Toward Pure Componentware

Saleh Alhazbi[*], Aman Jantan[**]

[*]Computer Science Department, Qatar University , P.O Box 2713,Doha- Qatar
salhazbi@qu.edu.qa

** School Of Computer Science, Universiti Sains Malaysia, 11800, Penang, Malaysia
aman@cs.usm.my

ABSTRACT

Componentware seems to be a promising methodology for software development in order to cope with software complexity. With componentware, the software development is shifted from building every thing from scratch into just assembling existing components. Therefore, Components must be integrated through well-defined infrastructure. This paper presents a component model and a framework for composing component-based systems based on message-pattern interaction among the components.

Keywords **:** *Component-based system, message-based interaction pattern, connectors.*

1. INTRODUCTION

Reusability in software industry is considered the solution for software development complexity. Stressing the need of reusability in software development started with early structured programming languages in the 1970s. The program was divided into modules, and the concept of reusability was applied by using function libraries to implement the system in order to reduce the cost and increase the flexibility of the system. Next generation of software reusability was appeared with the object oriented development approach. After its conception at the end of the 1980s, the appealing concept of object-oriented framework has attracted attention from many researchers and software engineers. Although object oriented paradigm offers much support for reusability concept, it could not cope with open systems because it still follows traditional models for software development where the requirements are assumed stable. The third generation of reusability in software development appeared in the late 1990s when the interest in component-based development (CBD) had grown in both research community and industry. The major role of a component approach is to manage changes better and make the system more flexible for future modifications. In componentware approach, the whole software system is built by integrating pre-built, pre-tested components rather than implementing every part from scratch. These pre-built components might be developed locally or acquired from a third party (commercial off-the-shelf components (COTS)).

1.1 Component

Component are pieces of software that are wired to build a system, the best-known formal definition of software component is formed by Szyperski in [7]. It states “A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties”. Brown[3] defines a component as “an independently deliverable piece of functionality providing access to the services through interfaces.” Actually, there seems to exist significant overlap- might be confused- between object-oriented concepts and component-based principles. This maybe because both methodologies focus on utilizing software reusability to build systems with high reliability. In OOD, it is done through white box reuse where the code is available and inheritance is applied instead of rebuilding everything from scratch. The interactions between system parts accomplished by sending and receiving messages between the objects of the system. On the other hand, in CBD, the system is built by putting pieces of software together where the source code usually is unavailable “black box”. The interaction between components is achieved through well-defined interfaces [6]. Essentially, objects can not be deployed independently while componentware assumes integrating components developed separately, maybe from different vendors with different programming languages.

1.2 Component Models

Generally, component model defines the rules that must be obeyed by developers. It specifies, at an abstract level, the standards and principles enforced on software engineers who develop and use components [2, 9]. Practically, there are different models that supports component-based development such as Microsoft’s Component Object Model COM+, Sun’s Java Beans, J2EE, and Common Object Request Broker Architecture (CORBA). Because that many of those models are based on object-oriented paradigm, they still suffer of objects concept limitations. The principal problem with compnentware is how to wire components together. It is no longer sufficient that components just be integratable. They must be interoperable. Interoperability can be defined as the ability of two or

more components to communicate and cooperate together to provide system functionalities [9].

In this paper, we propose a component model that supports compnentware principles. It, also, presents our framework, Message-based Interaction Component-based System (MICS), that utilizes our component model. In MICS, system functionalities are accomplished as a result of components interaction through soft system bus. The concept in MICS is similar to that one of integrating hardware parts, which communicate through well-defined bus.

This paper is organized as follows. Section 2 presents MICS component model, section 3 describes the architecture of our framework. Section 4 explains simple implementation of our framework. Conclusions and future work directions are given in section 5.

2. MICS COMPONENT MODEL

In our model, components represent the essential part of the system. They are the locus of computation and the core providers of system functionalities. They merely services providers and consumers where the communication among them is facilitated by other entities called connectors. We should distinguish between two views of the software component: component type and component instance. The first one as a static piece of software that provides specific functionalities and the second view as an instance that has run-time existence and state.

In MICS model, components only interacts through their interfaces, either provide service to other components or require services from them. We use XML notations to describe component's interfaces, which can help during system composition for automatic check of compatibility between their interfaces. Any tow components can only communicate if they are syntactically compatible. Compatibility can be described as the ability of two objects to work properly together if connected, i.e. that all exchanged messages and data between them are understood by each other [8]. Figure 1 is an example of XML notations that describes a MICS component with its interfaces, which includes its provided and required services. This component Comp1 provides a service binary Search that searches an integer array and returns integer represent the index of searched element. The component requires a service sort to sort an array of integer. MICS components need connectors to interact with each other, which are defined during composition phase by the integrator. This separation between computations and communications offers loosely architecture. It supports concepts of componentware as the components being easy pluggable and replaceable.

