Components Interaction Markup Language for Mediator Connector

H. Sanatnama, A.A.A. Ghani, R. Atan and M.H. Selamat
Faculty of Computer Science and Information Technology,
University Putra Malaysia, 43400, Serdang, Malaysia

Abstract: The concern of interaction or collaboration between components can be found when evolution of software engineering came a long way from machine-level language to procedural programming and then to object-oriented programming and now to component-based software development. An interaction is a set of activities that happens for a specific use case in a system, based on the ability of components (requires and provides services) to send messages to each other. This study introduces Component Interaction Markup Language (CIML) as an improvement of the attachment uses by mediator connector we proposed in earlier study. CIML aims to make the attachment well-formed as a generic framework for component composition based on interactions between components. CIML supports component composition based on interactions between components and has language constructs for description of component instantiations, component initializations and component interactions.

Key words: Composition language, interaction, component composition, mediator connector, deployment phase

INTRODUCTION

Composition of software components is a fundamental issue within the component-based software development. Component composition is the process of creating component instances, configuring and assembling them together to form composite components or application. Unfortunately components provide little support for automatic composition of components (Selamat et al., 2007).

The mediator connector proposed by Sanatnama et al. (2008) for composing decoupled software components; need a way to build up a system in a generic way. Composition of software components via mediator connectors requires encapsulation of interactions between components which should be considered as a separate issue.

The concern of interaction or collaboration between components can be found when evolution of software engineering came a long way from machine-level language to procedural programming and then to object-oriented programming and now to component-based software development. An interaction is a set of activities that happens for a specific use case in a system, based on the ability of components (requires and provides services) to send messages to each other.

An interaction is a sequence of method calls to components connected to a mediator connector. Therefore, it requires a language to describe how components interact with each other. Sanatnama et al. (2008) does not suggested how to do this. Hence, the problem we are interested in can be formulated as follows:

- How to design an interaction language, which can be seamlessly integrated in the mediator connector?

In this study, we propose Component Interaction Markup Language (CIML). The CIML is a declarative language where the interactions between components can be described. A CIML instance (document) contains different parts, which are instantiation, initialization and interactions.

CIML is influenced by Business Process Execution Language (Oracle, 2004, 2006), which has emerged as the clear standard for composing multiple synchronous and asynchronous services into collaborative and transactional process flows. BPEL uses Web Services/WSDL (Rayan, 2007) as component model and XML as data model (data loose-coupling).

MATERIALS AND METHODS

In recent years, XML (W3C, 1998) has been one of the most exciting developments in information technology. At its simplest, XML is a markup language for describing information regardless of the platform or
application that may use it. However, other markup languages such as HTML and voiceXML, which is designed for creating audio dialog that feature synthesized speech, digitized audio and recognition of spoken, requires specific applications to make them useful.

XML is little more than a text file and are stored as files on a hard drive. XML documents consist of intermingled character data and markup. Markup differentiates some character data from character data in other files. The markup texts are embraced in angle braces to produce self-descriptive elements known as markup tags. Table 1 show that the XML specification defines different kinds of markup and serve different purposes.

The promising wide acceptance of XML (Bray et al., 2004) and the integration into software tools convinced us to use XML based syntax for CIML. Because of the nature of XML, CIML consist of XML elements. This also makes CIML scripts created and used by software tools rather than by human beings.

The XML is considered to be useful in the field of software engineering, especially in describing components interactions. In this research, the XSD has been adopted to define the CIML structure, mainly because it has been widely used in the industry.

### REQUIREMENTS FOR CIML

The design objective of the Components Interaction Markup Language (CIML) is to provide a canonical representation of any interaction diagram suitable for mapping to existing languages. CIML should provides a highly language-independent method to describe components interaction.

CIML factors any component interaction description into following questions:

- What are the parts comprising the CIML?
- What is the content used in CIML?
- What is the behavior of the CIML (e.g., when some component send a message to another component)?
- What is the mapping of the parts to CIML controls in some development environment (e.g., Java Language)?

