Journal of Asian Scientific Research

journal homepage: http://aessweb.com/journal-detail.php?id=5003

BEHAVIOR MODELING, LANGUAGES AND DIAGRAMS IN COMPONENT BASED SOFTWARE DEVELOPMENT

Rihab Eltayeb Ahmed[1]
Nael Salman[2]

ABSTRACT

Considering that the component is the fundamental building block of software systems in CBSD, it is important to investigate methods for implementing its functionality or behavior. The functionality in components is implemented as code in a programming language, as a result the component become dependent on that specific programming language. Behavior modeling can provide a solution to that, increasing the component reusability. Reusability of a component will be higher when its behavior is described in a high level of abstraction (a model) that can be translated in different programming languages. This study investigated and analyzed various modeling diagrams, languages and tools used to specify components behavior.

Key Words: Component, Behavior modelling, COSEML, CBSD, Modeling languages/ Tools.

INTRODUCTION

A model is a simplification of reality (G. Booch, J. R. & I. Jacobson 2000) Models allow defining and breaking large applications and systems into smaller pieces. Building models provides a better understanding of the system which is very important in the development phase when we need to know what the system is made of(structure, relations, organization) and what it accomplishes (functionality) and how it behaves. After a concrete model of the system is created, it becomes as a valuable asset, well documented (especially when modeled by a standard formal representation) and a strong candidate for reuse.

Component Based Software Development (CBSD) approach is based on the reusability principle where systems are built by integrating reusable independent, tested and trusted software

[1] College of Computer Science and Information Technology, Sudan University of Sciences & Technology, Khartoum, Sudan

[2] College of Engineering and Technology Palestine Technical University-Kadoorie, Tulkarm, Palestine

components other than the development from the scratch. Accordingly modeling becomes more important than code writing where components are viewed as an architectural abstraction.

A component is defined in (BACHMANN F., E. A. 2000) as an opaque implementation of functionality, subject to third-party composition and conformant with a component model. Taking a closer look at its description as a concrete implementation of functionality is important. Functionality or behavior describes a set of possible executions; an execution is the performance of an algorithm according to a set of rules (G. Booch, J. R. & I. Jacobson 2000). Considering that the component is the fundamental building block of software systems in CBSD, it is important to investigate methods for implementing its functionality or behavior. The functionality in components is implemented as code in a programming language, as a result the component become dependent on that specific programming language. Behavior modeling can provide a solution to that, increasing the component reusability. Reusability of a component will be higher when its behavior is described in a high level of abstraction (a model) that can be translated in different programming languages and thus implemented in different component platforms.

This paper attempts to analyze and summarize behavior modeling languages and diagrams used to specify functionality or behavior of components in CBSD. Section 2 introduces various behavior diagrams types in general and their use to describe the functionality. Section 3 summarizes and discusses languages/tools and their support of behavior diagrams. In section 4 we briefly describe the UML component diagram and it usage, section 5 links Model Driven Architecture with component technology, and in section 6 we conclude the paper.

BEHAVIOR DIAGRAMS IN UML

Behavior is a direct consequence of the actions of at least one object; it describes how the states of the participating objects change over time (OMG, 2007). Behavior specifications can be used to define, describe or illustrate the behavior of an object such as the allowed and forbidden behaviors. A variety of mechanisms to specify behaviors are supported in UML such as automata (state machine), Petri-net like graphs (activity), informal description (use case) and sequence of events (interaction). These different behavior specification mechanisms differ in their expressive power and domain of applicability so the choice of one of them depends on convenience and purpose.

A state machine is a behavior that specifies the sequences of states an object goes through or an order of the invocation of its operations during its lifetime (OMG, 2007). A state is a condition or situation in the object lifetime. Modeling the lifetime of an object using a state machine would depict states and transitions between them and allow focusing on the event ordered behavior of an object, which is useful in modeling reactive systems.

An activity ultimately results in some action -that is made up of executable computations- that change the system state or return a value (OMG, 2007). Activity diagram in UML is used to model a workflow that represents the flow of work and objects through the business. It can be attached to classes, interfaces, components and other elements. The most common element to attach with is an operation to reflect its flow control or flowchart.

A use case is description of a set of sequences of actions that a system performs to produce results of value to an actor (role a user acts) (OMG, 2007). Use cases are applied to capture the intended behavior without specifying how that behavior is implemented. In UML use case diagram show the use cases, actors and their relationships. The diagram is used to specify system use cases, the behavior scenario is described in text and can be graphically represented and enriched by collaboration diagrams.

An interaction is a behavior that comprises a set of messages exchanged among a set of objects within a context to accomplish a purpose(OMG, 2007). When modeling an interaction, one essentially is building a storyboard of actions that take place among a set of objects. Interactions can be modeled in two ways: by the time ordering of its messages and by the structural objects that send and receive messages. In UML the first way of representation is called a sequence diagram, the second is a collaboration diagram.

