Z –Specification of Component Based Software

Prasenjit Banerjee¹ and Anirban Sarkar²

¹Midnapore College, Midnapore (W), West Bengal, India
²National Institute of Technology, Durgapur, West Bengal, India
¹prasenjitmca7@gmail.com, ²sarkar.anirban@gmail.com

Abstract

The paper focuses on a Z-semantic based conceptual model of component, called Z- Formal Specification of Component Model (ZFSCM), to conceptualize the different facts of component model in Component Based Software Engineering (CBSE). This model defines a set of components and their contracts or relationships using Z- notation. It is accompanied with different type’s services, classes, components and interfaces. This approach facilitates modeling of components using different schemas in software system. The ZFSCM has being checked validation using a type checker of Z notation (ZTC). The main focus of this paper is the formal specification of the components. The paper has also defined the relationships between the components and composition among them. Moreover, the paper also provides comparative study of several similar kinds of proposals for component models in Component Based Software Engineering (CBSE).

Keywords: Component, Component Based Software Engineering (CBSE), Formal Specification, Z-notation

1. Introduction

In present time, wide and enormous changes have occurred in software engineering. In the early time, the programmers wrote their program codes line by line which had limited functions and limited facility to interact with each other. But today, software development has changed its way by providing Object-Oriented Software Modeling (OOSM) and Component Based Software Modeling (CBSM) techniques. In Component Based Software Engineering, a component represents a software element or a modular unit of a system with the condition that it may be used by other software elements and it is not tied to any fixed set of other software elements i.e., its clients. A component defines its functions or the methods by the interfaces. These interfaces can be characterized with two types of interfaces – provided interfaces and required interfaces. One component then can be defined by the other components. The components may vary and can be reuse as a part of the other components. According to Ivica Crnkovic [4], a software component is a software building block that conforms to a component model. A component model defines standards for (i) Properties that individual components must satisfy and (ii) methods, and possibly mechanism, for composing components. This definition looks most general and appropriate definitions which give clear idea about the component. So A component is basic building block or software unit which contains, code for performing services and, interfaces for accessing those services and also conforms to a component model which defines standards for properties that individual component must satisfy and methods and mechanism for composing those components. To design a component, software reuse is an important
strategy for improving software development efficiency and increasing the quality of software systems [8-10]. In present time reuse approaches are used as a part of component based system (CBS). CBD also involves reusing application frameworks, which provide the architecture for assembling components into a software system. The recent standardization of reuse architectures, such as the Common Object Request Broker Architecture (CORBA), Sun’s JavaBeans, and Microsoft’s Component Object Model (COM) enable reuse of software artifacts across development groups. Moreover, it allows components to be freely exchanged between and across application domains and development contexts—to such an extent that it is now feasible to recognize the existence of software component markets [11]. Reusing components made for earlier products as an approach to new system development is a promising way of achieving the mentioned development and system improvements. There is also the possibility to buy software components from component vendors, so called Commercial-Off-The-Shelf, COTS, and components. Using Commercial-Off-The-Shelf, software components is increasing in today’s development of new systems. Shorter system life cycles and decreased development budgets make it. Using COTS, Components can be one way of reducing development time and be competitive by getting products to the market fast and inexpensively. COTS components can also provide an increased reliability compared to custom-made components since they are refined by substantial field-testing. Certain types of components are rarely developed solely for one intended system anymore. Development of a database management system or an operating system as a part of a larger project is almost unthinkable for many applications today [24].

Moreover, a component is the function or job of software which is to be done and component-based software models are the models by which all the composite components are interconnected by few mechanisms and all individual components are specified, providing few rules for their properties. So to be a component, it must have the following properties-(a) a component may be used by the other component (b) a component defines its function or methods by the interfaces (c) a component can be reused as a part of the other components. In object oriented approach, a system is designed as a set of interacting objects, but in component based software engineering, a system is designed from a set of components [1]. It has been noticed that many proposals have been made to develop CBSE but there was no precise way to design CBSE. The principles of the component based software (CBS) and their notations have not been clearly explained and also not formally defined. The best way to specify the components is to define it using formal specification because a formal specification is simple description of a system using a mathematical notation. In formal language, the semantic and formalism of the system at design domain are well defined. The main advantage of using mathematics is it is precise, unlike the more ambiguous natural language and diagrams which are often used for specifications. A specification language can be used as a design tool and the notations as documentation tool. The actual process of designing a system can be undertaken using a formal notation to communicate ideas between members of a design team.

