Abstract— Rapid advancements in Component based software Engineering has resulted in large number of component models (.NET, EJB, COM etc.) available. As a consequence of this advancement the number of reusable and ready to use components has grown as well. But this growth limits the reuse because the user writing an application in one component model is limited to use the components developed in the same model. Many components are being written; on the other hand many frameworks are also being developed. But there are compatibility issues among the frameworks and components. This fact motivates for such a framework where applications can be composed using diverse ready to use components written in different component model. In this paper presents a brief framework. In the future paper we will proposed a framework that allows diverse components to be added in a single application. It will provide an access mechanism where the framework does not need to know the interface of the component and it will allow the reuser to extend it without understanding the design or implementation details.

Keywords — Component Based Software Engineering, Component Composition, Heterogeneous Component Framework, Framework Instantiation.

I. INTRODUCTION

CompOnent-based software development is gaining recognition as the key technology for the construction of high quality, evolvable, large software systems in timely and affordable manner. In this new setting, interoperability is one of the essential issues, since it enables the composition of reusable heterogeneous components developed by different people, at different times, and possibly with different uses in mind. Currently most object and component platforms, such as Common Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), or Enterprise Java Beans (EJB) already provide the basic infrastructure for component interoperability at the lower levels, i.e., they sort out most of the “plumbing” issues. It is believed that reuse of software design and implementation has significant impact on productivity. In this scenario software frameworks [14] are considered as a definite way of improving the software quality and reducing the effort, cost and time. This is possible because the frameworks provide a vehicle for the reuse of the knowledge about the design and also the implementation that they encapsulate.

In the past few years component based software engineering has become very important for development of large scale enterprise level applications. Plenty of ready to use components are available to programmers for composing applications. On other hand several component models e.g. CCM (CORBA Communication Model), EJB [8] (Enterprise Java Beans), Java Beans [11], Fractal etc. have been introduced for development of components. The variety of component models limits the number of components that are available to the programmer for reuse in the application he is developing. For example, if the programmer is writing an application in EJB then he will seek ready to use components written only in EJB. A way is required to allow the composition of application with diverse components in order to let the programmers choose the best available component, without bothering about the underlying details of implementation of the component.

Secondly frameworks usually communicate with components using a predefined interface. This is a serious constraint since it requires the component to comply with the interface that the framework requires. At the time of writing application the programmer may discover that many ready to use components exist but they do not necessarily comply with the interface requirements of the framework. This makes them incompatible with the framework.

This requirement leads to a development of such a framework which allows developers to compose their application without bothering about the component models and interfaces of the component.

Only development of framework does not solve the whole problem. To reuse the design knowledge encapsulated in the framework, it needs to be extended to real systems. This
process of extending the framework to real system is called framework instantiation [1]. In this process the user reuses the partial implementation and extends it to incorporate the specific requirements of the application under development. It is not really easy to reuse the frameworks because of their implementation. For generality, frameworks have a very flexible architecture and design structure which is complex in nature and not easy to understand. This makes it difficult for developers to understand the rationale behind the design structure.

Endeavors have been made for developing powerful documentation to make the understanding process easy. But none of these techniques could solve the problem completely. This limitation leads to the process of automatic framework instantiation which helps the user of the framework to extend the framework without understanding the design rationale completely.

This paper is a sincere attempt to give introduction frameworks and a brief survey of different framework for composing applications using diverse components, with advantages and disadvantages.

II. RELATED WORK AND PROBLEMS
The problem in composing applications arises because of the diversity of components leading to:

Different Conventions and Standards: Different writing conventions and standards in different component models make them incompatible with each other. Also, the reuser might not be familiar with different models and their conventions and standards.

Diverse programming languages: Component models support a large variety of programming languages. For example COM components can be written in Visual Basic, Visual C++ or Visual J++ [13] whereas EJB can be written using Java. The reuser might not be familiar with all types of programming languages. This difference makes the components incompatible.

