specific service. An information system can be designed as a set of mutually supporting components that provide the intended services. The component-based systems development methodology uses *business type* and *use case* models for developing the requirements for its system components. The business type requirements modeling can be performed in the following steps [7]:

- Gather information of interest, type of information, and limitations such as maximum and minimum values,
- Create a conceptual map of information while eliminating redundant information,
- Conceive the system components.

Then, a business type model is created as a conceptual map of all related data and information as shown in figure 1.

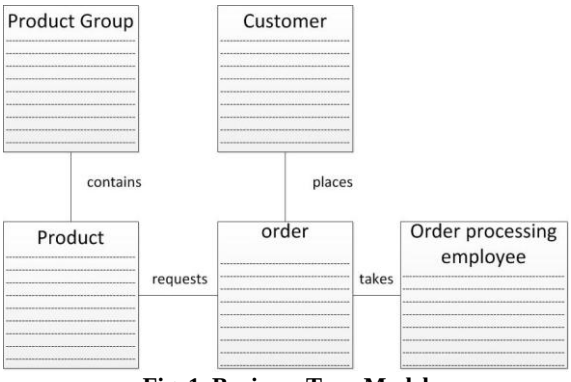

Fig. 1. Business Type Model

Indeed, a system is a set of requirements in terms of user inputs and expected outputs of the system. The system usage scenarios and business events detail how system should respond to the events [8]. A Use Case is a description of a user interaction with the system [7]. An *use case* contains the participating *actor*s where *actor* could be anything (a user, a role, a person) internal or external that interacts with system [9]. The figure 6 presents a set of users receiving the service of a sales system.

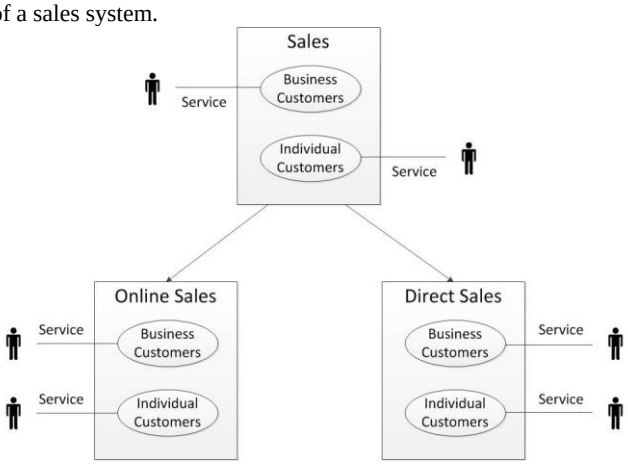

Fig. 2. Use Case Model

The challenge in component-based systems architecture is to reuse as many existing components and minimizing the need for new components. In order to arrive at the requirements of a component-based system, the business type models and use case models can be used in an effort to identify component specifications, interfaces and their dependencies [1]. The figure 2 identifies how a component-based architecture is created starting with the requirements models.

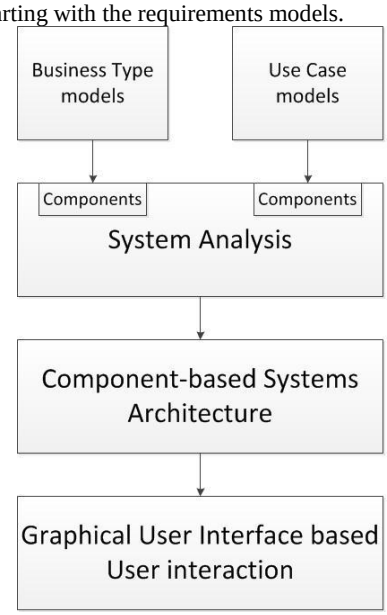

Fig. 3. Conceiving the component-based systems architecture

An information system can be viewed as a set of collaborating components. Each component acts upon a set of input parameters and states, and provide an interface to other components. The component interfaces in the interface diagram have operations with data signatures and use the terminology defined in the business type model or use case model. The component architecture model should define all components, their responsibilities, specifications, interfaces and dependencies. The figure 3 presents a set of components interacting among one another that constitute a component-based systems architecture. In figure 3, the incoming arrows represent input to a component and an outgoing arrow presents a result or a service offered by that component. The overall system provides the expected results.