The purpose of his section is to identify the operations CIML should contain in order to support above questions. The following objectives must be supported be the CIML:

- **Loosely coupling**: The main objective of mediator connector is providing loosely coupling. Therefore, CIML must be able to declare the instantiation of the components
- **Initialization**: Sometimes it is necessary to initialize a component explicitly before using it. But this can be optional
- **Interaction**: The aim of CIML is in a well-formed constructer describe the interactions between components. It should support to describe the set of activities that happen for a specific use case in a system, based on the ability of components (requires and provides services) to send messages to each other.

Mediator connector initiates and coordinates method calls to the components and handles their results. Thus, it encapsulates communication. From another view mediator connector is similar to a hub of communication. The origination of control from mediator connector leads to total loosely coupling between components and mediator connectors.

Figure 1 shows the process of building up a system by composing component in deployment phase. The behaviors of the components in a system are described in sequence diagrams, which show the set of components involved in the interaction. The sequence diagrams will be mapped to a CIML document according to its syntax. Mediator connector loads and parses the CIML file, where all components instances and their relations, based on the description in the CIML file, will be created.

Mediator connector will parse the CIML document and build up the system by creating all components and the connections (method calls) described in each interaction. Each interaction can be run by invoking the run method in mediator connector giving the name of interaction as in-parameter.

<table>
<thead>
<tr>
<th>Markup type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing instruction</td>
<td>Provides information to application during processing</td>
</tr>
<tr>
<td>Comments</td>
<td>Blocks of text that provide documentation or notes</td>
</tr>
<tr>
<td>Entity references</td>
<td>References to entities such as character data or XML; used to insert character data that may be considered markup</td>
</tr>
<tr>
<td>CDATA sections</td>
<td>Character data that do not need to be parsed or treated as markup</td>
</tr>
<tr>
<td>DOCTYPE declarations</td>
<td>Indicate a DTD containing the constraints for an XML document</td>
</tr>
<tr>
<td>Start and end tags &lt; &gt; and &lt;/ &gt;</td>
<td>Define the beginning and ending of elements</td>
</tr>
<tr>
<td>Empty element &lt;/ &gt;</td>
<td>Defines an element with no body or content</td>
</tr>
</tbody>
</table>
**CIML specification:** The Component Interaction Markup Language (CIML) is the Extensible Markup Language (XML) which describes interaction between components. CIML is influenced by Business Process Executable Language (BPEL) (Leymann, 2007; Keller, 2007) and the syntax of some CIML elements is influenced by IMB’s Bean Markup Language (Weerawarana et al., 2001), Component Markup Language CoML (Birngrubner, 2001).

By CIML, we try to provide a practical and cost-effective means for component composition through mediator connector. The CIML specification is the document that describes the structure of CIML i.e., defines what the elements (vocabulary) are and how these elements are used (grammar) to create CIML. CIML files must conform to the XML 1.0 specification, as published on the World Wide Web Consortium (W3C) website.

The promising wide acceptance of XML (W3C) and the integration into software tools convinced us to use XML based syntax for CIML. Because of the nature of XML, CIML consist of XML elements. This also makes CIML scripts created and used by software tools rather than by human beings.

**EBNF grammar for CIML:** The syntax of CIML is summarized in an EBNF-like form as shown in Table 2, for above description rules of CIML. All symbols on the left side of production rules are bold and on the right hand side we have alternatives, which consist of both symbols and terminals. The reserve words are enclosed in quotes ("").
Table 3: CML elements (tags)

<table>
<thead>
<tr>
<th>Element name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CML</td>
<td>Indicates the start of a CML document</td>
</tr>
<tr>
<td>Contains</td>
<td>Declares a list of existing components</td>
</tr>
<tr>
<td>Component</td>
<td>Declares a component</td>
</tr>
<tr>
<td>Initialize</td>
<td>Declares a list of initializing method calls</td>
</tr>
<tr>
<td>InitCall</td>
<td>Declares an initializing method call</td>
</tr>
<tr>
<td>Interaction</td>
<td>Declares an interaction in the system</td>
</tr>
<tr>
<td>OperationCall</td>
<td>Declares a method call</td>
</tr>
<tr>
<td>Inparameter</td>
<td>In-parameters to the method</td>
</tr>
<tr>
<td>Outparameter</td>
<td>Returns value from a method call</td>
</tr>
</tbody>
</table>