This paper focuses to formally specify the Component Based Software (CBS) using Z-notation because Z-language is a formal specification language and it supports all the required properties of the component. Z uses not only mathematical notation to describe properties of information system without constraining the way in which these properties are achieved, it also uses the schema to define these properties. They describe what the system must do instead of saying how it is to be done. This abstraction makes Z useful
in the process of developing a system. Z-specification can serve as a single, reliable reference point for those who investigate the customer’s needs, test results and write instruction for the system. It makes easier to write mathematical description of complex dynamic systems such as software. The descriptions are usually smaller and simpler than any programming language can provide. They should contain a mixture of formal and informal parts. Z is based on the standard mathematical notation using mathematical data types to model the data in a system. It also used in axiomatic set theory, first-order predictive logic, lambda calculus. Another thing is that it has a way of decomposing a specification into small piece called schemas. After splitting the specification into schemas, it can present it piece by piece. Every piece is connected with formal mathematics. All expressions in Z notation are typed, thereby avoiding few of the paradoxes of set theory. Z contains a standard mathematical toolkit of commonly used mathematical functions and predicates, called catalog. For detail reference on Z notations refer [17].

With these objectives, the paper has been organized in seven sections. In Section 2 previous researches related to the domain i.e., component based software engineering (CBSE) have been summarized. In Section 3, the ZFSCM has been introduced to specify the modeling elements of a component for CBSE using Z notation. This section also defines the relationships among the components. In Section 4, type-checking has been done to validate the model. In Section 5, a case study has been done to illustrate the model. In Section 6, major characteristics of the model have been summarized. This section also includes a comparative study of all CBSE data models and the paper has been concluded in Section 7.

2. Related Research

In early years, component based software engineering (CBSE) have been proposed few models to define and specify the components. It has been noticed that basically two types of approaches had been introduced in different proposals, Implementation approaches and design approaches. In implementation approach, since the main importance is on the solution domain so, the analysis of the problem domain is ignored and also to design a system as a whole, model design is very much needed. UML is basically based on implementation approach, similarly SOFA [32] is based on design approach. This paper has focused on the design domain. There are several proposals have been made based on implementation approach. Refinement of Component and Object System (rCOS) has been proposed [29], which is based on a unified multi-view modeling approach that is intended to formalize the notations and support separation of concerns to deal with the difficulties. ECM component model has been proposed [3], which attempts to develop a software component model especially the data access architecture using Unified Modeling Language (UML). Many of these approaches [Kotonya [26], IvicaCrnkovic [27], Ning [1], Anantula [6], Kaur [5], Crnkovic [4], Miller [36]) have been discussed the concept of CBSE for modeling of components. The major drawbacks of these proposals are in representation of formal specification of components. All of these proposals have specified the components in informal way. In recent literature, few attempts have been made to model components of CBSE paradigm. Now the major models description have been done in follow

In [3] ECM has been proposed which supports the component model partially. This approach is based on UML. It supports Object – Oriented paradigm and data access architecture in a component based application. Using the implementation of Data
Adapter interface, it expressed the interaction between the data access objects with business – tier and data – tier in achieving reusable, robust and scalable component based application. It also has been made the unit test on the code of the data adapter interface at the time when code efficiency increased and data access object implements the adapter, to validate the result. However, it supports the code efficiency increases by 31% and reduced code complexity in the Data Access Object layer of the multi – tier architecture. Moreover it does not provide any guideline for the formal specification of the component.

In [29] unified multi- view modeling approach called rCOS has been proposed to formalize the notations and support system construction by composing components. In this model interfaces and their contracts have been used by the users for using the services of the components. The provided interfaces and required interfaces have been extracted for component to provide the black- box specification. It supports few features of the modeling elements of the components in CBSE such as data-hiding, deadlock and divergence freedom, decomposing the components, contracts, synchronous communication and asynchronous communications.