Diverse ways of deployment: Component models also provide specifications for deployment of applications. Different models have their own way of deployment. This makes a big difference while making them interoperable. For example EJBs are deployed in the Application Server (Specified in EJB Core Contracts and Specification) [10] whereas COM is deployed by registering in the Windows registry.

Different communication mechanism: Components may be standalone or distributed. Standalone components may communicate by pipes or inter process communication mechanism or simple message passing whereas distributed components communicate using TCP ports, SOAP or other standard protocol. This makes them isolated from each other.

Different types of Component: The framework needs the reference of the components to create the objects of the components, to refer to these objects as well as to manipulate the object. This requires the framework to have the knowledge about the Type (Class or interface or super class or super interface) of the component. The problem lies in the mechanism that the framework uses for accessing the objects. Frameworks use Types to refer to the components. The type of a component is described at the time of writing the framework. So the component written prior to the time of writing framework or the component which are generic and are not written keeping in mind that they are going to be plugged into such a framework become incompatible to the framework.

III. INTRODUCTION TO FRAMEWORKS
A framework is a collection of classes which are intended for providing the architecture’s implementation [14] i.e. managing layout, communication among components, managing control flow and providing some domain specific functionalities which framework reusers can reuse. Frameworks are intended for facilitating development, by letting the application developers to utilize more effort on accommodating application requirements rather than low level details of making the system work as a whole.

For example, a developer using any MVC framework like Sun Java Server Faces to develop an online library portal can focus on how books issue and return are being anticipated rather than bothering about driving the pages navigation according to output of business logic components. It is the job of the framework to provide communication and manage flow of control among different components. Apart from this it provides a way of component configuration, assembly and execution as customized by the reuser.

Generally a framework is made of frozen spots and hot spots.

Frozen spots are the fixed points of the framework they cannot be extended. These spots define architecture of a system i.e.

- They specify framework specific components which implement the architecture of a family of applications which can be developed using the framework.
- The spots do not change in any application which is built using the framework.
- Usually in case of Object Oriented framework these are represented in the form of abstract and concrete classes.

Hot spots are the application specific components, implementing the business function required by the application. These are very useful in customizing the
framework. Hot spots can be implemented in one of the following ways:

- Inheriting the framework classes.
- Implementing framework interfaces.
- Adding new classes

IV. FRAMEWORK INSTANTIATION

The reuser has to extend the framework into real application in order to reuse the framework; this process is called instantiation [1]. Frameworks are classified [14] in two categories according to the ways of instantiation shown in figure 1:

- White Box Frameworks
- Black Box Frameworks

White box frameworks: Instantiation of this type of framework is done by adding classes manually. These classes must meet the rules of hotspots of framework. The added classes can be introduced either by inheritance or composition. The disadvantage of this type of framework is that the application developer (i.e. the user of the framework) has to have sufficient knowledge about the internal details of the framework in order to use it productively.

Figure 1: Framework Instantiation

White box frameworks are also called architecture-driven. To instantiate such frameworks the reuser must understand the design rationale of the framework. It is clear from the name itself that the reusers have to look into the framework specific code, understand it and then write application specific classes which meets the constraints of the framework.

Black box frameworks: In contrast to the white box frameworks, the reuser does not need to look into the framework code to use the black box frameworks. The framework components are seen as black boxes to the reuser. Frameworks of this type can be instantiated using intermediate languages or configuration scripts or can be assisted by a GUI tool. While using the GUI tool, GUI tool is used to take the application configuration as input. The GUI then generates the classes and required source code. This approach does not entail reusers to understand the details of the framework implementation. These frameworks are easier to use in most of the cases. In our work we are using the black box instantiation approach. It is assumed that components given to framework are binary (executable components source code of which is not available) and will be used in black box manner.

V. COMPONENTS AND COMPONENT MODELS

Typically a component [7] is an independent unit of

- development,
- deployment, maintenance and replacement
- reuse

It is a package of business functions which are

- Cohesive,
- Having contractually specified interfaces,
- Loosely coupled and independent of other components.

Component models [7] can be seen as a set of standards and conventions. They provide guidelines for writing components, their composition and execution of system as a whole.