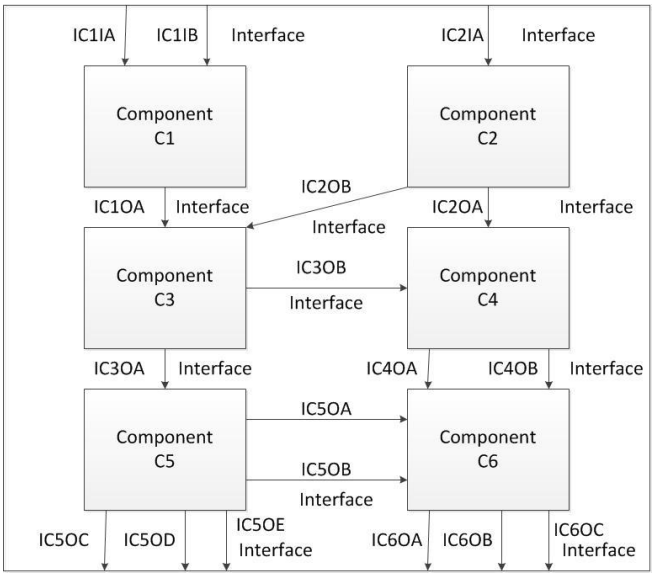

Fig. 4. Component-based system architecture

Once the candidate component architecture is created, the behaviour of each component and overall system can be analysed in more detail. At this stage, the system functionality is defined through component interfaces via *component interaction diagrams* that specify messages or data exchanged between the collaborating and communicating objects as shown in figure 5.

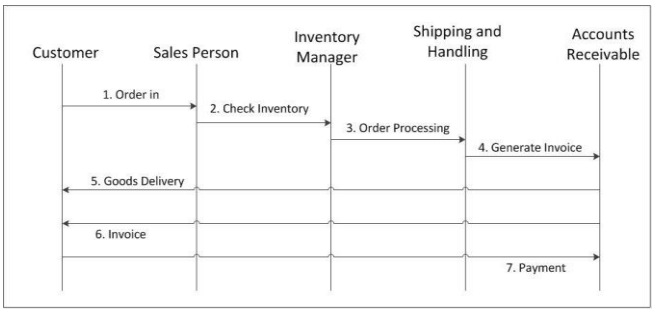

Fig. 5. Sequence Diagram

The component collaborations are presented in *collaboration diagrams* as shown in figure 6 that help context modeling of a use case.

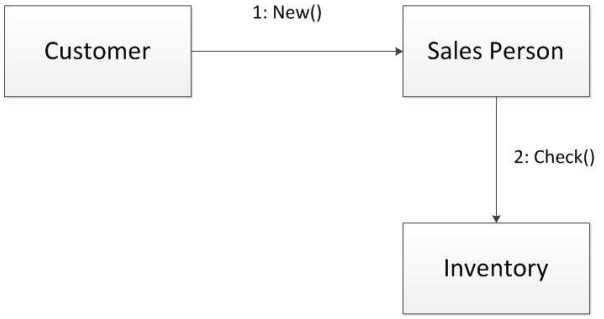

Fig. 6. Collaboration Diagram

The functionality resulted among these interactions can be characterized in *context diagrams* [7]. A context model is a high-level view of business context as procedures, roles and responsibilities. A context model can be used for describing the complete system in the context of business domain. The figure 7 presents the details of processing a use case, as a set of sequential and parallel processing procedures that reflect upon the business context of an information system.

