An Architecture-oriented Model for Requirements Analysis and Specification

Pingwen Yu
Department of Information Management,Shu-Zen College of Medicine and Management

Fushiau Li
Research and Development Department, SUN NET Technology Corporation Ltd

William S. Chao
Taiwan Chapter of Association of Enterprise Architects

Abstract

For requirements being able to convey the essential properties of a system, an ideal model for requirements analysis and specification should be based on a set of interacting components forming an integrated whole of that system's structure and behavior views which sustains the essential properties of a system. Current models for requirements analysis and specification such as data-oriented, control-oriented, function-oriented, and object-oriented, more or less, fail to grasp some essential properties of the system. In this paper, we present an architecture-oriented model for requirements analysis and specification named SBC architecture which helps analysts specify the requirements based on a set of interacting components forming an integrated whole of the system's structure and behavior views. SBC architecture is a perfect model for requirements analysis and specification because of its being able to convey the essential properties of any system.

Keywords: architecture-oriented model, requirements analysis and specification, SBC architecture

1. Introduction

In the simplest terms, the system development process consists of three activities: the analysis and specification of a requirement, the design of a solution, and the implementation of the solution. Requirements analysis and specification employ a model to analyze and specify various stakeholders' perception of their need and understanding of the solution. That is, a model for requirements analysis and specification is besought to convey the essential properties that the system must satisfy.

A system is complex that it consists of multiple views such as structure view, behavior view, input data view, output data view, business view, application view, technology view, function view, timing view, concurrency view, control view, management view, engineering view, etc. Among the above multiple views, the structure and behavior views are perceived as the two prominent ones. The structure view focuses on the system structure which is described by components and their composition while the behavior view concentrates on the system behavior which involves interactions among external environment's actors and components. Accordingly, we define a system as a set of interacting components forming an integrated whole of that system's multiple views [Bert69]. Components are sometimes labeled as parts, entities, objects, and structure elements [Chao09, Chao11]. Since structure and behavior views are the two most distinguished ones among multiple views, integrating the structure and behavior views becomes the most appropriate method for integrating multiple views of a system as shown in Figure 1. We thereat redefine a system as a set of interacting components forming an integrated whole of the system's structure and behavior views. Based on this definition of a system, we conclude that an integrated structure and behavior views sustains the essential properties of a system.

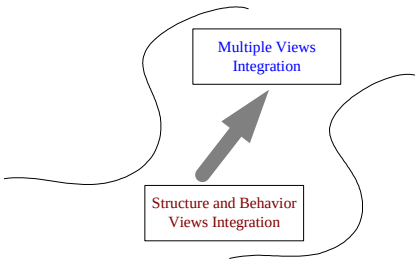

Figure 1 Structure and Behavior Views Integration Facilitates Multiple Views Integration

For requirements being able to convey the essential properties of a system, an ideal model for requirements analysis and specification should be based on a set of interacting components forming an integrated whole of that system's structure and behavior views. The purpose of this paper is to explore this principle in depth. The whole paper is organized as follows. Section 1 is the introduction. Current models such as data-oriented, control-oriented, function-oriented, and object-oriented for requirements analysis and specification failing to specify some essential properties of the system are discussed in Section 2. Section 3 examines in detail the SBC architecture

which integrates the structure and behavior views of a system. Advantages of using SBC architecture as a model for requirements analysis and specification are delineated in Section 4. Section 5 is a summary.

2. Deficiencies of Current Requirement Models

Current models for requirements analysis and specification are divided into the following categories: data-oriented, control-oriented, function-oriented, and object-oriented as shown in Figure 2. Each of these models, more or less, fails to grasp some essential properties of the system.

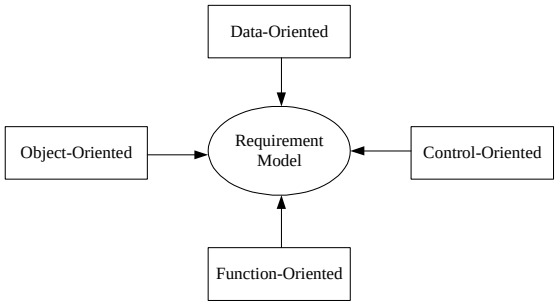

Figure 2 Current Models for Requirements Analysis and Specification

Data-oriented models for requirements analysis and specification stress the system state as a data structure. Jackson System Development (JSD) [Came89] and Entity Relationship Modeling (ERM) [Chen76] are primarily data-oriented. Data-oriented models concentrate only on data and completely ignore an integrated structure and behavior views which seizes the essential properties of a system. Therefore, data-oriented models should not and will never become an ideal requirement model.

