A DOMAIN FRAMEWORK APPROACH OFFERING DEFAULT RELATIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ERSİN ERAY KARGI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

NOVEMBER 2005

Doç.Dr Ahmet Coşar	(CENG, METU)	
Doç.Dr. Ali Hikmet Doğru	(CENG, METU)	
Y.Doç.Dr. Halit Oğuztüzün	(CENG, METU)	
Dr. Osman Abul	(ASELSAN)	
Dr. Meltem Turhan Yöndem	(CENG, METU)	

ABSTRACT

A DOMAIN FRAMEWORK APPROACH OFFERING DEFAULT RELATIONS

Kargı, Ersin Eray

M. S., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Ali Hikmet Doğru

November 2005, 86 pages

In order to use components that are developed for a domain, domain knowledge is required. If the default relations in a domain are offered by a framework, this can be a starting point for the application engineer as an important kind of domain knowledge. A generic design for creating and saving a domain is implemented in this thesis. This approach starts with creating a domain from components and relations among these components. The relations and components are saved once and used several times. In addition, this generic design helps for code generation by using components. A framework for this design is implemented and applied for GIS domain. A basic code generation approach is also implemented in this framework for demonstration purposes. This framework can be used by domain engineers in order to create a domain and by application engineers to develop custom applications. It has the ability to offer default relations and helps creating new relations between components. Parameters and sequence of function calls can be defined by using a GUI. All the relations including default and user-defined ones can be used for code generation. COSECASE, which offers a tool for component-oriented design is extended with domain operations such as creating domain, saving domain, loading domain, and generating domain code. As the starting point, domain analysis for GIS domain is completed to define the domain.

Then the components that have been implemented for GIS domain and relations between these components are saved within the framework. Moreover, some basic applications are generated by using this framework in the GIS domain and sample domain generated for demonstration. Moreover, a sample domain is created to prove that our approach can be applied to any domain. The relations in this sample domain are saved in the framework and same basic applications are generated.

Keywords: Component Oriented Software Engineering, Component Orientation, Software Component, Domain Engineering, Application Engineering, Code Generation, Domain Creation

ÖZ

İLİŞKİLERİN OTOMATİK OLARAK ÖNERİLDİĞİ BİR ALAN ÇERÇEVESİ YAKLAŞIMI

Kargı, Ersin Eray

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Danışman: Doç. Dr. Ali Hikmet Doğru

Ekim 2005, 86 sayfa

Bir alana ait uygulama geliştirmek için, ilgili alana ait bilgi birikimi gerekmektedir. Eğer bir alandaki ilişkiler bir çerçeve tarafından önerilebilirse, bu uygulama mühendisleri için bir türden alan bilgileri içeren başlangıç noktası olabilir. Bir alanın kaydedilmesi için genel bir tasarım bu tez kapsamında geliştirilmiş ve gerçeklenmiştir. Bu yaklaşım bir alanın bileşenler ve bileşenlere ait ilişkilerin saklanması ile başlamaktadır. Bileşenler ve ilişkiler bir kere saklanmakta ve defalarca kullanılmaktadır. Ek olarak, bu genel yaklaşım bileşenleri kullanarak kod oluşturmaya yardım eder. Bu tasarıma ait bir çerçeve geliştirilmiş ve CBS alanına uygulanmıştır. Yaklaşımın başarısını göstermek için basit bir kod üretme yaklaşımı da bu çerçeve kapsamında geliştirilmiştir. Bu çerçeve alan mühendisleri tarafından bir alanı yaratmak için, uygulama mühendisleri tarafından da uygulama geliştirmek için kullanılabilir. Bu çerçeve ilişkilerin önerilmesi yeteneğine sahiptir ve ayrıca yeni ilişkilerin yaratılmasına yardım eder. Parametreler ve fonksiyonların sırası kullanıcı arayüzü kullanılarak tanımlanabilir. Önerilen ve kullanıcı tarafından oluşturulan tüm ilişkiler kod geliştirme için kullanılabilir. Bileşen tabanlı tasarım için bir araç sunan COSECASE, alan yaratma, alan yükleme, alan kaydetme ve alan kodu üretme gibi alan işlemleri ile genişletilmiştir. Başlangıç noktası olarak CBS alanında ihtiyaçları belirlemek için alan analizi yapılmıştır. Daha sonra bu

CBS alanı için gerçeklenmiş bileşenler ve bu bileşenler arasındaki ilişkiler çerçeve kullanılarak kaydedilmiştir. Ve CBS alanında bazı basit uygulamalar çerçeveyi kullanılarak geliştirilmiştir. Ek olarak tasarımın herhangi bir alana uygulanabilirliğini göstermek için yeni bir deneme alanı yaratılmış, bu alana ait ilişkiler çerçeve kullanılarak kaydedilmiş ve bazı örnek uygulamalar geliştirilmiştir.

Anahtar Kelimeler: Bileşen Yönelimli Yazılım Mühendisliği, Bileşen Yönelim, Yazılım Bileşeni, Alan Mühendisliği, Uygulama Mühendisliği, Kod Üretme, Alan Yaratma

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Assoc. Prof. Dr. Ali Hikmet Doğru, for his efforts and guidance. To my family, I offer sincere thanks for their emotional support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The goal of the thesis is to develop a framework, which offers default relations in the domain. Application engineers can use these default relations and extend them to produce a custom application. Although our demonstration area is GIS domain, another goal is to develop this framework as generic as possible. Therefore, a generic approach for this purpose will be improved. This generic approach allows to import components for a domain and to save the relations in this domain. Then these default relations can be used. A code generation part is also needed to prove that the relations can be used to generate an application.

Domain engineers are responsible for creating custom components and define the usages of these components. Application engineers use these components to create custom applications. It is hard to transfer information from domain engineers to application engineers. In our approach, after domain engineers have created a domain by introducing components and relations between these components, application engineers will be able to use default relations as a starting point. In addition, these default relations will give an idea about their usages.

In a domain, the interaction between components is obtained with the custom application. There exists no direct function call between components as these components can work alone. Therefore, domain engineers define the interaction with a possible custom application and components. For a complete definition of relations in a domain, the sequence of function calls must be set because all functions are called from one place. Furthermore parameters must be defined for code generation part.

The framework offers a generic approach for creation of and using domains. However, one limitation is that the components must be coded in Java. In our approach, the core components and default relations between custom applications and these components form a domain. This domain can be used to produce custom applications. Some applications may be developed by only selecting the components to be used. However, for most of the applications the default relations must be extended by application. In addition, generated code has some limitations and this may prevent the generation of the full application of this framework.

GIS domain is chosen as the demonstration domain. For military applications, GIS plays a key role. Nearly all of the applications have a GIS part. So by domain engineering and domain implementation of GIS domain a great reuse can be obtained. GIS part uses a Common of The Shelf (COTS) Component named "Map Objects for Java". Domain implementation for the GIS domain is out of our scope. We principally concentrate on domain analysis of the GIS domain and offer default relationships for the GIS domain.

To prove that this framework approach can be applied to any domain, another domain other than GIS domain is needed. So a sample domain is created.

1.1. Reuse in Software Development

After the term *software engineering* was coined the problem of reuse in software development started. Beginning with the first software developed, same source codes are rewritten by different software developers for several times. Applying reuse in software development, great savings could be attained. However, the potential benefits of reuse have never been fully realized [1]. A good reuse can benefit in quality, reliability, cost, time and productivity [2]. Also by reusing program parts, such as components, a specialist can develop software that is more efficient.