MODELING TOOLS/LANGUAGES AND THEIR SUPPORT OF BEHAVIOR DIAGRAMS IN CBSD

Remes

REsource Model for Embedded Systems (REMES) is a language for resource modeling and analysis of embedded systems intended to describe the resource-wise behavior of interacting embedded components (C. Seceleanu, A. V., and P. Pettersson 2009). It focuses on component-based behavioral modeling. Resources in REMES include memory, ports, CPU and busses. REMES has a graphical behavioral language with timed and hybrid automata, and state charts in order to express resource usage in a system. It also has notions of continuous variables, flows, and progress constraints (invariants), which fit modeling timed behaviors in embedded systems. The internal behavior of a component is described by a mode. There are two types of modes, an atomic mode which does not contain sub-modes, and a composite mode that contains a number of sub-modes. Data is transferred between modes through global variables. A mode can execute either a discrete step, by a discrete action, or a continuous step, via a delay action.

REMES can be used to analyze resource consumption of an embedded system where the resource availability is limited due to environment constraints. It can aid the decision of replacing a component that requires more memory resources with another having the same functionality but with less resources demand (C. Seceleanu, A. V., and P. Pettersson 2009).

In another study (A. Cauˇsevic , A. V. 2009) that discusses component Based Systems (CBS) and Service-Oriented Systems (SOS), the authors showed how REMES can be used in the context of SOS. There idea was based on the fact that since both CBS and SOS are similar to each other in many ways, it could be beneficial to use common methods of modeling systems architecture and behavior. The goals of both CBS and SOS are to support reusability and to produce error free, efficient software systems. Beside that components and services can coexist in a large system.

In service modeling using REMES the service behavior description (internal workflow process) can be modeled using modes and sub-modes, service providers and quality of service prediction attributes related to resource usage in SOS are supported in REMES (A. Cauˇsevic , A. V. 2009). They mentioned additional modeling capabilities that can be added to the current version of their language to be more suitable for SOS such as the modeling of service failure and service discovery.

In a recent research work (Raluca Marinescu, E. P. E. 2011) a tool based support for REMES was developed in which REMES was extended to present a design framework for behavior modeling of SOS along with a graphical user interface using NetBeans API.

Abctool

ABCTool is visual software architecture (SA) tool that enables the development, deployment, maintenance and evolvement of component based software systems. It is the tool of ABC which is an architecture centric CBSE approach that contains four phases: design of SA, Architecture based component composition, Architecture based application deployment and Architecture based maintenance and evolution (Hong Mei , G. H. 2008).

The design view of ABCTool provides the type diagram for defining components and connectors to be reused in the system, it also provide the configuration diagram and the process diagram. The implementation view is for developers to implement the designed architecture specified by the design view diagrams. The ABCTool will select the suitable components and connectors based on the type diagram form an asset library. The selected assets are qualified and adapted as specified by the configuration and process diagrams. The components and connectors that are not available in the library should be developed by developers where a design model using UML or other would be generated automatically. Finally the selected and developed components are assembled as deliverable software packages (specific for J2EE WS or CORBA for this version). The tool also contains the deployment view that is resulted from transforming the implantation view. Both component implementation and deployment descriptors are generated as deployable packages to suit the target execution environment(Hong Mei , G. H. 2008).

Beside the mentioned views the tool provides many other views among them is the runtime view that help administrators maintain and evolve the system by monitoring and applying changes at runtime(Hong Mei , G. H. 2008).

The tool helps in locating and integration components in the design and implementation of CB systems but the developer intervention exists for the semantic adaptation of components and also to develop the unavailable assets. There is a big gap between the design view and the implementation view as noted by the researchers. The tool also support the basic idea of component based software development in which the system is not developed from scratch but assembling components together.

Coseml

Component Oriented Software Engineering Modeling Language (COSEML) is a graphical modeling language that supports building systems by integration not by code and following the concept representation described by the process model COSE. In COSE approach the system specifications are transformed to a set of components and connectors that connect the components in order to produce the target system. The approach starts by a structural decomposition to introduce the system building blocks, this process continues to lower levels until it arrives to a level where the module corresponds to a component in which the desired capabilities can be achieved (Dogru, A., Altintas, I. 2000).

The graphical modeling elements in COSEML are packages and its internal symbols: data, function and control. The behavior is described within the component symbol methods and events.

In many aspects COSEML is similar to UML. For example the symbols it used; the function symbol is similar to the UML use case. It uses the component as an object with properties and methods that describe its behavior. It also utilizes the UML class diagrams relations symbols, the diamond for composition and the triangle for the inheritance. The main distinguishing feature of COSEML is using the top-down approach by starting with structural decomposition and select the required and existing implemented component codes in a temporary bottom-up approach) is different than that is used in UML (top-down starting from requirement modeling to automatic code generation). they share some commonalities to a degree in which one could think of combining them together.