Component Models provide:

- Standards for component implementation, configuration and deployment.
- services and infrastructure to components such as:
  - a meta-information facility,
  - Lookup and Binding services.
  - Persistence and Transaction Management etc.

The component may have self contained implementation of the services it provides to the user or it may rely on the infrastructure where it is being deployed. Some components like Java Beans or COM have self contained implementation of the services whereas EJB [8], CORBA components rely on the application server or the container where they are being deployed.

Figure 2: Components and Container

Application Requirements

FIT

Class

White Box

Class

Black Box

Component

Component

Component

Container

Component

Component

Component

Component

Component

Component

Component

Component
The container shown in figure 2 is an environment for deploying the components, which provides a way of instantiating, requesting, broking, naming etc. to the client. On the other hand it provides various services like persistence, session management, transaction management, security, context etc. to the components so that they can perform their services efficiently.

VI. HETEROGENEOUS COMPONENT FRAMEWORK

With the introduction of component models, many ready to use reusable components have grown as well. As of today, developers have a large number of components to choose from. These components are diverse in nature; some of them are user interface implementations whereas others are implementing some complex business logic. These are well tested and are in use. Therefore, reusing these components is easier than developing new components. The lack of unified communication mechanism and common protocol among the components prevents the application developers from using the components. Component Models have originated from programming languages and architectural styles [12]. Thus, the way of operation, assembly, configuration and deployment varies among models. Due to this diversity the component composed in different component models are incompatible with each other. This incompatibility prevents the application developers from combining them in a single application. Heterogeneous component frameworks are intended to address this problem by allowing the application composition through the components from different component models.

VI. HETEROGENEOUS COMPONENT FRAMEWORK

With the introduction of component models, many ready to use reusable components have grown as well. As of today, developers have a large number of components to choose from. These components are diverse in nature; some of them are user interface implementations whereas others are implementing some complex business logic. These are well tested and are in use. Therefore, reusing these components is easier than developing new components. The lack of unified communication mechanism and common protocol among the components prevents the application developers from using the components. Component Models have originated from programming languages and architectural styles [12]. Thus, the way of operation, assembly, configuration and deployment varies among models. Due to this diversity the component composed in different component models are incompatible with each other. This incompatibility prevents the application developers from combining them in a single application. Heterogeneous component frameworks are intended to address this problem by allowing the application composition through the components from different component models.

Now we present a brief review of different proposed Heterogeneous Component Framework.

A. Vienna Component Framework
Johann Oberleitner et al [5] present Vienna Component Framework (VCF) which supports the interoperability of components across different component models and allows the user to compose components developed in different models.

![Figure 3: Architecture of VCF](image)

VCF has identified the following features of components:

- Lifecycle: This feature provides a way for creating and garbage collecting the instances of the component.
- Persistence: This feature allows the storage and retrieval of an instance of the component.
- Method: Methods are seen as objects with the feature itself being an object. They provide a way for calling the method and underlying details like parameters marshalling and returning the value.
- Property: This feature allows access to the attributes of an instance of the component.
- Event: Allows other components to be registered as event listener to act upon the events of the component. This also tells about the interface required for being event listener of this component.

B. Web services from generic components
Johann Oberleitner and Schahram Dustdar [2] present an approach for construction of web services from generic components. The paper presents a visual tool called component workbench for composition of components to build applications. The authors combine the components by means of Web Services. The web services are also regarded as a component models. First web services are created out of generic and diverse components and then they are coordinated to create applications where these services communicate using SOAP (Simple Object Access Protocol). The authors also present the ways of metadata discovery and dynamic calls to components in different component models.

C. UML–Framework Instantiation
Toacy C. Oliveira, et al [1], proposed an approach which is based on software processes to provide a systematic way for the development of the application shown in figure 4.
The authors felt the lack of a language to represent the frameworks. Unified Modeling Language (UML) has the ways to represent things which are needed in Object Oriented Modeling but does not have any way to represent a framework extension points. To overcome this shortcoming the paper introduces:

- UML-FI (UML–Framework Instantiation) [1]: It is an extension to the Unified Modeling Language which can accommodate the representation of things which are necessary to represent framework instantiation activities. Framework extension points are one of them.