The problem can be viewed in analysis, design and implementation steps of the software life cycle. Since the cost efficiency of solving problems is the maximum in the earlier steps, the best benefit can be obtained in the analysis step. Clements defines a software product line as "a set of software intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment of mission and that are developed from a common set of core assets in a prescribed way". This definition makes the sense that a product line can use many common objects. We can think of a software product line as a set of systems, having similar requirements. However, with the traditional method all software products have their own product lines. New methods like domain engineering offer a common product line for all products in a specific domain. Therefore, *Domain engineering* offers great reuse possibilities in a selected domain for all of the steps of the software product line.

There are two types of reuse in software development. The first one is horizontal reuse and refers to software components used in different applications [2]. This type of reuse mainly concentrates on code reuse. Envisioned library of components like, hash table classes, standard math routines and GUI libraries are examples for this type of reuse. This constitutes the basic software code reuse and can be named as function reuse. In addition, Commercial Off-The-Shelf (COTS) use can be referred as horizontal reuse. COTS have their own test reports and documentation. Therefore, they are more complex then function reuse and can be named as system/sub-system reuse. It is much more advantageous to reuse a design instead of reusing a component. This type of reuse is vertical reuse. The idea is reuse of domains; a specific system area that can be used by different applications with similar functionality. This newly spawned discipline is called domain engineering and will be discussed in the next section.

Today classic approach to software reuse is ad hoc reuse. Nevertheless, this type of reuse does not fulfil the requirements of large-scaled software. Most of the software today cost millions of dollars. Therefore, the new approaches like domain engineering are getting importance against classical approach. However, domain engineering has a cost. With a reuse process, every new system can be built from a set of core assets. However, this approach introduces new difficulties for management such as maintaining product line including core assets, upgrading the product line according to customer requirements, implementing a methodology for analysis, design, implementation, and test phases. In order to deal with these problems a very strong organizational structure is needed.

Against all of the difficulties, software reuse has success stories growing in the overall world. High-quality and low-cost products can be achieved at the end of a successful reusable product line. Despite the initial overhead, software reuse worth.

1.2. Motivation for Domain Engineering

As described before, a domain is an area of interest for software development. In addition, analyzing and implementing a chosen domain spawns domain engineering. Domain engineering is essentially used by the organizations, which have product lines to produce similar products repeatedly. The right thing for such organization is to adopt reuse as part of a systematic process improvement initiative. For domain engineering, domain must be chosen carefully. A domain must satisfy some properties. First property is similarity that means applications developed in the domain have some commonalities and variability. If the domain is so big then the variability grows up to a limit that domain cannot be implemented. The second property is adaptability. It can be summarized as a characteristic set of deferred decisions that distinguish among the members of a family.

We have chosen GIS as the domain four our work. Since our area of interest is mostly defence applications and all the applications need a GIS part, a great benefit can be obtained by reuse in the domain. Another property of the GIS domain is its being a static domain. Therefore, it means the requirements for this domain are nearly complemented. This property reduces the upgrading cost of this domain. After a successful domain analysis, this domain can be fully implemented for future software product lines. Another motivation for domain engineering is its being the most commonly used technique for application development tools.

1.3. Motivation for using COSEML and COSECASE Tool

For component oriented modelling, Component Oriented Software Engineering Modelling Language (COSEML) and COSECASE [13] are good alternatives. Since component-oriented tools are not widely used, COSECASE tool is one the best despite its deficiencies.

COSECASE is enhanced to save the default relationships in a given domain and code generation by using these relationships. This produces a user-friendly domain implementation framework. The components that are implemented before are loaded and the relationships between them are saved. After loading the domain, COSECASE allows extending these relations and code generation. Parameter definition and sequence indicators are also implemented for COSECASE's general use.

1.4. Organization of the Thesis

The focus of the thesis is to save default relationships such as event, method, composition links with the required parameters, and function call sequences in a domain and display them in a framework for easy code generation. Code generation is not the main part of the thesis but it is implemented to show that this approach can help automatic code generation. Moreover, domain analysis is an important phase of the thesis. After a successful analysis, the default relationships in the domain can be improved.

Nevertheless, GIS domain is chosen for demonstration purpose in this thesis. The technique can be applied to any given domain. To prove this, a simple domain other than GIS, containing only two components is created. The technique is easy; components are loaded, relations are defined and the domain is saved. Then if the saved domain is loaded, the relations and the components are ready for use. Then an easy code generation can be done. For creating and loading a domain, COSECASE must be used. Beyond this introductory chapter, the thesis is organized as follows: In Chapter 2, necessary background on software reusability and components, component-based methodology, domain engineering, and automatic code generation problems are included. Chapter 3 describes the Domain Analysis process applied to the GIS domain. Domain Analysis is conducted with a feature model of the domain as suggested by the Feature Oriented Domain Analysis (FODA) methodology. Domain implementation part is not covered in this thesis. In chapter 4, the framework structure is explained and demonstrations over the GIS domain and newly created sample domain are illustrated. Chapter 5 presents a conclusion and discuss about further works.

CHAPTER 2

BACKGROUND

2.1. Software Components and Component Based Software Engineering

> "*Software componentry is a field of study within software engineering. It builds on prior of software objects, software architectures, software frameworks and software design patterns, and the extensive theory of object oriented programming and object oriented design of all these. It claims that software components, like the idea of a hardware component used e.g. in telecommunication, can be ultimately made interchangeable and reliable [3].*"

With the enlargement of open applications, object-oriented programming techniques advanced to a reliable, reusable technique called component-oriented programming. Although component-oriented programming technique is old, its importance is realized a short time ago. Today's large-scale projects costs too much with the standard object-oriented technique since reuse that can be obtained by object-oriented technique are limited. By designing and implementing reliable, reusable parts the gain can be obtained from these projects cannot be underestimated.

> "*A component is encapsulated, distributable, and executable piece of software that provides and receives services through well-defined interfaces [9].*"

So we can think of a component as a piece of software, which interacts with outer world by using its interfaces. With components, software can be developed by composing the correct components together. The component is available in a black-box paradigm. Therefore, no one using components is aware of the details of the components; the only thing that the user is interested is the functions served by the interfaces of the component. Components hide implementation, conform to interfaces, and encapsulate data, just as classes do in object-oriented languages [2]. The usage of a component is shown in Figure 2.1.

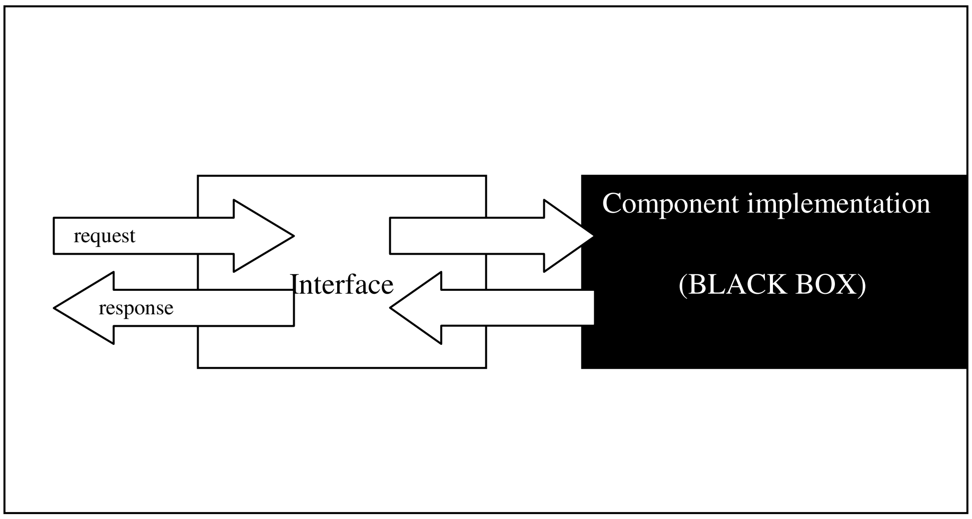


Figure 2.1 Usage of components

Since components can be used by multiple software projects, they must be well tested. This makes the software using the components more reliable. Because the parts of software are tested individually, they do not need to be tested again. Of course, this is a cost saving property for projects. Of course, components need an initial pay for development. Components must be developed by analysing for a wide use. If it is designed for a very specific process, than it will not have a general use. Therefore, the analysis and design of a component are very important for its life. Well-designed components surely will have a wide use and this will increase the benefit gained from the component orientation.