Control-oriented models for requirements analysis and specification emphasize synchronization, deadlock, exclusion, concurrency, and process activation of a system. Petri Net [Reis92] and Flowcharting [Bash86] are primarily control-oriented. Control-oriented models concentrate only on the control view and completely neglect an integrated structure and behavior views which grasps the essential properties of a system. Just like data-oriented, data-oriented models should not and will never become an ideal requirement model.

Function-oriented models for requirements analysis and specification take the primary view of the way a system transforms input data into output data. Each transformation from input data into output data demonstrates a function of the system. A system may contain many such kinds of functions which represent the function view of the system. Classical Structured Analysis (SA) [DeMa79] fits into the category of function-oriented models, as do Structured Analysis and Design Technique (SADT) [Marc88] and Structured Systems Analysis and Design Method (SSADM) [Ashw90]. Function-oriented models concentrate only on the function view and completely neglect an integrated structure and behavior views which grabs the essential properties of a system. Like data-oriented and control-oriented, function-oriented models should not and will never become an ideal requirement model.

Object-oriented models for requirements analysis and specification describe the system as classes of objects and their behaviors. Object-oriented Analysis (OOA) [Booc07], fitting into the category of object-oriented models, looks at the problem domain, with the aim of producing a conceptual model of the information that exists in the area being analyzed. The result of object-oriented analysis is a description of what the system is behaviorally required to do, in the form of a conceptual model. That will typically be presented as a set of use cases and a number of interaction diagrams. Object-oriented models stress both the structure view and the behavior view, but not an integrated structure and behavior views. Because object-oriented models do not emphasize an integrated structure and behavior views which grabs the essential properties of a system, and therefore object-oriented models should not and will never become an ideal requirement model.

3. Requirement Model of SBC Architecture

SBC (stands for structure-behavior coalescence) architecture is an architecture-oriented model for requirements analysis and specification which integrates the structure and behavior views of a system. SBC architecture consists of three fundamental diagrams for requirements analysis and specification. These diagrams are: a) framework diagram, b) component operation diagram, and c) interaction flow diagram.

(a) Requirement analysts use the framework diagram (FD) to specify the multi-layer (also referred to as multi-tier) decomposition and composition of a system. FD is the first fundamental diagram to achieve structure-behavior coalescence. Only components will appear in the framework diagram. As an example, Figure 3 shows the FD of an energy management system *EMS*. In the figure, layer *Layer_3* contains the components *A* and *B*; layer *Layer_2* contains the components *C* and *D*; layer *Layer_1* contains the components *E* and *F*.

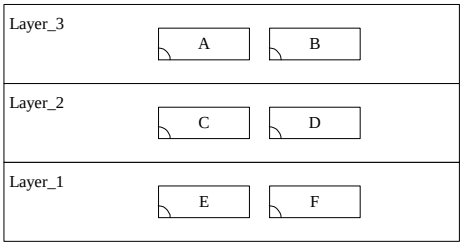

Figure 3 FD of *EMS*

(b) Requirement analysts use component operation diagram (COD) to specify all components' operations of a system. COD is the second fundamental diagram to achieve structure-behavior coalescence. An operation provided by each component represents a method of that component [Booc07]. A component should not exist in a system if it does not own any operation. As an example, figure 4 shows that component *A* has two operations: *op_01* and *op_02*.

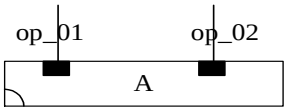

Figure 4 Two Operations of Component *A*

An operation formula is utilized to fully describe and represent an operation. An operation formula includes a) operation name, b) input parameters, and c) output parameters as shown in Figure 5.