Table 4: Element descriptions for components declaration

<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains</td>
<td>-</td>
<td>Collection of components in the system</td>
</tr>
<tr>
<td>Component</td>
<td>Identifier, ClassName</td>
<td>Declares component’s identifier and class name</td>
</tr>
</tbody>
</table>

beginning, we present a list of CML elements as shown in Table 3. The limitation of CML elements, keep CML as simple as possible. CML provides elements for:

- Describing components by a unique id and its class name
- Describing eventual explicit initialization of components
- Describing the component composition in form of an interaction

The syntax of some CML elements is influenced by IMB’s Bean Markup Language (BML) (Weerawarana et al., 2001), Component Markup Language (CoML) (Bringruber, 2001) and Business Process Executable Language (BPEL) (Leymann, 2007; Keller, 2007).

Table 3 describes briefly all elements defined in a CML document, in order to declare all possible interaction between components in a system. Each of these elements are described in details mentioned further. XML elements can contain only text or contain other elements or attribute.

**Component declaration:** Components are accessed via interfaces and have an implementation. The element Contains describes a list of components whereas the element Component denotes a component declaration whose value (i.e., the component) is to be used as the argument of the enclosing element.

Component element has an Identifier and a ClassName attribute, respectively. The Identifier has to be unique within the current CML document. Components will be created based on the ClassName attribute. Table 4 shows the Contains and Component elements, their attribute and their definitions.

**Example 1:** This example declares a component list with three components of different types and two components of the same type ComD. This example shows that CML supports also multiple (i.e., components of the same type) instance declaration.

```xml
<Clml>
  <Contains>
    <Component Identifier="a" ClassName="ComA"/>
    <Component Identifier="b" ClassName="ComB"/>
    <Component Identifier="c" ClassName="ComC"/>
    <Component Identifier="d1" ClassName="ComD"/>
    <Component Identifier="d2" ClassName="ComD"/>
  </Contains>
</Clml>
```

reveals the interaction (sequence) diagram that corresponds to the above example. At the top of this diagram we see the rectangles that represent components (objects). As in interaction (sequence) diagrams, the names of the components (objects) are underlined to distinguish them from classes. Also the object name is separated from the class name by a colon. Interaction (sequence) diagrams show the elements that are involved in the interaction and also represents time proceeding (Fig. 2).

**Initialize declaration:** Initialize element is another part of a CML document which has no attribute but contains a list of element InitCall. InitCall indicates if some components should be initialized explicitly before using them in an interaction. It has two attributes ComponentId and Name. Table 5 shows the Initialize and InitCall elements, their attributes and definition.

**Example 2:** This example declares a list of different initialize method calls in three different components.
Fig. 3: Initializing in an interaction diagram

Table 5: Element descriptions for initial method calls

<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize</td>
<td></td>
<td>Collection of initialize method call</td>
</tr>
<tr>
<td>InitCell</td>
<td>ComponentId, Name</td>
<td>Initial method call</td>
</tr>
</tbody>
</table>

<InitCall>

..........
</InitCall>

<InitCall ComponentId="ComA" Name="initialize"/>
<InitCall ComponentId="ComB" Name="initi"/>
<InitCall ComponentId="ComD" Name="InitComponent"/>

</Initialize>

..........

<InitCall>

Figure 3 shows the interaction (sequence) diagram that corresponds to the above example, where components (objects) have a method call to themselves.

Interaction declaration: Interaction element is a part of a CIML document where describes interactions between components as a set of activities that happens for a specific use case in a system. Interactions are based on the ability of components (requires and provides services) to send messages to each other. Interactions have unique names which can be distinguished from each other. Table 6 shows the Interaction and its sub-element, OperationCall. OperationCall indicates a message send to a specific component with specific Inparameter and Outparameter, in order to do a task in a system.