Present object-oriented methodologies, lack addressing the design of reusable frameworks. To allow reuse in large-scale software projects, evolution of a new software life cycle is needed [5].

Of course, component-oriented methodologies have many problems to deal with. First problem is the lack of analysis, design and coding tools suitable with the component-oriented approach. Today object oriented technology is widely used and all the tools are compatible with object-oriented methodologies. Second problem is the lack of a definite methodology for component-oriented approach to software development life cycle. Ongoing research projects all over the world are interested with these problems. In addition, absence of component-oriented metrics is another problem on the way to component-oriented software development methodologies.

For software engineering and distributed systems, Component-based programming has an importance. There are some work areas for component-based programming, which are subfields of software architecture. These topics are [4].

- software configuration managing
- configurable and contemplative distributed systems

There are different component models such as MG's CCM, Sun's EJB, or Microsoft COM. Components described in these models can be understood as *"units of software configuration, units of dynamic system configuration, units of deployment, units of distribution and mobility, etc. [4]"*. Despite these different work areas, the software engineering discipline is in need of highly flexible, highly dynamic, and mixed distributed environments. The failure of the different component models to meet several important requirements is identified by Van Hoek [23] for the analysis of component-based system models. In particular, today's component models insufficiently meet the requirement of an ideal component model.

2.2. Application Development Tools

Today object-oriented programming languages are generally used for developing applications. This approach is not sufficient to offer a pointable improvement in programmer’s productivity [4]. Therefore, component-oriented approach gains importance over object-oriented approach. The main problem to be solved in component-oriented programming is to develop reusable component sets and to minimize the effort for application development. This points us simply to an architecture where applications are developed by just uniting the components that are developed and tested before.

There are two key concepts to understand the idea behind application development tools. These concepts are;

- *“Application Engineering is the activity of abstracting the domain knowledge for selected application domains, developing reusable software components to address these domains, and encapsulating this knowledge into generic application frames”;*

- *“Application Development is the activity of instantiating a specific application from a generic application frames to meet some particular requirements”.*

The activity needed in here is to understand the application domain and the mechanisms to compose software components. The return on assets should take place during application development. Applications developed by this technique will be more robust and reliable since being developed by using fully tested and fully revised components. The efficiency of application development process is strongly related to application engineering techniques used and the software that offers reuse. Therefore, we can think of application engineering as a living process. It will always improve itself with new uses. In software development, the word says, “nothing is error-free”. By using, the developed application developed for a sufficient period errors will be corrected and the application will be in an ideal state. By application engineering, the components used will be improved with every use.The key idea behind application engineering is, reuse is gained by design. Therefore, a good design and analysis is very important in application development tools. In addition, the environment must support and use the generic application frames structure. These generic frames drive the application development process by means of pre-packaged requirements models, application designs and software components.

2.3. Domain Engineering (DE)

Today’s software product lines are used to develop similar projects. Organizations have their own libraries to speed-up their software-product line and organizations improve themselves in a known area such as military applications, GIS applications, document management applications, etc. This definition supports domain and domain engineering ideas.

“Domain engineering is a systematic process to provide common core architecture for these applications [4].”

“A domain can be defined by a set of problems or functions that applications in that domain can solve and has the parts;

- *collection of problems (problem domain)*
- *collection of applications (solution domain)*
- *area of knowledge with common terminology. [17]”*

The purpose of domain engineering is to provide reuse in a product line that is used to develop similar applications. Phases of domain engineering are shown in Figure 2.2. Domain engineering process can be applied to existing systems or newly evolving systems. Domain engineering and domain analysis are sometimes distinguished. However, domain analysis is a part of domain engineering. Domain engineering is the process of;

- defining the scope (domain definition)
- analyzing the domain (domain analysis)
- specifying the structure (domain architecture development)
- building the components (software code)

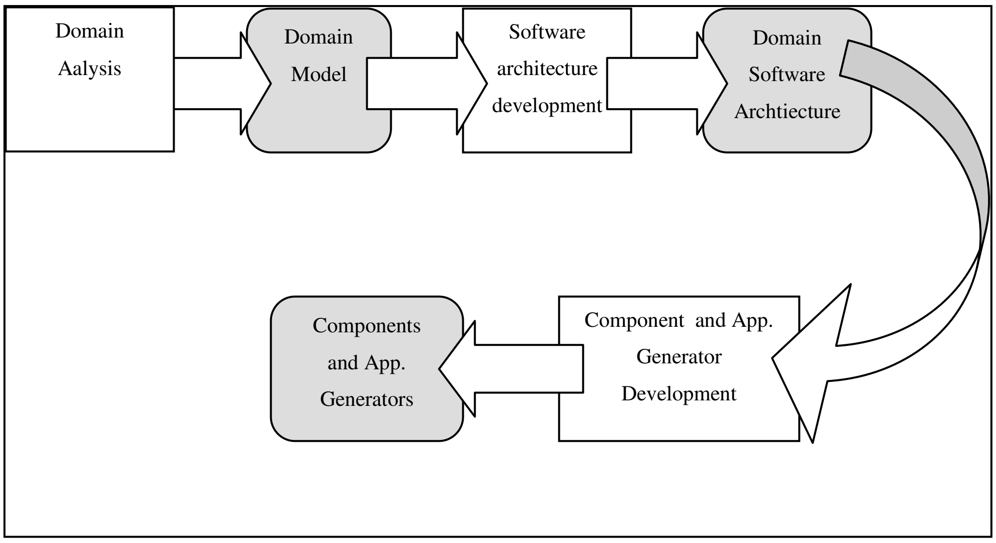

Domain Aalysis
Domain Model
Software architecture development
Domain Software Archtiecture
Component and App. Generator Development
Components and App. Generators

Figure 2.2 Phases of Domain Engineering

Determining the borders of domain is the key activity in domain engineering. Domain must be selected carefully. Because the success of domain engineering is fully related with the borders of the domain, the requirements, which are inside and outside the domain, must be represented first. To determine the domain first the area of interest must be determined and checked for common requirements. However, finding domain boundaries is not an easy process. This makes the need for an experienced domain analyst.

A domain can be seen as real world or can be seen as a set of systems. The first approach is mostly used for object-oriented context. It has the advantage of a reusable domain model. Because the model is derived without care of different applications, the concepts will be stable. The requirements for the domain can change but it will not affect the model much. The second idea is suitable for component-oriented context. The different applications in the selected domain must be considered. It is mostly interested in application families. The border of the domain is determined with the similarities of the domain. It is also suitable for application engineering, which will be described later. The first approach is mostly used for single-application development and the second approach is suitable for software product lines.