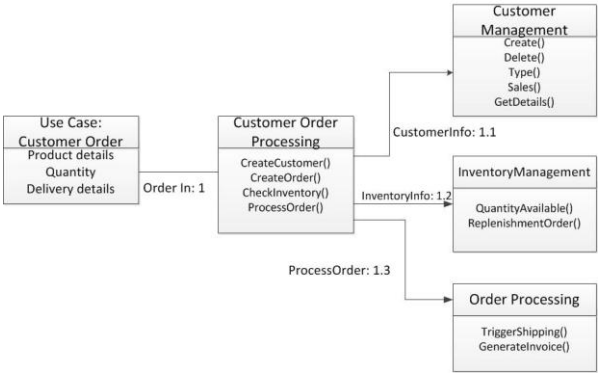

Fig. 7. Context Model

At a component interface, a set of parameters, data structures, messages, or procedures as an input trigger a certain processing, or as an output for results or input to other component. The component interfaces and collaborations among components are shown in figure 4. For each operation, its parameters are defined by making use of appropriate types, namely, *pre-condition* and *post-condition* behaviours [7].

3. COMPONENT REPLACEMENT

The component-based system reengineering is accomplished via redesigning or revising an existing component for the required changes. The target component can also be replaced with another component with an equivalent new component that offers the same service. Redesigning an existing component may require reverse engineering when the existing software system is not fully documented. The overall effort for redesigning a component can be expensive. In such cases, component replacement could be a better strategy for system reengineering. It is often difficult to find another component that offers an exact interface. However, it may be possible to take only the required services and ignore irrelevant services at the interface. A *glue code* can be triggered for transforming the interface for the intended results. The component replacement strategy for system reengineering requires a through analysis since the component may require all relevant inputs for producing the required output. At the component interface level, if an input parameter is not needed, it can be ignored and be left to a default value. While replacing a component with another that offers similar but not exactly the same service, a trivial *glue code* can help revising the service as required. The *glue code* framework is similar to the control part of Model-view-Control (MVC) framework, and turns the component services and results that meet the user expectations. The framework involves a *model* as a set of communicating classes, *view* as a graphical user interface (GUI) to the user, and *control* as a communication control. The figure 8 depicts a MVC framework where the *glue code* is on the *model* side in (a) and on the *view* side in (b).

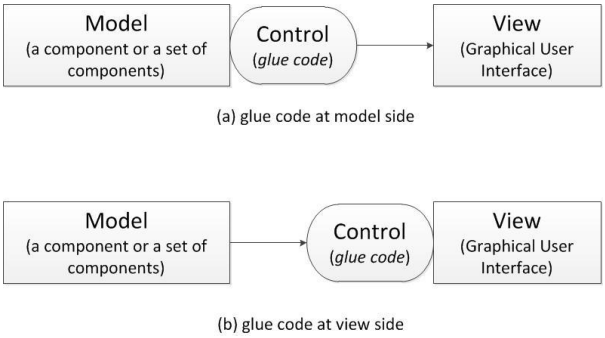

Fig. 8. MVC framework

A carefully documented component provides all data and information regarding the service offered and how it is accomplished. Such documentation serves as a basis for evaluating the feasibility of using the component as a replacement for an existing component for system reengineering. The component replacement strategy for system reengineering can be considered based on complexity as follows:

a) One-to-One component replacement,
b) One-to-Many component replacement,
c) Many-to-One component replacement,
d) Many-to-Many component replacement.

The size and complexity of a component may require either grouping of simple components into one, or dividing into multiple components that provide the same service. Let us consider the component-based system architecture in figure 4 that consists of six components with well defined interfaces for evaluating the component replacement strategies as detailed below. Suppose the component C6 consists of classes that are simple, such component can be replaced with a new component NC6 as shown in figure 9. While implementing such replacement, the interface provided by the new component NC6 should be exactly same as the interface of C6. Due to system evolution if any of the input interface elements (IC4OA, IC4OB, IC5OA, and IC5OB) or output elements (IC6OA, IC6OB, and IC6OC) become redundant, such elements should be omitted in the new component NC6. Conversely, if any of these elements require some kind of input or output transformation, the *glue code* can be implemented.