In COSEML only the static modeling is supported which is not enough to describe the system. Connectors are used to show the relations, but messages and interactions between components cannot be expressed. A study (Tuncel, M. B. 2006) was performed to extend COSEML to support behavior modeling (dynamic modeling) by adding the collaboration diagram. The collaboration diagram was chosen to be modeled because it reflects the structural organization of model elements and their roles. It is suitable for COSE approach since it is structural too. Beside that it enrich the models by increasing its expressiveness capabilities showing more details than before. Moreover since collaboration diagram models the message calls between components, automated testing to eliminate problems in the early stages can be applied to COSEML models. The study (Tuncel, M. B. 2006) introduces two types of collaboration diagrams, abstract and runtime. Abstract

collaboration diagram describes the externally visible actions and their sequences with no implementation details. It shows the high level system behavior. The runtime collaboration diagram describes the sequence of method calls, event subscriptions and data flow between the interfaces of components. It is used to integrate components through their interfaces. Both abstract and runtime collaboration diagrams were implemented as part of the COSECASE tool which is a tool for COSEML language.

UML COMPONENT DIAGRAM

The components package in UML addresses the area of component based development and component based systems structuring where a component is a modular unit with well defined interfaces. The components package supports both logical components (business and process components) and physical components (EJB, CORBA, .Net etc). (OMG, 2007)

UML provides the Component diagram as one of the structure diagrams that are used to describe and show the static structure of objects in a system. Structure diagram such as class, component and deployment diagrams may include abstract, real world and implementation concepts as their elements. Behavioral diagram such as the activity, interaction, use case and state machine diagrams are used to show the details of the dynamic behavior that were not shown in the structural diagrams. Components may be grouped together to form a coherent group of elements and to formalize the concept of components as a building block. (OMG, 2007).

The Basic Component in UML is a subtype of Class and it has a number of provided and required interfaces for wiring components together as shown in Figure 1. Optionally a behavior may be attached to an interface, port and to the component itself in terms of use case or interaction specification for the component external view as shown in Figure 2. The internal view of the component reflects its private properties and realizations. More detailed behavior specifications such as activity and interactions can be used to describe the internal behavior, and the mapping between the internal and external views. (OMG, 2007)

Figure- 1. A UML Component with two provided and one required interfaces.

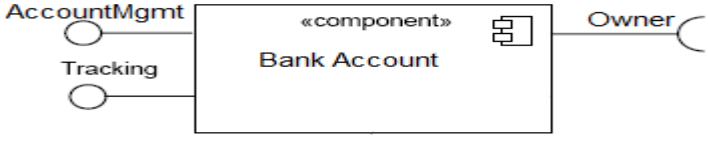
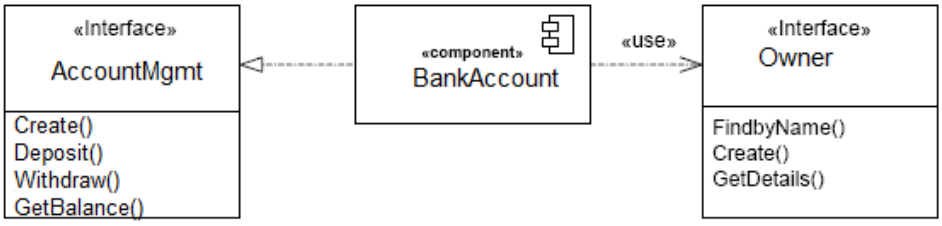


Figure- 2. Explicit modeling of required and provided interfaces, showing details of operations.



UML component diagram can be considered rich enough because it has the ability to access and link with other static and behavioral diagrams available within UML itself. Also the ability to construct a larger component from other components and showing the flows and interactions between them internally and externally. In this way UML component diagram would be suitable for both component development and selection processes. UML is a formal modeling language, adopted as a standard specification for object modeling of systems. Being formal means to model with well defined syntax and semantic and with no ambiguity. Being a standard allows for sharing, reusability and integration. In CBSD, reusability and integration are corner stone's.

A study (Ferdelja, I. 2009) analyzed various modeling languages and specifically investigated the ways to apply them to Progress Component model (ProCom). ProCom is a component model developed as a part of a project that aims at providing theories, engineering methods and tools for component based development of embedded systems(Ferdelja, I. 2009). The major concern of the study was the adaptation of Construction and Composition Language (CCL) and Executable UML (xUML) to ProCom.