There are two types of domains:

- Vertical domains,
- Horizontal domains.

In vertical domains, area of interest is a business area. Existing systems are examined fully. Military systems, hotel reservation systems can be given as examples. In vertical domains, the selected domains cover the whole application area. The software product line can be applied to the selected domain. Engineering of only one domain is enough to have a software product line in the given area. Nevertheless, for huge software development product lines it can be hard to implement one domain. In this case, although the application area is strict the lateral domains will be needed to reduce the cost of management.

Horizontal domains have smaller borders than vertical domains. Parts of applications correspond to domains. Workflow systems, GUI (graphical user interface) libraries can be given as examples for horizontal domains.

Vertical domain is suitable for organizations that develop similar applications. Domain will cover the application area. However, horizontal domains are suitable for the organizations that have different software product lines. This type of organizations can reuse at the level of similarities of their product lines. So part of applications can be developed by domain engineering and the product lines will divide in sub product lines to reduce the cost and reuse the common components.

Domain engineering is generally described in three phases: domain analysis, domain design, and domain implementation [16, 17]. However, according to some sources, domain engineering can be explained in two phases, namely domain analysis and domain design.

2.3.1. Domain Analysis

"Domain analysis (first introduced in the 1980s) is an activity within domain engineering and is the process by which information used in developing systems in a domain is identified, captured, and organized with the purpose of making it reusable when creating new systems [18]."

As domain analysis is the first part of domain engineering, its purpose is to outline the requirements needed to implement the domain. It is similar with traditional software development cycle's analysis part. However, the interest is the selected domain. By outlining the commonalities and the differences of systems within a domain, domain analysis mainly focuses on a systematic and large-scale reuse. This process results efficient development and easy maintenance for selected domain. After the domain is selected and the borders are ruled, the similarities of applications in the selected domains are covered. Domain analysis can be done for single-system engineering or product-line engineering. After completed domain analysis, domain model is sketched. The ingredients of domain models are different from references to references. Nevertheless, mostly domain definition, domain requirements, domain dictionary, concept modelling and feature modelling can be counted as ingredients of a domain model.

First step in domain analysis is to find the boundaries of the domain and is called domain scoping. In this process, example applications in the interested domain are examined. Sometimes applications, which are out of domain, are examined to find out the requirements out of the domain. Some references divide scoping into three phases as domain scoping, product line scoping, and asset scoping. As understood from the name, domain scoping is used for domain engineering. Product line scoping is used to find out the boundaries of the software product line and is related with application engineering. Finally, asset scoping is used to scope the components in a domain. Requirement document and domain dictionary are prepared after scoping process. Domain dictionary is used to have a common language for the domain. The other documents and implementation are completed using this dictionary.

Characteristics outlined from requirements for a domain and that are reusable among individual applications are called features [17]. A feature is a characteristic of a system. Feature models are prepared to understand the relationship between features. Features can be mandatory, optional or alternative as used in FODA (Feature Oriented Domain Analysis). After extracting the features in a group of application or domain, the analysis proceeds to find similarities and differences between applications. Once the similarities and differences are outlined, domain model can be prepared. Some features may not be implemented due to rare use. Nevertheless, for an optimum analysis all the features that can be used in an application for that domain's software product line must be implemented. After completing domain engineering it is hard to add new features to the domain since the process must go backward in domain engineering (from implementation to analysis).

Feature diagram is outlined to the find the features in a domain. In a feature diagram, the top item is an application having several properties called features. All features can have sub features and so on. Feature diagram is a part of domain model. According to selected domain, different outputs can be prepared for domain analysis.

> *"For example, interactive systems may require use cases and scenarios. In addition, data-centric applications can be more easily described by entity-relationship diagrams or object diagrams. Special properties such as real-time support, distribution, and fault tolerance may require special modelling techniques [17]. "*

Domain model is the general name used for outputs of domain analysis. Therefore, every output for understanding domain easily can be prepared inside domain model. These outputs help the domain implementation.

2.3.2. Domain Design

Domain design is the second step for domain engineering. It is the general design of family of applications. The inputs of domain design are domain model and software/system architecture. Using generic design techniques, coordination models, portioning strategies, and design specification design model is extracted.

By using the domain model completed after domain analysis, domain design is documented for an easy domain implementation. The core architecture prepared must be generic enough to reply to requirements of any application in selected domain. The decision of which features or core assets will be implemented is given in this part. The decision belongs to, which components have a general use and which components are special parts of applications in the domain. After deciding the features to be implemented, production plan is prepared. Production plan contains descriptions of the systems with their interfaces and serves as a guideline for the assembling process of the components [17]. The key difference between the domain design and the domain model is the knowledge gained for software system reuse. The potential risks concerning the architecture also outlined in this process. In addition, the architecture are outlined from design model can be used for the software product line.

2.3.3. Domain Implementation

Domain implementation is the final process of domain engineering. Using compiler tools, coding standards, design models, and application generators, reusable components are completed. Companies use their core assets, COTS or standard libraries to achieve the goals. Reusable components that are created are catalogued into a component library for software product line. Management and maintenance of a repository of reusable components also play an important role in domain implementation [19].

The knowledge gained by domain engineering should be maintained and updated all the time according to new applications and new requirements. In addition, the feedback from users or application engineers must be reflected to domain implementation. Domain model can never be completed, since there are always new requirements.

2.3.4. FODA

The Feature-Oriented Domain Analysis (FODA) methodology resulted from an in-depth study of other domain analysis approaches. Successful applications of various methodologies pointed towards approaches that focused on the process and products of domain analysis.

FODA is a technique of domain analysis where the core aspects are features. Features cover common and uncommon aspects among a family of applications. There are two key concepts in FODA. One is context analysis which points to defining the extent (or bounds) of a domain for analysis and the other one is domain modelling providing a description of the problem space in the domain that is addressed by software. In context analysis, operating environments and standards used to output the context model, which consists of structure diagram and context diagram. Domain modelling consists of three distinct parts; feature analysis, information analysis, and operational analysis. Feature analysis uses features and context model to achieve feature model. In information analysis application, domain knowledge is used to get the information model. Finally, operational analysis takes domain technology, context model, features model, information model, and requirements to output operational model, which consists of behaviour and functionality.

FODA is currently a part of the in-based approach of domain engineering [4]. This approach is used both domain engineering and application engineering. In addition, FODA is further extended into FORM (Feature-Oriented Reuse Method) to include also software design and implementation phases

2.3.5. Domain Engineering versus OOA/D methods

The classical approach for software engineering is object-oriented analysis and design (OOA/D) methods. Unfortunately, object oriented methodology is suitable for single system development. Nowadays the agreed idea is that techniques and methodologies for reuse must be invested. Domain engineering is the best alternative offering reuse. Domain engineering can be applied to object oriented methodologies but this limits the reuse that can be obtained by domain engineering.

2.3.6. Application Engineering

For product line engineering, software engineering is divided into two different processes: domain engineering and application engineering. These two processes use their inputs and outs to obtain a fully functional software product line. Domain engineering is used to obtain abstract design, core components and application development tools. Then application engineering is used for a specific application. The focus of application engineering is a single system whereas the focus of domain engineering is on multiple related systems within a domain. The abstract design model is used to model the specific application by using new requirements. Then this model is used with core components and application development tools to accomplish the application. The new requirements, error reports and other artefacts are used for domain engineering to revising the domain model and core components.