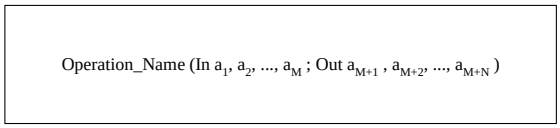

Figure 5 Operation Formula

Operation name is the name of this operation. In a system, every operation name should be unique. Duplicate operation names shall not be allowed in any system.

An operation may have several input and output parameters. The input and output parameters, gathered from all operations, represent the input data and output data views of a system [Date03, Elma10]. As shown in Figure 6, component *B* owns the operation *op_03* which has no input/output parameter; component *B* also owns the operation *op_04* which has one input parameters *CustomerName* (with arrow direction pointing to the component) and one output parameter Customer_Report (with arrow direction opposite to the component).

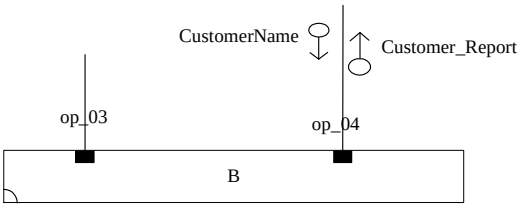

Figure 6 Input/Output Parameters of Component *B*

Data formats of input and output parameters can be described by data type specifications. There are two groups of data types: primitive and composite. Figure 7 shows primitive data type specifications of the input parameters *Quantity* occurring in the operation formula *op_01(In Quantity)*, *CustomerName* occurring in the operation formula *op_04(In CustomerName; Out Customer_Report)*, *UnitPrice* occurring in the operation formula *op_09(In UnitPrice)*.

Parameter	Quantity	CustomerName	UnitPrice
Data Type	Integer	Text	Real
Instances	300	General Electric, IBM, HP	100,000.00

Figure 7 Primitive Data Type Specifications

Figure 8 shows composite data type specifications of the output parameter *Customer_Report* occurring in the operation formula *op_04(In CustomerName; Out Customer_Report)*.

Parameter	*Customer_Report*
Data Type	TABLE of Date : Text CustomerName : Text EnergyType : Text Quantity : Integer UnitPrice : Real Total : Real End TABLE ;
Instances	Date : 20120517 CustomerName : General Electric EnergyType | Quantity | UnitPrice A00001 | 400 | 100,000.00 A00002 | 300 | 200,000.00 Total : 100,000,000.00

Figure 8 Composite Data Type Specifications

To specify a system, COD is used to display all components' operations. Figure 9 shows the COD of an energy management system *EMS*. In the figure, component A has two operations: *op_01* and *op_02*; component *B* has two operations: *op_03* and *op_04*; component *C* has one operation: *op_05*; component D has three operations: *op_06*, *op_07*, and *op_08*; component *E* has one operations: *op_09*; component *F* has one operations: *op_10*.

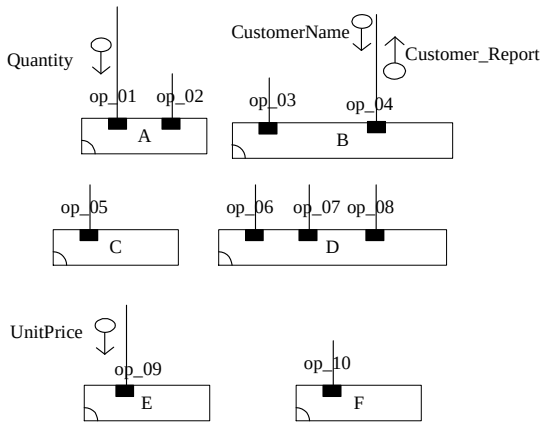

Figure 9 COD of *EMS*

(c) Requirement analysts use interaction flow diagram (IFD) to specify all individual behavior of a system. IFD is the third fundamental diagram to achieve structure-behavior coalescence. In a system, if the components, and among them and the external environment's actors to interact, these interactions will lead to the system behavior. That is, "interaction" plays an important factor in integrating structures with behaviors for a system. In the example of energy management system *EMS*, an actor interacting with six components shall describe the overall system behavior. As shown in Figure 10, interactions among the actor *Customer* and the components *A*, *C*, and *E* generate the individual behavior *ems_1*; interactions among the actor *Customer* and the components *A* and *D* generate the individual behavior *ems_2*; interactions among the actor *Customer* and the components *B*, *D*, and *F* generate the individual behavior *ems_3*; interaction among the actor *Customer* and the component *B* and *D* generates the individual behavior *ems_4*.