- There was no structured language for representation of activities of framework instantiation, which can let users to represent the requirements for the instantiation environment. To overcome this, the paper uses RDL (Reuse Description Language) [6], a process language which has provision for the representation of activities needed to guide the environment about framework instantiation. The RDL script is the input given to instantiation environment which in turn generates classes and code.

- There was lack of techniques to verify the consistency of framework and to impose the constraints on the application which was developed using the framework. For example suppose there is a constraint that a particular class in a framework must be extended or any method of a class must be overridden to instantiate the framework. The provision for checking whether the RDL script contains instruction for the same or not is made in the instantiation environment proposed in the paper. The instantiation environment proposed is supported by a tool called xFIT (XML based Framework Instantiation Tool) [1].

The analysis of different approaches show that these approaches lack in following ways:

- The instantiation activities are to be written manually in RDL, which in turn must be assisted by a GUI tool.
- There is lack of many object oriented constructs like packages, class hierarchies etc.
- The approach introduced can be used only for the frameworks consisting of homogeneous composition.
- The approach does not consider data types.
- Basically text based approach for framework developer binds the instantiation in terms of cookbook recipes.

Marcilio Mendonca et al. [3] suggested an approach which is an extension to the process language based approach of [1]. The approach tries to enhance the shortcoming of earlier approach of being text based for both framework developer and reuser.

D. xFIT instantiation Framework

The approach introduces a process language which is combination of UML [15], RDL and uses design patterns extensively.

The paper has some predefined recopies which can be made using the framework. Framework developer uses the language which is a combination of above stated languages to specify the steps for adoption which are ultimately used by reuser of the framework by means of xFIT instantiation tool shown in figure 5. The tool uses Framework’s design specification in the form of class diagram and modifies it to incorporate the framework reuser’s application specific requirements and generates

- Customized Class diagram
- Stubs of classes
- Instantiation Steps
The framework reuser fills the stubs with application specific code to fit his requirements and the xFIT tool verifies the customization for satisfaction of constraints.

The enhancements in the paper were as follows:

- Extension to validate the correctness of design manipulation.
- Reuse of instantiation artifacts.

E. Automatic instantiation using generative programming

Vaclav Cechticky et al [4] present the approach for automatic instantiation using generative programming shown in figure 6. The work is supported by a GUI tool which allows the application developer to compose and assemble components. The approach uses domain specific language for formalizing and interpreting the requirements. The tool support in [4] uses customized java beans builder for the environment for composing the application out of components. Visual Proxy Beans based on Proxy of java is used for visual representation of components in the environment. The proxy is actually a representation of the component which acts as the component but it uses the actual components for serving the services requested. The diagram shows the generative approach for framework instantiation:

![Diagram showing generative approach for framework instantiation](image)

Figure 6: Generative approach for framework instantiation

To collect requirements it provides a GUI tool which translates the user clicks into domain specific language. It uses XML [17] for encoding information, XSLT [16] program for code generation and Java Bean Builder for component composition.

XML is used for decoupling the way components are created from the way they are configured. The figure 7 shows the xml code and corresponding instantiation code. XSLT are the style sheets which act as skeleton of code with some placeholders. These placeholders and their values are written in XML files in predefined format. The XSLT transformer program takes the XSLT style sheet and XML file with values of placeholder and generates the code correspondingly. The benefit of this approach is loose dependency on programs. The program is there for transformation; the style sheet contains guidelines for transformation of the data to be put in placeholders which is written in XML which is independent of any programming language or technology. The approach makes the system easier by providing a GUI tool which takes the configuration as input and generates the XML file accordingly. This is fed into the XSLT code generator for instantiation code generation. The figure shows the overview of the approach:

![Diagram showing code generation through OBS](image)

Figure 7: Code Generation through OBS

XSLT programs are used for code generation from encoded information. Framework components, when imported to composition environment, create overhead. This is accomplished by Visual Proxy for components. The paper is supported by a tool called OBS (On Board Software) Instantiation Tool.