The phases of application engineering are called requirements engineering, design analysis, and integration and testing. Figure 2.3 depicts such a process.

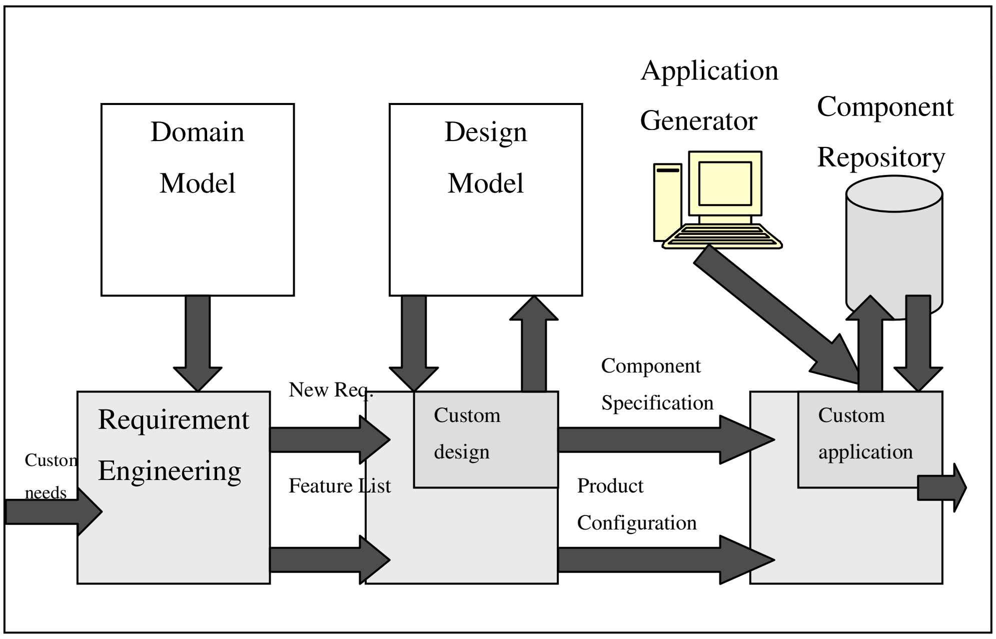


Figure 2.3 Interactions Between Domain Engineering and Application Engineering

In requirement, engineering part domain model is used to satisfy the customer needs. The requirements for customer needs are decided. Moreover, features to be used are exploited from domain model. By using requirements and feature list, a custom design is obtained for specific application. For custom design, generic design model obtained from design analysis of domain engineering is used. By using application generators and component library, system implementation implements one system, while domain implementation produces reusable components, core architecture, and production process.

2.4. COSE Modelling Language (COSEML) and COSECASE

COSEML is a component oriented modelling language and COSECASE is a tool implemented for COSEML. The user interface part of this thesis is implemented by extending the COSECASE tool. Graphical symbols used in COSEML are shown in Figure 2.4. Table 1 presents all graphical symbols in COSEML with further details.

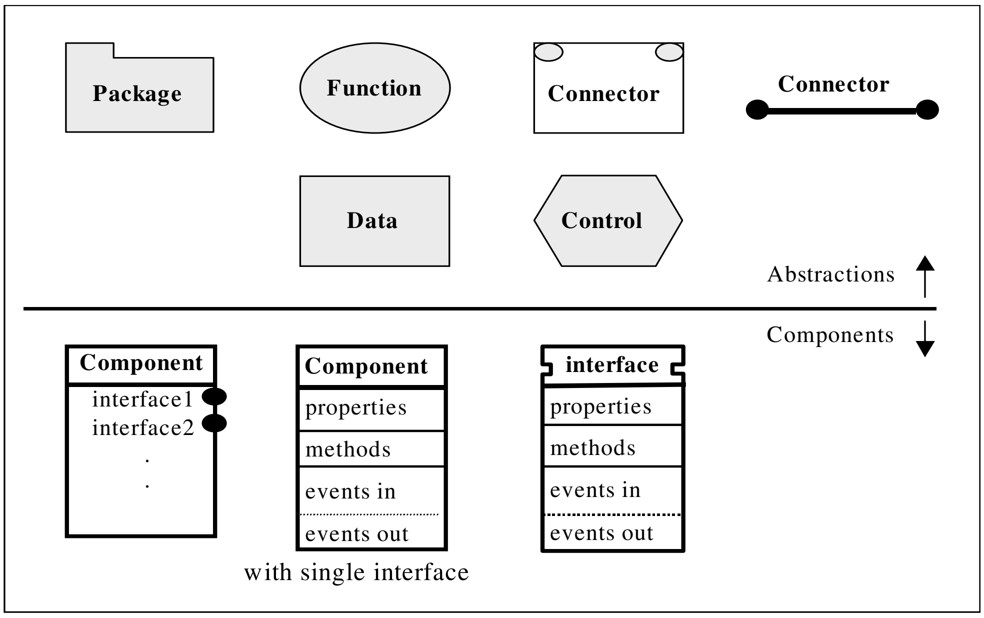
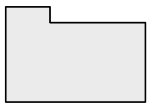
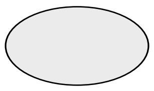
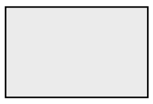
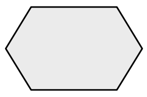
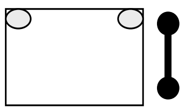
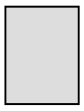
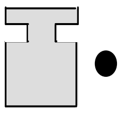
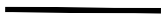
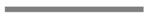
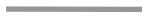
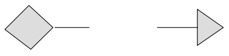


Figure 2.4 Graphical symbols in COSEML

Table 1 COSEML symbols and their meanings.

Symbol	Explanation
	**Package:** Package is for organizing part-whole relations. A container that wraps entities and functions etc. at a decomposition node. Can contain further package, data, function, and control elements. Also can own one port(s) of one or more connectors. Can be implemented by a component. The contained elements are within the scope of a package; they do not need connectors for intra-package communication.
	**Function:** Function abstractions represent a high-level function. Can contain further function, data, and package elements. Can own connector ports. Can be implemented by a component.
	**Data:** Data abstractions represent a system-level entity. Can contain further data, function, and package elements. Can own connector ports. Has its internal operations. Can be implemented by a component.
	**Control:** Control abstractions correspond to a state machine within a package. Meant for managing the event traffic at the package boundary, to affect state transitions, as well as triggering other events. Can be represented by a component.
	**Connectors:** Connectors represent data and control flows across the system modules. Cannot be contained in one module because two ports will be used by different modules. Ports correspond to interfaces at components level.
	**Component:** A Component corresponds to the existing implemented component codes. Contains one or more interfaces. Can contain other components. Can represent package, data, function, or control abstraction.
	**Interface:** An Interface is the connection point of a Component. Services requested from a component have to be invoked through this interface. A port on a connector plugs into an interface.
	**Represents Relation:** A Represents relation indicates that an abstraction will be implemented by a component.
	**Event Link:** An Event link is connected between the output event of one interface and the input event of another. The destination end can have arrows corresponding to the synchronization type.
	**Method Link:** A Method link is connected between two interfaces to represent a method call. Arrow indicates message direction.
	**Composition and Inheritance Relation:** UML class diagram relations are utilized. Diamond: Composition, Triangle: Inheritance.