Formally, component in our model can be defined as follows:

Definition1:*Component type is a tuple Ct=<Desc,Iinterf> where Desc represents the XML description of the component interface, Interf represents its interfaces where Interf= in* $\cup$ *out and In={incoming interfaces} , out={out-going interfaces}*

Defintion 2: *Component interfaces is a tuple Interf=<InInterf,OutInterf> where both InInterf and OutInterf are sets of methods M (operations), where each M=<Name,Ret,In>*
Name: represents the name of the method.
Ret: represents the return type of the method.
In : represents a set of input parameters.

```
<component>
<name> Comp1</name>
<provide>
        <service>
                <name> binary Search</name>
                <return>int</return>
                <arg>int[ ]</arg>
        </service>
</provide>
<required>
        <service>
                <name>sort</name>
                <return>int[ ]</return>
                <arg>int[ ]</arg>
        </service>
</required>
</component>
```

Figure 1. XML –based component description

3. FRAMEWORK

In this section, we explain our framework that utilizes our component model described in previous section. Our framework is based on message interaction style between components. Components send/receive messages through a soft bus to provide the functionalities of the system. Additionally, each component is hooked to the soft bus through a connector to facilitate message exchanges. Generally message-oriented pattern of interaction has the following advantages:

1. All dependencies are centralized and no explicit decencies between components which makes component integration easier [5].
2. It reduces the architecture complexity of the system which means it’s more maintainable and adaptable [1,4].
3. Message-based systems are more upgradeable and reconfigurable as new components can be added for satisfying new requirements without changing the basic system architecture [4]. Figure 2 depicts MICS architecture.

Formally, system in MICS framework can be defined as follows:

Definition 3: *System is a tuple S=<Cs,CNs,Sb,Ms> where*
Cs : is a set of MICS components.

CNs: is a set of connectors.
Sb: is the soft bus.
Ms: is set of messages that sent/received among system's components.

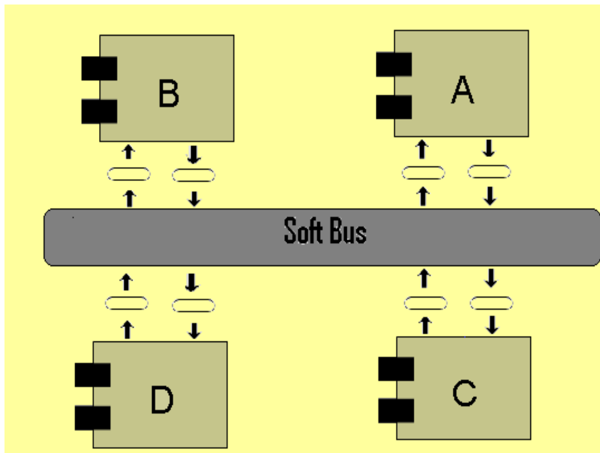

Figure2. Message-based Architecture in MICS Framework

Connectors

Connectors in our framework are not computation parts of the system, they facilitate components interaction. Each component in MICS communicates with other components in the system through connectors, which hook the component up to the bus. Each connector represents the gateway between the component and the bus. We have two types of connectors Out-port and In-port, Out-port connector masks the services provided by the component, therefore this connector has the same methods as the component behind it. The task of this type is to interpret incoming messages according, and call the service from the component. On the other hand, out-port connector represents the gateway for the service required by the component.

Soft bus

Soft bus in our framework is a special component that is responsible of tracking and identifying all components connected to it, so it routs messages from sender components to the target ones. It simulates the concept of using bus with hardware, so the components can be plugged in or out easily.

Messages

MICS framework has two types of messages: Request message (RQ), and Response message (RS). Every message contains two parts: a message part (such as service required, service arguments), and a control part (such as message ID, message type). The types of messages as follows:

1. Request message (RQ): this message is sent from a component to another asking for one of its provided services. The message is six tuple < Message type, Receiver, Service, no of arguments, arguments, sender>
2. Response Message(RS): this message is sent as a successful response to a previous request, it carries the result back to the sender of the request. This message format is five tuple <Message type, Receiver, Result, Sender>, even thought the service might not returns any result, an RS message should send back to the requester component. RS considered as acknowledgment message of finishing the process.