In [6], no specific model has been considered. Few behavioral patterns have been proposed for creating components of CBSE such as subtraction, multiplication, division, task- unification, and attribute dependency change. Few applications have been done and made comparison with the tradition approach with the help of few parameters such as reliability, maintainability, adaptability and testability. It assured high quality software product to the customer. However, there is no formal design model of components has been considered in this approach.

In [1], component based architecture model has been proposed which supports provided interfaces and required interfaces, binding or interconnection of components and system configuration. An extended of CORBA IDL and uses of OLE/ COM’s multiple interface have been developed by an architecture specification language. Using the component specification tools, the compatibility of interfaces between components and adapters’ generation to bridge in compatible interface, can be checked. It has been transformed abstract component – based architecture specifications into executable code which may run on distributed communication infrastructure such as CORBA or OLE by integration tools. This model also supports to bind together of different pieces under a graphical, interactive and user – oriented environment. This model is an extension of CORBA IDL, so it does not support any formal specification of component.

So it has been already seen that several attempts have been made for the CBSE on implementation and as well as design approaches but only few proposals can support partially to specify it formally. Few proposals support modeling elements, few proposals are based on UML and few proposals extends UML. As UML is not formally specified, these proposals have not been formally specified. Few proposals support mathematical formulation to find the complexity of components. So there is no proper guideline by which CBS can be modeled not only by formal specification, also supports different properties of CBS. The principals of the CBS and notations of the components must be clearly explained and also formally defined. This paper focuses to describe the modeling elements formally and the notations of the software components are cleared.


The Z-Formal Specification of Component Model (ZFSCM) is based on Z notation. In ZFSCM, a component has been modeled formally with the relevant properties. It is a
collection of services, classes and interfaces. These elements are composed into a component. Different graphical notations (Table 1 have been used for clear visualization of the modeling elements separately. In ZFSCM, it defines the inheritance property by which it describes that a component may be used by the other components. It also describes aggregation property by which a component can be reused as a part of the other components.

3.1. Structural Modeling Constructs of ZFSCM

Based on earlier discussion, this paper intends to formalize the characteristics of a component in a model with few modeling elements of component in component based software engineering. There are several modeling elements in CBSE, This paper describes and design few of those elements.

a) Service: A service can be defined as a small part of specification. It specifies that the thing user is doing. A service contains an Eid for itself, a Cid for a class, and the description where it describes the service it provides which has declared globally at the time of type checking. A service can be graphically noted as a triangular box. Figure 1(a) shows the Z specification of service.

![Figure 1(a-b). Z Specification of Service & Class](image)

b) Class: A class can be defined as a collection of services i.e., the overall description of the related services. It implements one or more interface also. A class contains and Cid for itself, and an Eid where Eid are taken from the domain of services. Another condition is that Eid must have a limit which means it specifies that how many services can be occupied by the class. A class can be graphically noted as circle. Figure 1(b) shows the Z specification of class.

c) Interfaces: It describes the structural nature of a system and is only used for checking syntactic dependencies and compositionality. They are represented in terms of signatures of service operations. An interface contains an Iid for itself, a Cid. The Cid must be the domain of the class and it must have a limit which specifies that how many classes may use or implement this interface. An interface can be graphically noted as pipe. Figure 2(a) shows the Z specification of interface.

![Figure 2. (a –c). Z Specification of Interface, Component & Limit](image)
Component: It is a collection of classes and interfaces by which the small pieces of specification can be combined into single unit. A component contains a Coid for itself, Cid and Iid where Cid & Iid are taken from the domain of class and interface respectively and they have limits which specify that how many classes or interface may be composed into the component. A component can be graphically noted as rectangular box. The limit is a global variable. Figure 2 (b & c) shows the Z specification of component & limit respectively. The summary of ZFSCM constructs and their graphical notations have been given in Table 1.