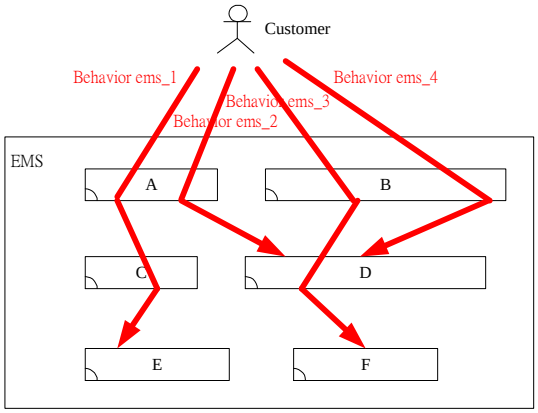

Figure 10 Four Individual Behaviors of *EMS*

The overall behavior of a system is the collection of all its individual behaviors. All individual behaviors are mutually independent of each other. They tend to be executed concurrently [Hoar85, Miln89, Miln99]. Each individual behavior represents an execution path. We use interaction flow diagram (IFD) to demonstrate this individual behavior. Figure 11 demonstrates the IFD of the behavior *ems_1*. The X-axis direction is from the left side to right side and the Y-axis direction is from the above to the below. Inside an IFD, there are four elements: a) external environment's actor, b) components, c) interactions, and d) input/output parameters. Participants of the interaction, such as the external environment's actor and each component, are laid aside along the X-axis direction on the top of the diagram. The external environment's actor which initiates the sequential interactions is always placed on the most left side of the X-axis. Then, interactions among the external environment's actor and components successively in turn decorate along the Y-axis direction. The first interaction is placed on the top of the Y-axis position. The last interaction is placed on the bottom of the Y-axis position. Each interaction may carry several input and/or output parameters.

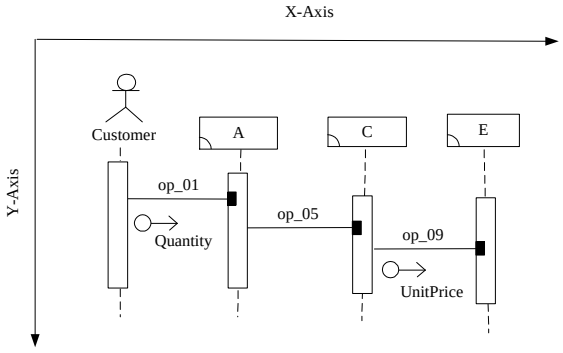

Figure 11 IFD of the Behavior *ems_1*

In Figure 11, *Customer* is an external environment's actor. *A*, *C*, and *E* are components. *op_01* is an operation, carrying the input parameter *Quantity*, which is provided by the component *A*. *op_05* is an operation which is provided by the component *C*. *op_09* is an operation, carrying the input parameter *UnitPrice*, which is provided by the component *E*.

The execution path of Figure 11 is as follows. First, actor *Customer* interacts with the component *A* through the operation call interaction *op_01*, carrying the input parameter *Quantity*. Next, component *A* interacts with the component *C* through the operation call interaction *op_05*. Finally, component *C* interacts with the component *E* through the operation call interaction *op_09*, carrying the input parameter *UnitPrice*.

For each interaction, the solid line stands for operation call while the dashed line stands for operation return. The operation call and operation return interactions, if using the same operation name,

belong to the identical operation. Figure 12 exhibits two interactions (operation call interaction and operation return interaction) having the identical operation "*op_04*."