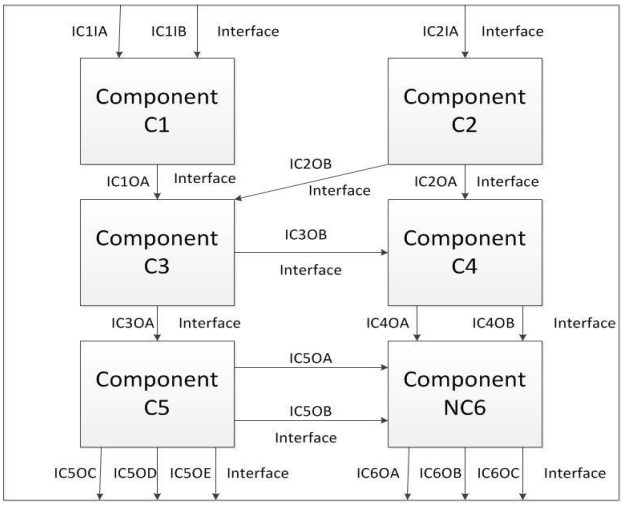

Fig. 9. One-to-One Component Replacement

When a component is very complex and an equivalent component is not available, then a set of components that collectively match the services offered by the existing one, it can be replaced with those new components by carefully reengineering the interfaces for accomplishing the same service. In figure 10, component C6 is replaced with two new components NC61 and NC62 which receive their required interfaces from other components that collectively provide an exact interface of C6. Such system reengineering simplifies system architecture and enhances the scope for better system maintenance and evolution. When several simple components collectively can be replaced with a moderate component that offers all services, it is worthwhile to use the new set of components to replace the existing complex component in an effort to reengineer the system that accomplishes the same service.

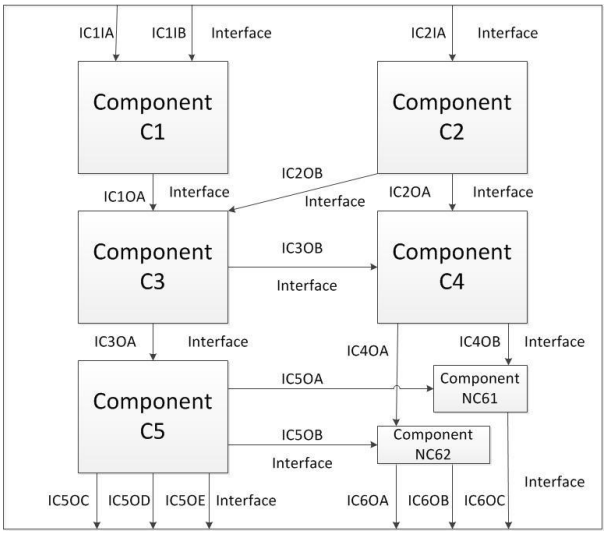

Fig. 10. One-to-Many Component Replacement

In figure 11, the functionality of components C5 and C6 is replaced with a new component NC5C6 while accomplishing the same interface and services offered by both C5 and C6. It should be noted that the services should be directed to the target components or users selectively, so that the end results of both C5 and C6 components match the overall system.

When the above strategies are not possible and a group of components that define a service can be collectively replaced with a new group of components that accomplish the same service.