### Table 1. Summary of ZFSCM Constructs and their Graphical Notations

<table>
<thead>
<tr>
<th>ZFSCM Constructs</th>
<th>Graphical Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>△</td>
</tr>
<tr>
<td>Class</td>
<td>○</td>
</tr>
<tr>
<td>Interface</td>
<td></td>
</tr>
<tr>
<td>Component</td>
<td>□</td>
</tr>
</tbody>
</table>

3.2. Relationship in ZFSCM

a) **Aggregation:** An aggregation is an operation which may be defined as a collection of data as input and returns a single data as output. Here the data means the services, classes or components. It is denoted by an arrow with a diamond at the tail end (→○). Figure 3 shows the aggregation operation between the services, classes and the component.

![Figure 3. Aggregation between the Component, Classes and Services](image)

The Z specification of the above relation can be derived in two ways- (a) services are aggregated into class and (b) classes are aggregated into components. The following schema diagrams show these aggregations.

![Figure 4. (a-b).Z Specification of Aggregation Operation](image)
aggregated into a Component. Here Eid is the member of class and scope indicates the aggregated classes. Here, the class do not change it member.

Again the components may be aggregated into an interface that may be called by another component then the calling component uses all the aggregated components via that interface. This may be graphically shown in the Figure 5(a).

![Figure 5(a-b). Aggregation between the components and Interface and their Z Specifications](image)

The Figure 5(b) describes that different components can be aggregated into a Component. Here Cid is the member of Component1 and scope indicates the aggregated Components with its classes. Here, the Component1 do not change it member.

b) Inheritance: An inheritance is a process by which classes of one component acquire the properties of classes of another component. Inheritance solves the problems of repeating codes. It reuses the code of one class to another class. The child class or inheriting class includes all of the properties of the parent class or inherited class implicitly, and can also add or modify to these properties. It is denoted as an arrow pointed to the parent component from the child component ( ). In the model if A be a parent class and B be a child class then inheritance can be shown in Figure 6.

![Figure 6. Single Inheritance](image)

Here class B inherits class A maintaining the same interface I1 and/or it can be extends interface by including the new operation. This is a case of single inheritance. The Z specification of single inheritance can be defined as follows. Here, first the class A is defined, and then the interface I1 is defined which inherits the class A, after that the class B is defined which implements the interface I1. Now the class A, class B & interface I1 can be defined as follows-

![Figure 7. Z Specification of Single Inheritance](image)
It is possible to inherit more than one class simultaneously. This is the case of *multiple-inheritance*. In Figure 8, \( A, B \) are the parent *classes* and \( C \) is the child *class*.

![Figure 8. Multiple Inheritances](image)

The Z specification of *multiple inheritances* has been shown in Figure 9. Here, first the *class A & class B* are defined, and then the *interface I_2* is defined which inherits the *class A & class B*, after that the *class C* is defined which implements the *interface I_2*. Now these can be defined as follows-

![Figure 9. Z Specification of Multiple Inheritances](image)

c) **Interaction specification:** There are several interactions can be made using the Z notation such as compositions between the components *etc*. A component can be two types: *calling* component and *called* component. For each *calling* component there must be a *called* component. One component can call another component via an interface. One component can call two separate components via two separate interfaces or through only one interface. Two separate components can call a component via an interface only. Now more precisely, calling/called the component means calling/called the particular service with its related class through one or more interfaces as it requires. So it can be easily understood that there are two basic interactions happen, one is the interaction from a class to interface and the other is from interface to class. This paper here describes each of these interactions separately and gives a graphical representation of these interactions. Here using arrow it specifies the direction of the component which based on the component type. The Direction of the arrow is from calling component to interface or from interface to called component. It also specifies the service with related class using dot (.)

Case 1: One component calls another component via an interface. Suppose a service \( S31 \) which belong to the class \( C3 \) of the component \( comp2 \) called by the service \( S11 \) of the class \( C1 \) of the component \( comp1 \) via the interface \( I1 \). This interaction can be graphically plotted as below.
Case 2: One component calls another component via two separated. Suppose the services $S_{31}$ & $S_{41}$ which belong to the class $C_3 & C_4$ respectively, of the component $comp_2$ called by the service $S_{11}$ & $S_{21}$ of the class $C_1 & C_2$ respectively, of the component $comp_1$ via two separate interfaces $I_1$ & $I_2$ respectively. This interaction can be graphically plotted as below.