CHAPTER 3

DOMAIN ENGINEERING IN GEOGRAPHICAL INFORMATION SYSTEMS (GIS) DOMAIN

This thesis offers default relationships between components in a domain for easy application development. To realize this, the components should be uncovered and the relations between these core assets should be outlined. Therefore, domain analysis is strictly required to understand the needs and to outline the relationships in the domain. Although another key point in this thesis is to develop a generic application development tool, GIS domain is dealt with as a case study.

Domain modelling and domain implementation are not parts of this thesis. These two processes for the GIS domain are completed in [8]. Requirements analysis for some defence projects, outlining feature trees and feature charts have been completed for this work.

The next part of the work is to develop a generic application developer for the GIS domain by using components prepared in [8]. In this chapter, steps in domain analysis will be discussed.

3.1. Domain Analysis for GIS Domain

An important step in domain analysis is to outline the customer (user/developer)'s needs. The organization that is inspected, works on defence projects area. During the analysis step, a general approach is used. All the features that must be found in general GIS domain are outlined. However, features not needed for the company are deleted.

3.1.1. Context Analysis for GIS Domain

The purpose in context analysis phase of domain analysis is to find the boundaries of the domain. In order to understand the scope of a domain, basic knowledge about the higher-level domains of which the target domain is a part must be collected. In addition, understanding the usage of domain and determining how the target domain is used by several applications are important.

A general context analysis report is prepared to find the boundaries of GIS domain. The results achieved are summarized below:

- A basic GIS application must cover simple functions like pan, zoom, extend, show map, distance analysis, and some 3D functions such as visibility and profile analysis.

- The usage of GIS software in military applications extends beyond the displaying of maps and other simple functions. Such software is also used for displaying information that comes from sensors, radars or other units in the battle area. Therefore, a GIS application must have interfaces to accept some predefined data into system and display this data. While displaying data, near real time constraints are sometimes enforced by the applications. In addition, some terrain analysis is needed. In addition, to achieve fire capabilities in the battlefield the system uses some vector data in the map. In addition, visibility analysis and profile analysis are involved. GIS application must serve this information to outside and must have the ability of displaying some predefined symbols and geometries. In addition, integration with GPS plays a chief role for GIS applications used in command and control systems.

- In addition, some general properties must be served by GIS applications like saving data in a relational database such as MSSQL or Oracle, porting application to different platforms, and integration with different GIS applications by predefined interfaces.

- Some GIS applications offer functionality on web, like serving maps and manipulating geographical data. Although this functionality is not needed by the military applications, this ability is concerned to find the boundaries of a general GIS application.

3.1.2. Context diagram

After discovering the boundaries of a domain, the structure context diagram is outlined.

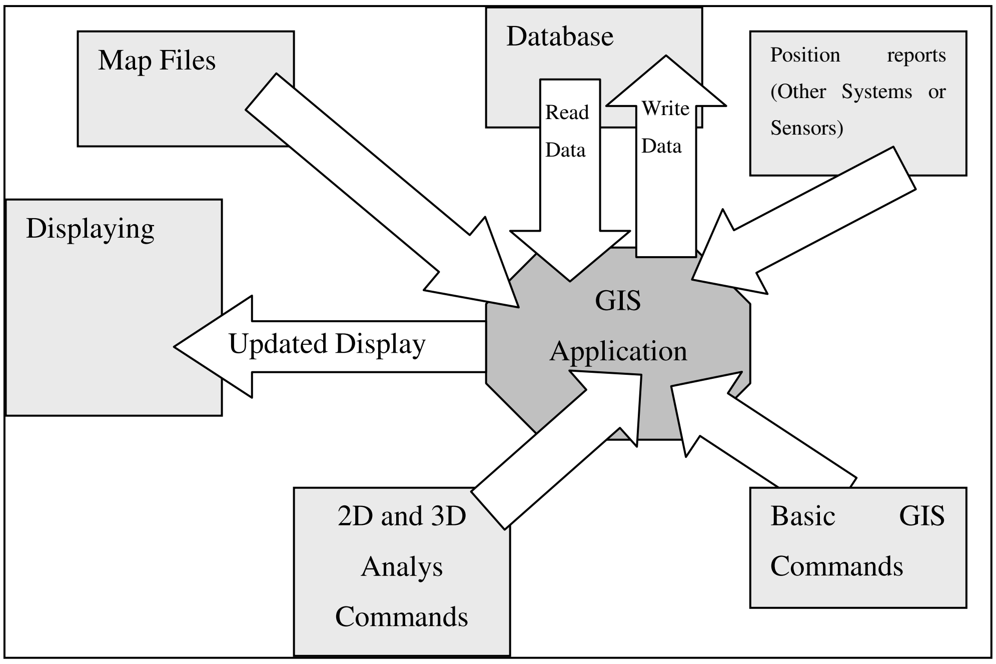


Figure 3.1 Context Diagram of GIS Domain

3.1.3. Requirement Document

After completing context diagram to show the interfaces of the GIS domain with outer world, the requirements document is prepared. While writing these requirements the general needs of military applications are used. These requirements are:

1. The application must be portable to different operating systems. Therefore, the language selected to implement the domain must satisfy platform independency. Java and C++ languages are compared and java is selected since COSECASE tool is coded in java.
2. Raster and vector data can be shown by the application.
3. Raster maps must be shown automatically according to the current scale of the map.
4. More than one raster or vector layer can be shown on the map at the same time. The ability of adding and removing layers must be supported by the application.
5. A symbology layer must be included to show custom symbols for applications. A symbol can be a font, a geographical layer, or a picture.
6. Custom GIS operations like, zoom in/zoom out, pan, extend, and scale must be achieved.
7. Distance between two points can be calculated.
8. 3D analysis like visibility analysis, and profile analysis can be achieved.
9. Ability of conversion between Universal *Transversal Merkator (UTM) and Geographic* coordinate systems must be covered.

10. Getting coordinates and altitude of a selected point on the screen must be covered.

11. Getting and setting extent of the map: Extent of the map can be set and accessed by the applications.

3.1.4. Feature Table

By using the requirements given below, a feature list is prepared and then this list is used in a feature chart to find similarities and differences among different military applications that use the GIS domain. Most of the projects are classified so they are not revealed in detail here. Feature table is represented in **Table 2.**

Table 2 Feature Table Generated For Five Projects

	Project	Project	Project	Project	Project
Map Layout					
Raster Map	X	X	X	---	X
Vector Map	X	X	X	X	X
Platform					
Windows	X	X	X	---	---
Unix	---	---	---	X	X
Tools					
Zoom In	X	X	X	X	X
Zoom Out	X	X	X	X	X
Zoom Scale	X	X	X	X	X
Pan	X	X	X	X	X
Symbols					
Font Symbols	X	X	X	X	X
Image Symbols	---	---	---	---	---
Geometric	X	---	---	X	X
2D Analysis					
Distance	X	---	X	X	X
Altitude	X	---	X	X	X
3D Analysis					
Visibility	X	---	---	---	X
Profile	X	---	---	---	X
Coordinate Conversion					
UTM	X	X	X	X	X
Geographic	X	X	X	X	X

As shown in the feature table nearly all the features are similar for the applications. 3D analysis can be mentioned as an exception because it is not common to all applications. In addition, image symbols are not used in any of the projects. This is because the symbology abilities of used COTS. Since we design a generic framework and COTS can change, we use image symbols as a feature of our domain.

After completing the context diagram, requirements document, feature diagram, and feature table, the analysis part is completed. The next step for domain engineering is outlining the domain model and implementation of the domain. In this thesis, modeling and implementation parts are not emphasized. Our task is to develop an application development framework by using the components prepared by [8]. The components are prepared and their usages are discussed in [8] with details.