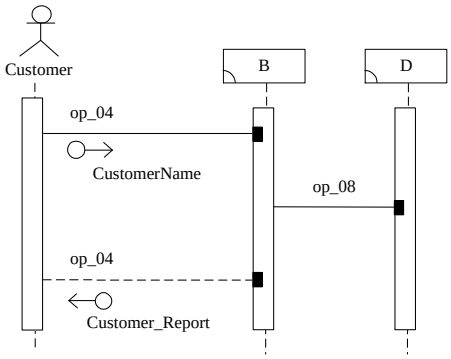

Figure 12 Two Interactions Have the Identical Operation

The execution path of Figure 12 is as follows. First, actor *Customer* interacts with the component *B* through the operation call interaction *op_04*, carrying the input parameter *CustomerName*. Next, component *B* interacts with the component *D* through the operation call interaction *op_08*. Finally, actor *Customer* interacts with the component *B* through the operation return interaction *op_04*, carrying the output parameter *Customer_Report*.

The overall *EMS*'s behavior includes four behaviors: *ems_1*, *ems_2*, *ems_3*, and *ems_4*. Each of them is described by an individual IFD. Figure 13 shows the IFD of the behavior *ems_1*. First, actor *Customer* interacts with the component *A* through the operation call interaction *op_01*, carrying the input parameter *Quantity*. Next, component *A* interacts with the component *C* through the operation call interaction *op_05*. Finally, component *C* interacts with the component *E* through the operation call interaction *op_09*, carrying the input parameter *UnitPrice*.

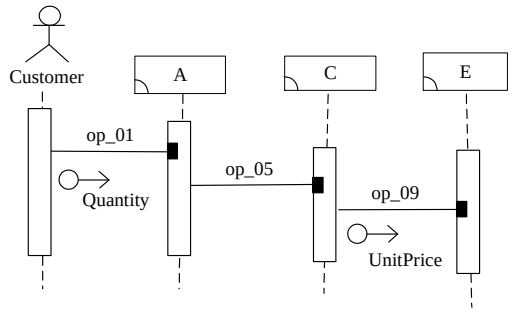

Figure 13 IFD of the Behavior *ems_1*

Figure 14 shows the IFD of the behavior *ems_2*. First, actor *Customer* interacts with the component *A* through the operation call interaction *op_02*. Finally, component *A* interacts with the component *D* through the operation call interaction *op_06*.

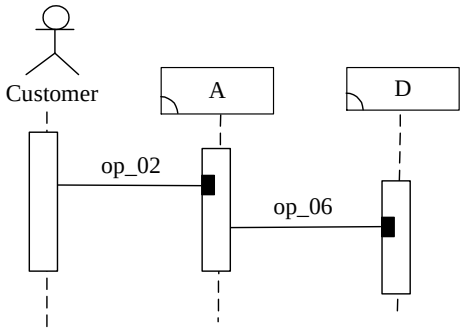

Figure 14 IFD of the Behavior *ems_2*

Figure 15 shows the IFD of the behavior *ems_3*. First, actor *Customer* interacts with the component *B* through the operation call interaction *op_03*. Next, component *B* interacts with the component *D* through the operation call interaction *op_07*. Finally, component *D* interacts with the component *F* through the operation call interaction *op_10*.

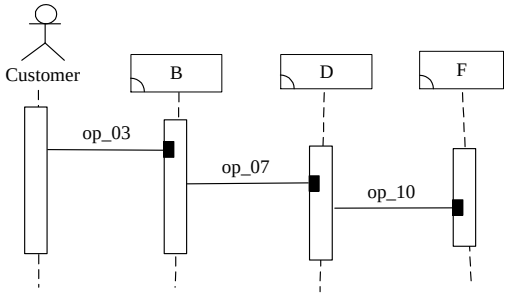

Figure 15 IFD of the Behavior *ems_3*

Figure 16 shows the IFD of the behavior *ems_4*. First, actor *Customer* interacts with the component *B* through the operation call interaction *op_04*, carrying the input parameter *CustomerName*. Next, component *B* interacts with the component *D* through the operation call interaction *op_08*. Finally, actor *Customer* interacts with the component *B* through the operation return interaction *op_04*, carrying the output parameter *Customer_Report*.