Case 3: One component can call two separate components via one interface. Suppose $comp_1$ has two classes $C_1 & C_2$ with the services $S_{11}, S_{21}$ respectively and $comp_2$ has one class $C_3$ with service $S_{31}$ and $comp_3$ has one class $C_4$ with service $S_{41}$ and the interface can be defined as $I_1$ is between the classes $\{C_2, C_3\}$. Now the service $S_{41}$ calls both services $S_{21}, S_{31}$. This interaction can be graphically plotted as below.

Case 4: Two separate components can call only one component via one interface. Suppose $comp_1$ has two classes $C_1 & C_2$ with the services $S_{11}, S_{21}$ respectively and $comp_2$ has one class $C_3$ with service $S_{31}$ and $comp_3$ has one class $C_4$ with service $S_{41}$ and the interfaces can be defined as $I_1$ is between the classes $\{C_1, C_2\}$. Now the services $S_{41}, S_{31}$ call both services $S_{11}, S_{21}$. This interaction can be graphically plotted as below.
The summary of ZFSCM relationship types and their graphical notations have been given in Table 2.

Table 2. Summary of ZFSCM Relationships and their Graphical Notations

<table>
<thead>
<tr>
<th>ZFSCM Relationships</th>
<th>Description</th>
<th>Graphical Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregation</td>
<td>Defined between two elements or components where tail end diamonds indicates the aggregating elements and arrow point to the aggregated elements.</td>
<td><img src="image1" alt="Diagram" /></td>
</tr>
<tr>
<td>Inheritance</td>
<td>Defined between two elements or components, called child elements and parent elements. An arrow pointed to the parent component from the child component.</td>
<td><img src="image2" alt="Diagram" /></td>
</tr>
<tr>
<td>one-to-one</td>
<td>Defined between one calling element and one called element via an interface.</td>
<td><img src="image3" alt="Diagram" /></td>
</tr>
<tr>
<td>one-to-many</td>
<td>Defined between one calling element and more than one called element via an interface.</td>
<td><img src="image4" alt="Diagram" /></td>
</tr>
<tr>
<td>many-to-one</td>
<td>Defined between more than one calling element and one called element via an interface.</td>
<td><img src="image5" alt="Diagram" /></td>
</tr>
</tbody>
</table>

4. Type Checking

Z is a non-executable language and has no compiler but it is strongly-typed specification language. This paper uses ZTC type-checker for Z notation which determines whether there are any syntactical and typing errors or not. This paper uses ZSL for the input which may be written in plain text style and save it with .zsl extension. After saving the inputted files, ZTC may be run to check for type errors with the commands. For further reference of ZTC referred to [16].

4.1. Type-checking of the Modeling Constructs of ZFSCM

Here this paper considers all the construct form (e.g., service, class, component and interface) of ZFSCM. It has been noticed that type checking is the process to check the validity of the designed component. It also provides the report of the type checking to each of the model. Now the above form of Z-notation can be inputted in zsl form as follows-

The ZSl form of input for ‘component’ is as follows

```plaintext
specification
  [ ClassID , InterfaceID , ComponentID ]
global
  limit : N
axiom
  limit = 1
end axiom
Response ::= success
```
\begin{verbatim}
| not
\end{verbatim}

\%\% state-schema Component

\begin{verbatim}
schema Component
Cid : P ClassID ; Iid : P InterfaceID ; Coid : P ComponentID
where
# Cid >= limit;
# Iid>= limit;
end schema
end specification
\end{verbatim}

The corresponding type report of the ‘component’ is as follows

Type Report
Input file: component-1.zsl
given sets
ClassID
InterfaceID
ComponentID
Response
end given sets

\begin{verbatim}
global names
limit : Z
success : Response
not : Response
end global names
schema Component
Cid : (P ClassID) ([P ClassID])
Iid : (P InterfaceID)([P InterfaceID])
Coid : (P ComponentID) ([P ComponentID])
end schema
\end{verbatim}

In the above, \textit{ClassID}, \textit{InterfaceID}, \textit{ComponentID} and Response are four global specifications, \textit{limit} is a global variable. Now see, it describe that \textit{Cid} is a member of \textit{ClassID} and \textit{Iid} is member of \textit{InterfaceID}. \textit{Coid} is a member of \textit{ComponentID}. In the same way, other forms can be type checked.