Example 3: This example describes an Interaction called DisplayMap. The interaction starts when the system receives a telephone call. The telephone number sends to the ComA by calling its getAddress() method. ComA returns the address assigned to the telephone number. The returned address sends to the ComB by calling getPostCode() method in order to get the postcode assigned to that address. Finally, the postcode sends to the ComD by calling displayMap() in order to show the map for the ambulance driver.

Fig. 4: Interaction description part

Table 6: Element description for an interaction

<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction</td>
<td></td>
<td>Declares an interaction</td>
</tr>
<tr>
<td>OperationCall</td>
<td>ComponentId, Name, [ReferenceId]</td>
<td>A method call</td>
</tr>
<tr>
<td>Inparameter</td>
<td>Identifier, ClassName, [ReferenceId], [Value]</td>
<td>An eventual input parameter to a method call</td>
</tr>
<tr>
<td>Outparameter</td>
<td>Identifier, ClassName</td>
<td>An eventual output parameter</td>
</tr>
</tbody>
</table>

<Interaction>

..........
</Interaction>

<Interaction name="DisplayMap">

<OperationCall ComponentId="ComA" Name="getAddress">
<Outparameter Identifier="address" ClassName="String"/>
</OperationCall>

<OperationCall ComponentId="ComB" Name="getPostCode">
<Parameter ReferenceId="address" ClassName="String"/>
<Outparameter Identifier="postcode" ClassName="String"/>
</OperationCall>

<OperationCall ComponentId="ComD" Name="displayMap"/>
<Inparameter ReferenceId="postcode" ClassName="String"/>

</Interaction>

..........

<InitCall>

Figure 4 represent interaction between components (objects) where a use case scenario shows as ordered steps. This interaction (sequence) diagram corresponds to the above example.

CIML implementation: The CIML schema, which is an XML Schema Definition (XSD), (Fallside and Walmsley, 2004), is used to implement the CIML specification. A CIML instance is a CIML document. The CIML schema will be used by XML analyzer to carry out the validation of rules made by CIML. It provides the means for defining the structure, content and semantics of CIML documents.
The CIML schema diagram: The CIML schema diagram shown in Figure 5 depicts the different parts of CIML document.

The CIML XML schema definition: Figure 6 is the XML Schema Definition (XSD) for CIML. XSD defines the elements and structure of CIML, including the relation and cardinality among elements.

Figure 7 is an example of a CIML instance document for a simple bank system with one ATM, two BankConsortium and four Bank components. There are five interaction descriptions in this document: CheckPassword, getBankName, Withdraw, Deposit and Balance.

Future work: CIML uses by mediator connector for component composition at deployment phase. To investigate capability of CIML for design phase composition, where composite component can be deposit to a repository for further composition.

Fig. 5: XML Schema for CIML components composition structure description (Altova, 2007)
Fig. 6: The CIML XML schema definition
Fig. 7: Continued
Fig. 7: A CIML instance document for a small bank system

To develop a Component Composition GUI Tool in order to help software developers to compose software components by either point and click or drag and drop. The development will also help in creation physical weaving lines between software component interfaces; carry out data type validation and verification, providing contextual composition analysis for software developers.

CONCLUSION

CIML presented in this study is as an improvement of the attachment used by mediator connector proposed by Sanatnani et al. (2008). However, CIML is well-formed as a generic framework for describing the components and the interaction between them. Mediator connector is to encapsulate control to minimize coupling between components. We find out the important issue in composing components is the interaction between them (i.e., how they invoke each other’s methods). That’s why we consider that connector should be just a platform for connecting, where interactions must be considered as a separate issue. CIML is where the interaction of components will be described which later will be parsed by mediator connector.

CIML doesn’t support concurrency. The composition of components using a composition language should promise correct synchronization among components. Supporting concurrency and composition at design phase (i.e., composite components are component and they can be stored in a repository) if considered as future work to this research.

ACKNOWLEDGMENT

This research is supported by eScience Fund SF0704, Ministry of Science Technology and Innovation, Malaysia.
REFERENCES