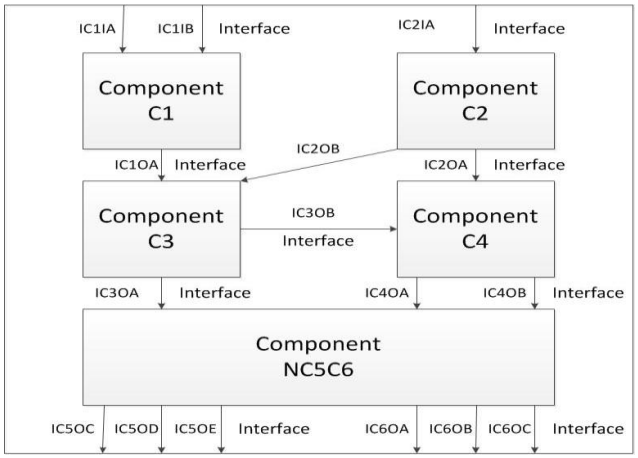

Fig. 11. Many-to-One Component Replacement

The resulting reengineered system will adapt a multitude of aspects of the system. In figure 12, the components C4, C5 and C6 are replaced with a new component NC41, NC42, and NC5NC6 reflecting upon the same final interface of both C5 and C6.

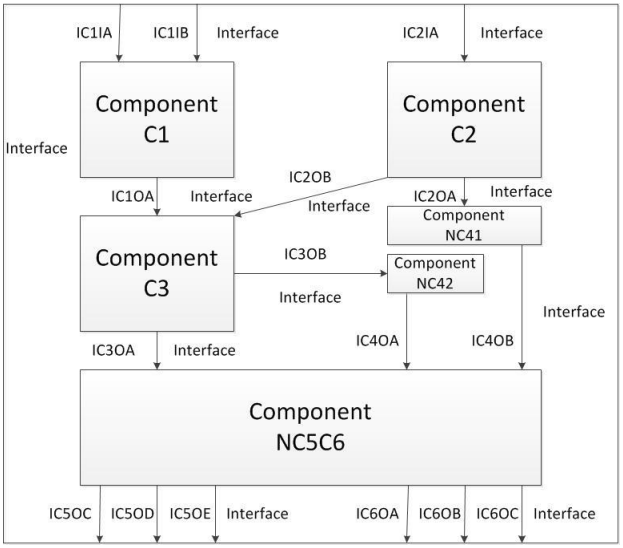

Fig. 12. Many-to-Many Component Replacement

A careful system reengineering involves a keen evaluation of all above strategies to choose the one which is the most appropriate and economical. The component simulators can be used for testing an existing component functionality, or a new component results for faster development and reengineering of a software system.

4. CONCLUSIONS

In this paper, the component based modelling methodology is detailed for system architecture and design; while several strategies for replacing the components are presented for system reengineering. The concepts and strategies in this paper can be used for new systems as well as for legacy systems reengineering. If an existing system is component-based, the system components can be replaced with new components that use the latest business rule set; however if the existing system is a legacy system, component-based modelling allows reengineering via plugging equivalent components and an on-going system reengineering. The main contribution of this paper is in reengineering via component replacement while balancing the complexity among the components. Such system reengineering enhances the scope for efficient system maintenance and evolution. When a component is processing-intensive, such component can be divided into multiple components all of which may exist within the system, or distributed over a network. The *glue code* proposed in this paper allows a client/server architecture for reengineering. Using these concepts, a legacy system can be reengineered to adopt a new client/server architecture. Further work may involve in evaluating component-based modelling to benefit from network oriented client/server architecture where most common components can be centralized at server side whereas the user interface components can be located at client side. Such component-based modelling opens a gateway for numerous application systems reengineering.

5. REFERENCES

[1] Cheesman, J. & Daniels J. (2001), "*UML Components: Simple Process for Specifying Component-based Software"*. Addison Wesley. N.J., US

[2] Landry Chouambe, Benjamin Klatt, and Klaus Krogmann, (2008), "Reverse Engineering Software-Models of Component-Based Models", *Proc. IEEE Conf. on Software Maintenance and Reengineering",* 978-1-4244-2157-2, pp 93-102.