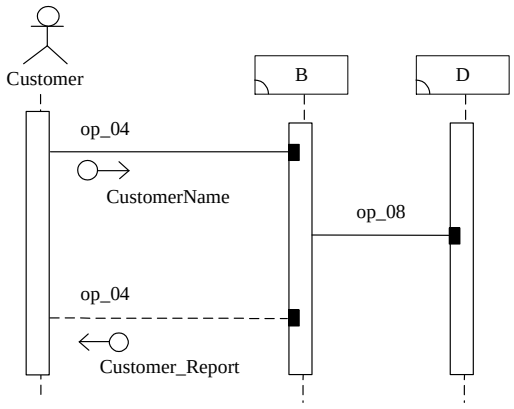

Figure 16 IFD of the Behavior *ems_4*

4. Adequacies of SBC Architecture

There are many advantages of using SBC architecture as a model for requirements analysis and specification. Let us discuss them now. Firstly, SBC architecture furnishes the framework diagram to help requirement analysts specify the multi-layer decomposition and composition of a system. Early in the system development process, analysts begin to consider what components should be defined in the hierarchy. Framework diagram is exactly the perfect tool for analysts to write down the requirement specification once the hierarchy of components is defined.

Secondly, SBC architecture furnishes the component operation diagram to help requirement analysts specify all components' operations of a system. An operation is a method which defines the mapping between inputs and outputs. A system may contain many operations. Each operation is provided by a component. A component owns at least an operation. Component operation diagram is a perfect tool for analysts to write down the specification once all operations are defined.

Thirdly, SBC architecture uses the interaction flow diagram to help analysts specify all individual behavior of a system. The overall behavior of a system is the collection of all its individual behaviors. All individual behaviors are mutually independent of each other. They tend to be executed concurrently. Each individual behavior represents an execution path. Interaction flow diagram is a perfect tool for analysts to write down the specification once the overall behavior of a system is defined.

Fourthly, because SBC architecture integrates the structure and behavior views so it is able to integrate the multiple views of a system. Therefore, as a model for requirements analysis and specification, SBC architecture is fully capable of seizing the essential properties of a system.

5. Summary of Paper

If a model for requirements analysis and specification is able to integrate the structure and behavior views, then it is able to integrate the multiple views of a system. If a model for requirements analysis and specification is able to integrate the multiple views, then it is able to grasp the essential properties of a system.

Current models such as data-oriented, control-oriented, function-oriented, and object-oriented for requirements analysis and specification failing to specify an integrated structure and behavior views hence they are not able to integrate the multiple views of a system. We conclude that data-oriented, control-oriented, function-oriented, and object-oriented for requirements analysis and specification should not and will never become an ideal requirement model.

The characteristics of SBC architecture lie in its integrating the structure and behavior views hence it is able to integrate the multiple views of a system. Therefore, as a model for requirements analysis and specification, SBC architecture is fully capable of grasping the essential properties of a system.

References

[1] Ashworth, C., SSADM : A Practical Approach, McGraw-Hill Book Company (UK) Ltd., 1st Edition, 1990.

[2] Bashe, C., IBM's Early Computers, The MIT Press, 1986.

[3] Bertalanffy, L. V., General System Theory: Foundations, Development, Applications, Revised Edition, George Braziller, Inc., 1969.

[4] Booch, G., Object-oriented Analysis and Design with Applications, 3rd Edition, Addison-Wesley, 2007.

[5] [Cameron, John R., The Jackson Approach to Software Development, IEEE Computer Society Press, Silver Spring, 1989.

[6] Chao, W. S. et al., System Analysis and Design: SBC Software Architecture in Practice, LAP Lambert Academic Publishing, 2009.

[7] Chao, W. S., Software Architecture: SBC Architecture at Work, National Sun Yat-sen University Press, 2011.

[8] Chen, P. et al., "*The Entity-Relationship Model - Toward a Unified View of Data*", ACM Transactions on Database Systems 1 (1), pp. 9–36, 1976.

[9] DeMarco, T., Structured Analysis and System Specification, Prentice Hall, 1979.

[10] Date, C. J., An Introduction to Database Systems, 8th Edition, Addison Wesley, 2003.

[11] Elmasri, R., Fundamentals of Database Systems, 6th Edition, Addison Wesley, 2010.

[12] Hoare, C. A. R., Communicating Sequential Processes, Prentice-Hall, 1985.

[13] Marca, D. A. et al., SADT: Structured Analysis and Design Technique, McGraw-Hill, 1988.

[14] Milner, R., Communication and Concurrency, Prentice-Hall, 1989.