4. IMPLEMENTATION

We have prototyped our framework in java language where the main component is a class. Figure 3 illustrates the implementation of Bus concept in java

When a component requires a service, it calls general method in its out-port connector, where the connector forms that request as a RQ message and sends it through the bus. On the other side, the target in-port connector identifies the message is sent to it, interprets its fields, and call the required service with the parameters send with RQ. When it finishes the service, the result is sent back as RS.

```
public  class Bus {
private static List<OutPort> OutPorts=new
ArrayList<OutPort>();
private static List<InPort> InPorts=new
ArrayList<InPort>();
public static  void connectOutPort( OutPort l ) {
   OutPorts.add( l );  }
public static  void disConnectOutPort( OutPort l ) {
            OutPorts.remove(l );}
public static void connectInPort( InPort l ) {
            InPorts.add( l );  }
public static  void disConnectInPort( InPort l ) {
            InPorts.remove( l );}
public static  void sendRequest(RequestMessage m) {
    int receiver=m.getRecieverId();
            Iterator listeners = InPorts.iterator();
            while( listeners.hasNext())
            {
            InPort c=((InPort) listeners.next());
            if(c.getID()==receiver )
             c.receiveRequest(m );
            }
}
public static  void sendReply(ReplyMessage m) {
 int receiver=m.getRecieverId();
            Iterator listeners = OutPorts.iterator();
            while( listeners.hasNext())
            {
            OutPort c=((OutPort) listeners.next());
            if(c.getID()==receiver )
            c.receiveReply(m);
            }
}
```

Figure3 . Java-based Implementation of Soft Bus concept

We have built a simple prototype that generates random numbers and use binary search algorithm to look for a specific element in that array. Our example composed of three components: Main, Gen_Comp to generate an array of random integers, Sort_Com to sort

the array, and Search_Com to search the sorted array. Figure 4 depicts the architecture of our example.

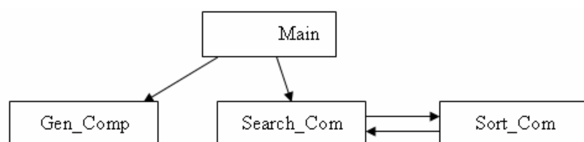

Figure 4 . Prototype of MICS Framework

5. Conclusion and Future work

In this paper, we present a framework (MICS) for component integration based on message-based interaction pattern where message exchange among system's components. The concept presented here through MICS is preliminary step toward fully pluggable components for building component-based systems with more maintainability. Moreover, this framework supports run-time updating as components can be plug in and out easily to the bus that routs messages among the components. Future work is needed to build visual tool in order to ease integration of components. Real application implementing with MICS framework will be good experience to evaluate system performance, to estimate overhead resulted of using indirection (connectors, bus) communication between components.

REFERENCES

[1] Alhazbi, S., "Measuring the Complexity of Component-based System Architecture". In Proceesdings of International Conference on Information and Communication Technologies: From Theory to Applications, 593-594,Syria April 2004.

[2] Bergner K., Rausch A., Sihling M., Vilbig A.. "Putting the parts togehter: Concepts, description techniques, and development process for componentware," In 33rd Hawaii International Conference on System Sciences volume 8, 2000.

[3] Brwan G., "Background information on CBD," SIGPC, vol.18,no.1, August 1997.

[4] Cheng J., "Soft System Bus as a Future Software Technology," Proc. 8th International Symposium on Future Software Technology, Xi'an, China, SEA, October 2004.

[5] Medvidovic N., Taylor R., "A framework for classifying and comparing architecture description languages," In M. Jazayeri and H. Schauer, editors, Proceedings of the Sixth European Software Engineering Conference (ESEC/FSE 97), pages 60–76. Springer–Verlag, 1997.

[6] Meijler T., Nierstrasz O., Beyond Objects: Components. In Cooperative Information Systems: Current Trends and Directions, M.P. Papazoglou, G. Schlageter (Ed.), Academic Press, 49-78, 1997.

[7] Szyperski C., Component Software: Beyond Object-Oriented Programming , Addison-Wesley, November 2002.

[8] Vallecillo A, Herandez J, and Troya J., "Component Interoperability," Tech. Rep. ITI-2000-37, Dept. de lenguajes Ciencias de la computación, University of Málaga, July 2000.

[9] Wegner P.," Interoperability," ACM Computing Surveys, vol. 28,no.1, pp.285-287,1996.