CCL is a construction and composition language for component based systems, moreover it is an essential part of Prediction Enabled Component Technology (PECT) approach. A State chart language based on UML state charts- showing states and transitions between them- was used to define component behavior in CCL along with an executable action language. Modeling in xUML serves for creating highly abstract system models without technical details of hardware and software platforms, more specifically creation of Platform Independent Models (PIM) in Model Driven Architecture (MDA) terminology. A model compiler can transform the executable UML model (a higher level than programming languages) into an implementation. (Ferdelja, I. 2009).

CCL was proven in the study as more convenient for modeling ProCom component behavior than xUML because of the reasonable compatibilities between CCL and component model in ProCom. Further solutions would be by adapting and utilizing action languages used to describe behavior of classes such as EP, ASL, JUMBALA and Scrall. The study concluded that the adaptation would lead to a language similar to CCL.

SUMMARY

In this study we investigated and analyzed various modeling diagrams, languages and tool used to specify components behavior. Diagrams are a visual means for specifying behavior with elements and symbols, it provides an easier way to show, express and communicate the modeled behavior. Languages provide a base that diagrams and tools can be build on. Tool support is important as they include more than one capability besides modeling.

A variety of behavior specification mechanisms are supported in UML including state machine, activity, use case and interaction diagrams. These different behavior specification mechanisms differ in their expressive power and domain of applicability. They provide a different angle of view to the behavior being modeled. The choice of one or more of them depends on convenience and purpose. UML as a formal modeling language can be used in CBSD in the development phase of the component itself. The above mentioned behavioral diagrams in addition to the component diagram which support structuring component systems would result in a high level of abstraction when used, although UML is a general purpose modeling language not built specifically for CBSD.

(REMES) is a language not only for resource modeling but also for resource analysis of embedded systems that include memory, ports, CPU and busses. The analysis help to decide when to replace a component with a less resource consuming one. It had been extended to cover domains other than embedded systems. It is a good example of reusability among languages support. An adaptation of various modeling languages to embedded systems is realized in ProCom component model.

ABC is an approach to CBSE in which the phases are defined, it is enriched with an architecture description language, called ABC/ADL. The ABCTool, the core tool of ABC is a visual architecture (SA) tool that enables the development, deployment, maintenance and evolvement of component based software systems. The strength of ABCTool is that it is based on a defined approach that covers the lifetime of a component based system. It also increases the value of modeling and system architecture building in a systematic and automated manner.

COSEML is similar to ABCTool because it is based on an approach (COSE) that defines the process model. In COSE the problem of system development is reduced to selecting and assembling components, assuming that the most functions are already implemented in components. COSEML does not address how to deploy, maintain and evolve the component based system, where ABCTool does. COSEML is similar to UML in the symbols it uses, where ABC approach is similar to UML in providing a set of diagrams for handling different concerns. Neither COSEML nor ABCTool and other tools are formal ,standard or widely adopted as UML.

CONCLUSION

Behavior modeling in CBSD is an essential activity, because modeling introduces higher levels of abstraction that in turn allow for reusability between different languages, frameworks and execution environments. Modeling languages, tools and diagrams were investigated and analyzed. The research works analyzed covered languages used in component behavior modeling, diagrams used to represent component behavior in those languages and toolsets. Part of them is extended to provide more adequate support of behavior modeling. Behavior modeling is promising and the efforts and studies are adapting and reusing ideas from other modeling domains. Still a standard, widely adopted and general purpose component behavior modeling languages, diagrams and tools are to be waited for.

REFERENCES

G. Booch, J. R., I. Jacobson (2000) The Unified Modeling Language User Guide, Addison Wesley.

Bachmann F., e. a (2000) Technical concepts of Component-Based Software Engineering, CMU/SEI.

OMG (2007) OMG Unified Modeling Language (OMG UML),Superstructure.

C. Seceleanu, A. V., and P. Pettersson (2009) Remes:A resource model for embedded systems, 14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2009). IEEE Computer Society.

A. Cauˇsevic , A. V (2009) Towards a unified behavioral model for component based and service-oriented systems. 33rd Annual IEEE International Computer Software and Applications Conference, IEEE.

Raluca Marinescu, E. P. E (2011) A Design Framework for Service-Oriented Systems. Sweden, M¨alardalen University. Master's Thesis in Computer Science.

Hong Mei , G. H (2008) ABCTool: A Tool for Architecture Centric Engineering of Component based Systems. ICSE Companion '08 Companion of the 30th international conference on Software engineering.

Dogru, A., Altintas, I (2000) Modeling Language for Component-Oriented Software Engineering: COSEML. The Fifth World Conference on Integrated Design and Process Technology, Dallas, Texas.

Tuncel, M. B (2006) Using Collaboration Diagrams in Component Oriented Modeling, Middle East Technical University. Master of Science in Computer Engineering

Ferdelja, I (2009) Component Behavior Modeling, Middle East Technical University. Master of Science in Computer Engineering.