\section*{4.2. Type-checking of the Relationships of ZFSCM}
In this section all the relationships which declared in Section 3.2 have been type checked. The relationships like aggregation, inheritance and different types of interaction have been type check here. It also provides the report of the type checking to each of the relations. The method type checking of aggregation relation (services into \textit{class} and \textit{classes} into \textit{component}) can be inputted as follows. Suppose, the four services (A,B,C,D) define with two classes (CL1&CL2). The services A & B are aggregated into the class CL1 and the services C & D are aggregated into the class CL2 and finally the classes CL1&CL2 are aggregated into the component Co. In the same way other relations can be type checked as given in Section 4.1.

\section*{5. Case Study}
This paper considers a library management system where a member can register, cancel membership, issue and return books from library. To describe these four operations, this paper combines the services into \textit{classes} and the \textit{classes} into
components. Here the total system divided into five components to describe these operations viz. Library-Management, Authorization-Management, Transaction-Management, Member-Management, and Book-Management. Each component contains several classes and each class performed few services described in Table 4. The service descriptions are detailed in the Table 3. Now the system has been created using these five components together. Now the component model of this system can be viewed as Figure (9) below.

![Component Model of a Library Management System](image)

**Figure 14. Component Model of a Library Management System**

This paper consider the services for Library Management System which are listed in the Table 3.

**Table 3. Services with Service ID**

<table>
<thead>
<tr>
<th>Service ID</th>
<th>Service description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Personal details are entered</td>
</tr>
<tr>
<td>S2</td>
<td>Checking is made whether the member is already exists or not</td>
</tr>
<tr>
<td>S3</td>
<td>Generate Member id</td>
</tr>
<tr>
<td>S4</td>
<td>Book details are entered</td>
</tr>
<tr>
<td>S5</td>
<td>Checking is made whether the book is already exists or not</td>
</tr>
<tr>
<td>S6</td>
<td>Generate Book id</td>
</tr>
<tr>
<td>S7</td>
<td>Member id is entered</td>
</tr>
<tr>
<td>S8</td>
<td>Member id is validate</td>
</tr>
<tr>
<td>S9</td>
<td>Book id is entered</td>
</tr>
<tr>
<td>S10</td>
<td>Book id is validate</td>
</tr>
<tr>
<td>S11</td>
<td>Member is applying for issue a book</td>
</tr>
<tr>
<td>S12</td>
<td>Checking is made whether the member is allowed for issue or not</td>
</tr>
</tbody>
</table>
The components, classes related to the above services are listed in Table 4. It describes that the component which owns the classes and the services. In Library Management System, as discussed earlier, there are six major interactions can be considered. Figure 10(a)-10(f) shows these interactions with the calling classes to called classes. In Figure 10(a)-10(f) several interfaces have been used. The Table 5 has listed those interfaces corresponding to the interfaceID.

<table>
<thead>
<tr>
<th>Component ID</th>
<th>Component Name</th>
<th>Class ID</th>
<th>Class Name</th>
<th>Service ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP1</td>
<td>Library-Management</td>
<td>C1</td>
<td>Librarian</td>
<td>S1, S4, S7, S9</td>
</tr>
<tr>
<td>CP1</td>
<td>Library-Management</td>
<td>C2</td>
<td>Keyboard</td>
<td>S11, S16, S21, S26</td>
</tr>
<tr>
<td>CP2</td>
<td>Member-Management</td>
<td>C3</td>
<td>Registration</td>
<td>S2, S3</td>
</tr>
<tr>
<td>CP2</td>
<td>Member-Management</td>
<td>C4</td>
<td>Member-Details</td>
<td>S22, S24</td>
</tr>
<tr>
<td>CP2</td>
<td>Member-Management</td>
<td>C5</td>
<td>Cancel-Membership</td>
<td>S27</td>
</tr>
<tr>
<td>CP3</td>
<td>Book-Management</td>
<td>C6</td>
<td>Book</td>
<td>S5, S6</td>
</tr>
<tr>
<td>CP3</td>
<td>Book-Management</td>
<td>C7</td>
<td>Book-Details</td>
<td>S23, S25</td>
</tr>
<tr>
<td>CP4</td>
<td>Authorization-Management</td>
<td>C8</td>
<td>Authentication</td>
<td>S8, S10</td>
</tr>
<tr>
<td>CP5</td>
<td>Transaction-Management</td>
<td>C9</td>
<td>Transaction</td>
<td>S12, S13, S17, S18, S19</td>
</tr>
<tr>
<td>CP5</td>
<td>Transaction-Management</td>
<td>C10</td>
<td>Issue</td>
<td>S14, S15, S7, S9, S20</td>
</tr>
<tr>
<td>CP5</td>
<td>Transaction-Management</td>
<td>C11</td>
<td>Return</td>
<td>S28</td>
</tr>
</tbody>
</table>
Table 5. Interface Name with Interface ID