CHAPTER 4

APPLICATION DEVELOPMENT FRAMEWORK IMPLEMENTATION

After completing the domain analysis and components for GIS domain are implemented, a generic application development framework is developed. This developed framework is tested with the analyzed GIS domain and a sample domain, which is created for test purpose. By inspecting the structures of several code generators and usage of Java Beans, a generic approach is developed for offering default relations and code generation part. This approach allows creating code generator for any domain and by loading components and defining relations. The approach is also suitable for other components such as COM and CORBA. Because of the tool used for GUI is coded in java, code generation for java and Java Beans are used. This approach assumes that the interaction is always between the application and the components. Components interact with each other by passing parameters or with event mechanisms. This approach is applied for GIS domain and an application developer is completed for this domain. For GUI part of the application developer, the COSECASE tool is used. This tool provides the ability of component oriented design and analysis. The domain creation and application development parts are implemented for this tool.

4.1. A General Code Generation Approach using Java Beans

Since the main idea behind this thesis is to develop an application generator for the GIS domain, a generic approach is needed to solve the problem. Application developers and components are traditional outputs of a domain engineering process. An application developer uses the generic domain model to develop a special application for the domain. Our idea is to go one-step further and design a technique for developing an application developer.

The components interact with each other by their interfaces. Moreover, an application uses components as parameters to send to other components or directly through function calls defined in their interfaces. By using this approach, applications can be developed if we can define the components to be used, the interfaces that will be used by other interfaces, the parameters that will be passed to interfaces and interface call sequence for the application. This interaction is showed in **Figure 4.1.**

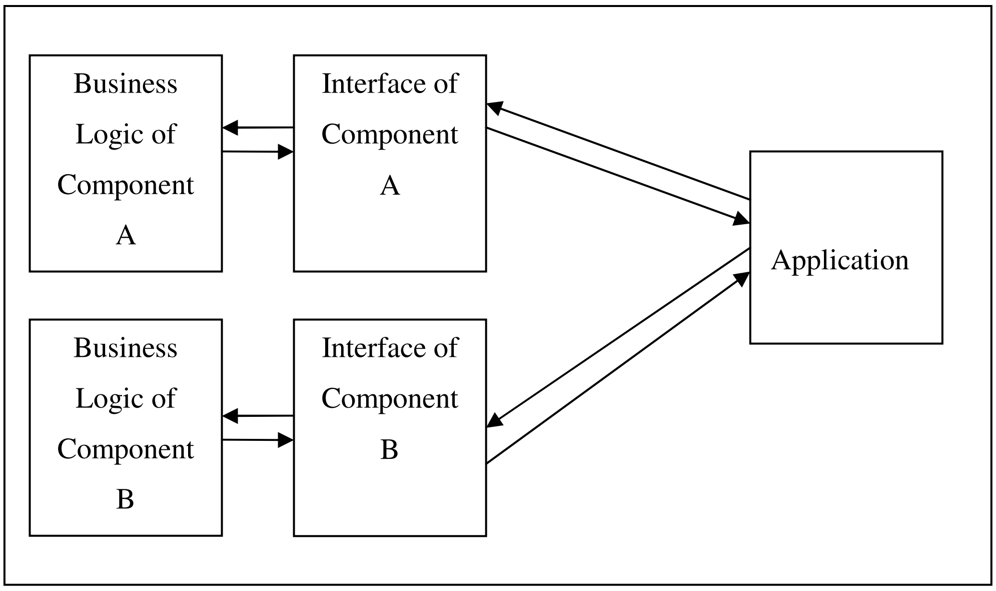


Figure 4.1 Interaction Between Components by using Application

Components that are developed in java do not interact with each other directly. The function calls are only allowed between the application and a component. Although some programming languages use event mechanism for direct function call, java uses a listener mechanism for events fired by components. For this mechanism, components implement some interfaces of the event listener.

In addition, the implementation of what to do for the fired event is implemented as business logic for the component. To add a listener to a component's event, another component implementing the related event interface can be passed through the application. Therefore, to implement event mechanism for the java beans, the application must transmit objects that implement the correct interface.

If we can define and save relations between interfaces and application then the user, (application developer) can use these default relations to generate simple applications. The relations can be expressed as a data structure explained below:

- Application Class Name (name of the application class that makes the function call to component)
- Application Function Name (function name that the application calls)
- Component or Application Class Name (name of component or application class related to the function called)
- Component's Interface Name (name of interface that has the given function)
- Component's Function Name (name of the destination function)
- Parameters (can be expressed as another component or a function call from any of the components used in the component repository and denotes an array)
    - Object or Function designator
    - Object name
    - Function name (empty if object is passed as parameter)
- Function Call Sequence Number (sequence number of the destination function call)

As an example, consider that Component A and Component B will be used for an application. Application has a function called main and this function is executed with the program execution. First, application calls for the function

"AInterface1Func1" from component A's Interface "Interface1" by using two parameters (an instance of B and "AInterface1Func2" function). Then "AInterface1Func3" function is called from the application. "BInterface1Func1" function is called at the end of the application. The sequence diagram and the data structures that are saved for these relations are given in **Figure 4.2**. The data structure is shown in UML notation since UML offers a universal approach for data representation.

```
<Relations>
     <Relation>
          <Source Class Name>
                    Application
          </Source Class Name>
          <Source Function Name
                    main
          </Source Function Name
          <Destination Class Name>
                    A
          </Destination Class Name>
          <Destination Interface Name>
                    Interface1
          </Destination Interface Name>
          <Destination Function Name>
                    AInterface1Func1
          </Destination Function Name>
        <Parameters>
                    <Parameter>
                              <Object of Function Designator>
                                        Object
                              </Object of Function Designator>
                              <Object Name>
                                        B
                              </Object Name>
                              <Function Name>
                              </Function Name>
                    </Parameter>
                    <Parameter>
                              <Object of Function Designator>
                              Function
                              </Object of Function Designator>
                              <Object Name>
                              A
                              </Object Name>
                              <Function Name>
                              AInterface1Func3
                              </Function Name>
                    </Parameter>
          </Parameters>
          <Call Sequence>
               1
          </Call Sequence>
     </Relation>
     <Relation>
          <Source Class Name>
                    Application
          </Source Class Name>
          <Source Function Name
                    main
          </Source Function Name
          <Destination Class Name>
                    A
          </Destination Class Name>
          <Destination Interface Name>
                    Interface1
          </Destination Interface Name>
          <Destination Function Name>
```

```
                AInterface1Func3
            </Destination Function Name>
            <Parameters>
                <Parameter>
                </Parameter>
            </Parameters>
            <Call Sequence>
              2
            </Call Sequence>
        </Relation>
        <Relation>
            <Source Class Name>
                Application
            </Source Class Name>
            <Source Function Name
                main
            </Source Function Name
            <Destination Class Name>
                B
            </Destination Class Name>
            <Destination Interface Name>
                Interface1
            </Destination Interface Name>
            <Destination Function Name>
                BInterface1Func1
            </Destination Function Name>
            <Parameters>
                <Parameter>
                </Parameter>
            </Parameters>
            <Call Sequence>
                3
            </Call Sequence>
        </Relation>

    </Relations>
```

Figure 4.2 Representations of Three Sample Relations

Moreover, to enhance this approach, we assume that the relations between the application layer and components are mostly defined. Therefore, if we can save these relations and components as a domain, the architecture for application development will be ready.