VII. OPEN PROBLEMS IN HETEROGENEOUS COMPONENT FRAMEWORK BASED SYSTEM

- Commercially available Off-The-Shelf software is usually delivered as black box components with limited specification making it difficult to predict how the components behave under different conditions.
- There is a general lack of methods for mapping user requirements to component based architecture.
- Components are packaged and delivered in many different forms (example: function libraries, off-the-shelf applications and frameworks).
- Component framework offer varying features (example: component granularity, tailorability, platform support, distributed system support, interoperability).
- Most component integration processes suffer from inflexibility by a lack of component evaluation.
schemes. This problem is often compounded by a lack of interoperability standards between component frameworks and adequate vendor support.

- Most Commercially available Off-The-Shelf software is generally not tailorable or “plug and play”. Significant effort may be required to build wrappers and the “glue” between components in order to evolve the applications or tailor components to new situations. As the system evolves these wrappers must be maintained.

VIII. CONCLUSION

In this paper we have discussed about the basics of frameworks, the process of framework instantiation and categories, components, component models and finally the heterogeneous component framework. This section gives a summary of important terms used in the paper.

- Framework: Incomplete application implementing architecture, consisting of some concrete and some abstract classes and offers hotspot to add application specific components.

- Framework instantiation: Since frameworks are incomplete applications, the process of extending the framework in applications which meets user requirements is called framework instantiation.

- Component: Component is an independent and cohesive unit of development, deployment and maintenance.

- Component Models: It can be understood as a standard for developing and deploying components. Also, these models provide an architectural level infrastructure to components.

- Heterogeneous component Framework: A framework which allows the application developer to compose application with components built across component models.

Interoperability is one of the major challenges, particularly within component based software development environments, an approach in which prefabricated reusable software components from independent sources are assembled together to build applications. There are many aspects related to component interoperability, including syntactic agreements on method names, behavioral specifications of components, service access protocols, business domain knowledge and shared ontology, negotiation of Quality of Service and other non-functional properties. XML allows the flexible development of user-defined document types. It provides a robust, non-proprietary, persistent, and verifiable file format for the storage and transmission of text and data both on and off the web [18]. It can be customized to meet the users’ needs in many kinds of applications. Whereas CORBA, RMI, (D)COM and .NET Remoting try to adapt to the web, SOAP middleware ensures a native connectivity with it since it builds on HTTP, SMTP and FTP and exploits the XML web-friendly data format. Reasons noted for the success of SOAP are its native web architecture compliance, its modular design, its “simplicity”, its text-based model (in contrast to binary and not self-describing CORBA, RMI, (D)COM, .NET protocols), its error handling mechanism, its suitability for being the common message handling layer of web services, its standardization process and its support from major software editors. Enterprises are characterized by an organization networked through their information system in which all the elements have to interact. This results in an increasing dependence with regard to information technologies for interoperability. Corresponding multi-tier architecture information systems are today built over advanced EJB or .NET component frameworks, themselves relying on middleware technologies such as CORBA, RMI and (D)COM. Initially EJB technology is multi-system and monolingual (Java) while .NET technology is mono-system (Windows) and multi-language. CORBA Component Model aims at proposing a multi-system and multi-language technology. This paper has dealt in detail, the Interoperability issues pertaining to Component based software development.

After having reviewed some of the existing approaches, we shall now proceed to develop our framework and the technique for instantiation in the next paper.

REFERENCES


Mohan Prasad Banerjee is a MTech scholar in Department of Computer Science & Engineering of Thakral College of Technology Bhopal. His research interest includes Software Engineering, Machine Learning and Software Interoperability.

Rupali Soni is an Assistant Professor in Department of Computer Science & Engineering of Thakral College of Technology Bhopal. She has ten years of teaching experience. She has published many research papers in national and international journals. His research interest includes Software Engineering, Neural Network, Data mining and Algorithm Optimization.