<table>
<thead>
<tr>
<th>Interface ID</th>
<th>Interface name</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>IR</td>
</tr>
<tr>
<td>I2</td>
<td>ILAT</td>
</tr>
<tr>
<td>I3</td>
<td>ILAM</td>
</tr>
<tr>
<td>I4</td>
<td>ILAB</td>
</tr>
<tr>
<td>I5</td>
<td>ITM</td>
</tr>
<tr>
<td>I6</td>
<td>IBT</td>
</tr>
<tr>
<td>I7</td>
<td>IT</td>
</tr>
<tr>
<td>I8</td>
<td>IIR</td>
</tr>
<tr>
<td>I9</td>
<td>ILA</td>
</tr>
</tbody>
</table>

Figure 16(a). Member-Registration

Figure 16(b). Book-Registration

Figure 16(c). Issue-Book

Figure 16(d). Re-Issue Book

Figure 16(e). Return-Book

Figure 16(f). Cancel-Membership

Figure 15(a-f). Interactions between the Classes using their Services

The ZFSCM representing the above model as-

Figure 16. ZFSCM for Library Management System

Here the interface $I_9 = I_2 \cup I_3 \cup I_4$. 
6. Feature of Component Based Data Model

In this section the paper describes the basic features of component based data model. The list of features has been discussed in the context of the ZFSCM model. The features are divided into different groups which are as follows:

6.1. Meta Modeling Level Features

These set of features include descriptions of related concepts those are used for the purpose devising the conceptual modeling constructs. The set of features are as follow:

a) **Meta model Type**: Few models support operation based interface type e.g., methods invocation and few models supports port-based interface e.g., data passing.

b) **Meta model language**: This shows the language main by Meta data model to express its meta-schema. ZFSCM used Z-language.

c) **Formalism**: This feature indicates the formalism used to describe the conceptual model. ZFSCM is however formal in that sense.

d) **Modeling Specification**: This feature indicates that the models support either architectural description of the systems and components e.g., ADLs, or the specification and the verification of particular system and component properties e.g., meta-model.

6.2. Semantic and Construct Level Features

These set of features are used to describe the expressiveness of a conceptual model towards realizations of the target domain. The set of features are as follows:

e) **Explicit Separation of structure and Content**: This feature enhances the capability for reuse of the elements of any conceptual model. The ZFSCM provides a unique design framework to specify the design for the component based software system using semantic definitions of different levels (from elementary to composite) of data structure through formal specification. In the model, the nature of contents that corresponded with the instances and the functional constraint on the instances has been separated from the system’s structural descriptions.

f) **Abstraction**: Abstraction mechanism is an essential property in OO models to reduce the complexity of the system design. Such a representation is highly flexible for the user to understand the basic structure of semi-structure database system and to formulate the alternative design options. In ZFSCM, the concepts of layers deploy the abstraction in semi-structured data schema. The upper layer views will hide the detail structural complexity from the users.

g) **Reuse Potential**: This is another important property in OO models. This can be achieved either using whole-part relationship or inheritance mechanism. The ZFSCM is supported with inheritance mechanism using the Link relationship.

h) **Hierarchical and Non-hierarchical Structure**: The ZFSCM explicitly supports both hierarchical and non-hierarchical representation in meta-modeling.
at conceptual level. Associated components of different or same layers form the hierarchical or non-hierarchical structure in meta-model.