[3] Xin Peng, Yijian Wu, Wenyun Zhao (2007), "A feature oriented adaptive component model for dynamic evolution", *Proc. IEEE conf. on Software Maintenance and Reengineering",* 0-7695-2802-3/07.

[4] Pasquale Ardimento, Giovanni Bruno, Danilo Caivano, Giuseppe Visaggio (2007), "A maintenance oriented framework for software components characterization", *Proc. IEEE Conf. on Software Maintenance and Reengineering",* 0-7695-2802-3/07.

[5] Nasreddin Aoumeur, Kamel Barkaoui, Gunter Saake (2007), "Incremental specification validation and runtime adaptativity of distributed component information systems", *Proc. IEEE Conf. on Software Maintenance and Reengineering",* 0-7695-2802-3/07.

[6] Biplav Srivastava (2004), "A feature oriented adaptive component model for dynamic evolution", *Proc. IEEE Conf. on Software Maintenance and Reengineering",* 0-7695-2802-3/07.

[7] Brown, A.W. 2000, *Large-Scale, Component-based Development*, Prentice Hall, New Jersey.

[8] Whitten, J. L., Bentley D. L. and Dittman K.V. 2000, *Systems Analysis and Design Methods*, McGraw-Hill, New York.

[9] Rumbaugh, J. 1994, 'Getting started: Using use cases to capture requirements', *Jour. of Object-Oriented Programming*, vol.7, no.5, September, pp. 8-1-,12,23.

[10] Heineman G. T. and Councill W.T (2001), Definition of a Software Component and its elements, Ch.1, Component Based Software Engineering , Addison-Wesley.

[11] Serrano, M., and Carver, D., de Oca, C. (2001), "Reengineering legacy systems for distributed environments", *Journal of Systems and Software*, 64(1), 37-55.

6. AUTHORS PROFILE

Malleswara Talla is a Professor (sessional) in the department of Decision Sciences & MIS at John Molson School of Business (JMSB), Concordia University, Montreal. He received B.Tech. degree from J.T.U. College of Engineering, Kakinada, India in 1979, a M.Tech. degree in 1981 from I.I.T., Kharagpur, India, and a Ph.D. degree from Concordia University, Montreal, in 1995 specializing in Computer Communications and Networks. He worked for Tata Consultancy Service (TCS), Bombay, and Societe Internationale de Telecommunications Aeronautique (S.I.T.A), Montreal for several years. Dr. Talla managed several projects involving data communications, computer networks, and business performance excellence. Dr. Talla is a member of Canadian Operations Research Society (CORS), Professional Engineers of Ontario (PEO), Institute of Electrical and Electronics Engineers (IEEE), Project Management Institute (PMI), The Association for Operations Management (APICS), and The Institute for Internal Controls (THEIIC). His teaching and research interests are mainly in Operations Management, Management Information Systems, Systems Re-engineering, Business Intelligence, Data Communications and Computer Network, Software Engineering and Evolution, Software Architecture, Design, and Development. Dr. Talla is a registered professional engineer in Canada.

Raul Valverde is working as a Professor in the department of Decision Sciences & MIS at John Molson School of Business (JMSB), Concordia University in Montreal. He holds a Bachelor of Science degree in Mathematics and Management from Excelsior College (US), a M. Eng. degree in Electrical & Computer Engineering from Concordia University, and a Ph.D. degree in Information Systems from University of Southern Queensland, Australia. He has more than 17 years of professional experience in IT/IS, mathematical modeling, and financial analysis. Dr. Valverde is a member of the Society of Management Accountants, Canadian Operational Research Society, Institute of Internal Controls, Forensic CPA society, Professional Engineers of Ontario and the Association for Operations Management. His main research interests include Supply Chain Systems, Risk Management, E-business, Information Security and Auditing, Accounting and Financial Information Systems, Fraud Detection and Reengineering. Dr. Valverde is a registered professional engineer and accountant in Canada.