i) Services: A service can be defined as a thing what user is doing. The ZFSCM support this service mechanism to describe its basic functions in the meta-model constructs level.

j) Classes: A class can be defined as a collection of services and variables. The ZFSCM support this mechanism to describe its classes in the meta-model constructs level.

k) Components: A component can be defined as a collection of classes and its services. The ZFSCM support this mechanism to describe its components in the meta-model constructs level.

l) Inheritance: Inheritance means that each instance of a subclass truly is an instance of the super class. So, all operations and attributes of the super class must uniformly apply to the subclass. Inheritance should only be used when the generalization relationship is semantically valid. The ZFSCM support inheritance property.

m) Aggregation: Aggregation is the “part-whole” or “a-part-of” relationship in which objects representing the components of something are associated with an object representing the entire assembly. Aggregation is a tightly coupled form of association with few extra semantics. The ZFSCM supports aggregation to satisfy the transitivity and anti- symmetric properties.

6.3. Comparative Study of Component based Model

The set of features described in subsections 6.1, 6.2 are used to compare different component models which are based component based software engineering. The models described in [23, 21, 31, 32] are used for that purpose. Interested readers can also follow the [4] for the comparison study related to the component based models. The comparison has been summarized in Table 6. Majority of the models do not support hierarchical and Non- hierarchical structure. Also majority of the model are not formal.

### Table 6. Comparative Study of Component Based Model

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Interface Specification and Modeling Level</th>
<th>Semantic and Construct Level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (k) (l) (m)</td>
<td></td>
</tr>
<tr>
<td>CCM</td>
<td>operation based, port based</td>
<td>CORBA IDL,CIDL _ N/A Y Y Y _ _ _ Y _ _</td>
</tr>
<tr>
<td>KobrA</td>
<td>operation based</td>
<td>UML _ UML profile N/A Y N/A Y Y Y Y _ Y</td>
</tr>
<tr>
<td>Palladio</td>
<td>operation based</td>
<td>UML _ UML profile N/A Y N/A _ Y Y Y _ _</td>
</tr>
<tr>
<td>SOFA</td>
<td>operation based</td>
<td>Java, SPC algebra _ Meta model based specific Y Y Y Y Y Y Y P _</td>
</tr>
</tbody>
</table>
7. Conclusion

This paper proposed a formal specification model of component based software using Z-notation. This paper guides to describe a component based software system formally. It also gives a generic guideline to formalize different components and the interfaces of a component based software system. In this paper, different feature of the component based system has been discussed. A type-checking has been done using ZTC. This paper also describes the practical approach of the ZFSCM model with a case study using the Library-Management system. The paper also highlighted the essential features for component based models. A detailed comparison study also has been performed among few of the available component models for such system and ZFSCM. Further, the paper also has included few future research directions with high potential for component based software domain.

This paper proposes the different modeling elements, such as services, classes, interfaces and also the different relationships models among those elements such as aggregation, inheritance etc., of a component for component based software engineering (CBSE) and describes their properties using Z notation and checks the validity using the type checker for Z-notation ZTC [16] which determines the syntactical and typing errors in Z specifications. As there is no compiler for Z, ZTC has made a strong support for determining different syntactical and typing errors. Though ZTC accepts two type of inputs viz. LATEX with oz or zed packages, and ZSL, this paper follows the inputs of the different modeling elements and their relationships models in ZSL form. In future, the ZFSCM can construct using Object-Z specification. Using ZFSCM, different Case Tools can be generated which nothing but few API’s that must be expressive and user friendly.

References


Authors

Prasenjit Banerjee, presently a faculty member in the Midnapore college, Paschim Midnapore, West Bengal, India. He completed Master of Computer Application from Vidyasagar University in 2009. His domain of research interest is Software Engineering.

Anirban Sarkar, presently a faculty member in the Department of Computer Applications, National Institute of Technology, Durgapur, India. He received his PhD degree from National Institute of Technology, Durgapur, India in 2010. His areas of research interests are Database Systems, Software Engineering and Cloud Computing. His total numbers of publications in various international platforms are about 45. He is actively collaborative research with several Institutes in India and USA. He has also served in the committees of several international conferences in the area of software engineering and computer applications.