Anyone who wants to develop a custom application can use these default relationships for a rapid application development. Our suggestion is saving the components implemented after a successful domain analysis, and default relations uncovered in the step of domain analysis, is enough for application generator for any component repository.

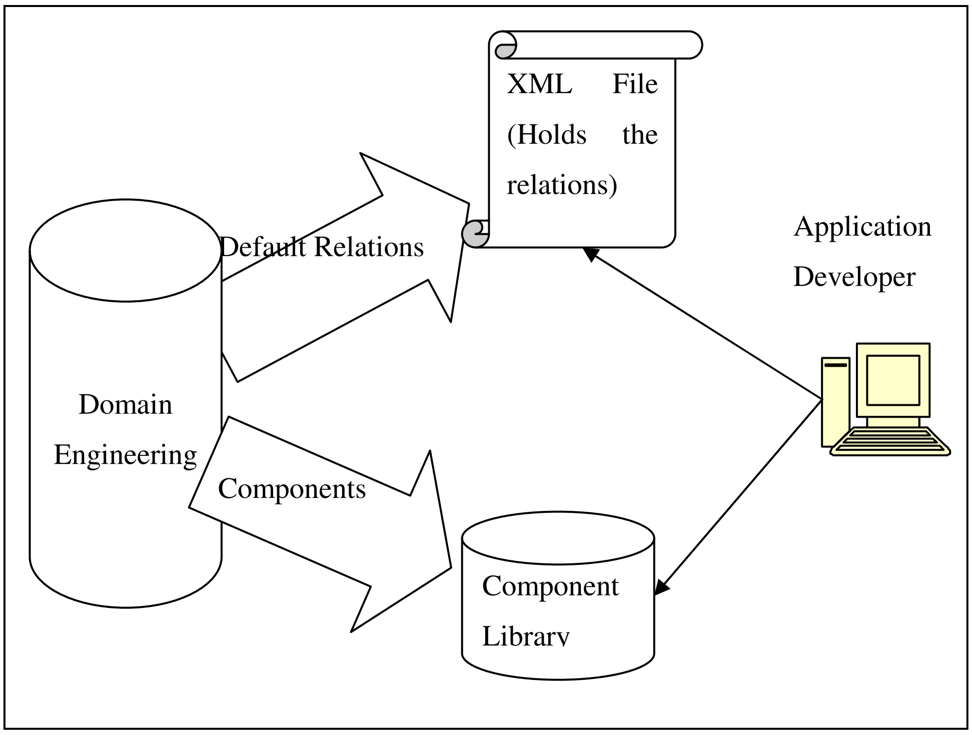


Figure 4.3 Design For An Application Developer

This generic approach is tested with the GIS domain and a simple test domain that is developed for demonstration purpose. By using the implemented framework, some sample applications are generated and the goals of reuse are achieved.

4.2. Data Structures Used for Porting Java Beans into COSECASE and Overview of Framework

The framework has two aspects for data structures. One is “Mapping Java Beans to Component Model for Domain Creation” and the other is “Mapping Component Model to Java Code for source code generation”.

Previously created jar files are being reused. Each jar file is abstracted and java classes residing inside the jar are regarded as components. Interfaces are connected to the component via composition relation. Inheritance and composition relations defined between java classes are mapped as inheritance and composition relations between components. While mapping java classes to components, properties, methods in, events in and events out information of the component interface are collected through applying reflection API of JAVA and certain filtering operations on java classes in jar files.

Properties of components are obtained through accessing fields of java classes through *public Field[] getFields() throws SecurityException* API of Class. If the field type is primitive (int, long, char etc.), is interface or is abstract they are taken into account as properties. However if field type is a class that does not fall into above categories and resides in jar library; it is regarded as a new component that is connected through composition relation to the component that is parsed.

All the classes in a given jar file are not ported to components. A special package is filtered and classes in this package are ported as components. This special package name is read from a configuration file since the name of the package can vary across different domains. There are two reasons for why package filtering is used.

First reason is, if there was no filter on the package or jar files - though any java class can still be mapped – then the project would go beyond its scope by trying to display hundreds of components connected through spider-web like links, making everything unnecessarily complex to perceive. Thinking about how many times a field with type *java.lang.String or Java.lang.Object* is declared in a class, the tremendous frequency would make COSE nothing but just useless blocks of information displayed. The vision is to be able to create new GIS domains; save, reuse them and generate executable java source code resulting in functional output GIS API without writing source code, but just with preserving the component links drawn throughout COSEML tool.

In addition, the special package choice is needed for java code generation. At the beginning, for the GIS domain the *"GISFramework.Interfaces"* package is selected as the special package. However, to allow code generation the choice is changed and the *"GISFramework.Beans"* package is selected. Both of them display the same interfaces but by using Beans package, the code generation part can be more accurate. For the Test domain, *"Test.Beans"* package is selected. The implementation also allows using all of the classes in a jar file by giving a higher package name as a special package. For example if the *"GISFramework"* package is selected in the configuration file, all the classes in the jar file will be ported as components since all of the other packages reside in this package.

Methods of a component are obtained through the elimination of Public methods of the corresponding java classes. There is also an extra decomposition between public methods since a public method may map onto a component's method-in, event-in, or event-out. In Java language, event handling is carried out through the observer pattern. Events-in means the output of an event notifies events-in of the component that is desired to be notified and events-out means the source of the event. Let us convert the above statements into JAVA language, events-out fall into methods that help to register an observer on the observed, which in return will notify the observer about the changes through events-in of the observer. Then this perspective directs us to the steps listed below:

The template for events-out discovery:

- add<ListenerType>(<ListenerType> listenerType)
- remove<ListenerType>(<ListenerType> listenerType)

- where <ListenerType> has to be a subclass of java.util.EventListener

The template for events-in discovery:

- Void <eventOccurenceMethodName>(<EventObject Type> evt)

The class which public method is declared has to be a subclass of a class that implements *java.util.EventListener* or implement *java.util.EventListener*. Briefly the candidate class should be assignable to a java*.util.EventListener* object and the parameter <EventObject Type> should be derived from *java.util.EventObject*. If the public method is not in any of two categories, then it is placed into methods-in set.

Inheritance relations are almost the same as they are in the object-oriented perspective. If the class implements a java class or the superclass is abstract, then this class is not converted into a component. Otherwise, the superclass is mapped into a component which is connected by a component inheritance link to the subclass representing the component.

Method links can not be drawn from byte-code due to the limited information contained in the java class; which method of which java class is called can not be revealed. Though events-in and events-out can be determined, event links cannot be drawn because of the same reason for not having the capability of drawing method links no matter how reflection package is provided. **Table 3** and **Table 4** list required information for mapping. Also, **Figure 4.4** shows the representation of a sample component in the framework

Table 3 Availability of COM Interface Features in Mapping Jar Files to COM

COM Interface Features	**Properties**	**Methods-In**	**Events-In**	**Events-Out**
Availability	Available	Available	Available	Available

Table 4 Availability of COM Links in Mapping Jar Files to COM

COM Link Types	**Method Link**	**Event Link**	**Inheritance Link**	**Compositio n Link**	**Connector Link**	**Represents Link**
Availability	Not Available	Not Available	Available	Available	Not Available	Not Available

One of the goals of this thesis is to ease usage of a jar file corresponding to the selected component repository to obtain an executable program. User is allowed to draw new method links that will be included in code generation. User can generate an executable code through a special component named *Application* that does not exist in any of the jar files. It is the main class for generated code execution. When user defines a method link, the sequence number is automatically assigned. Moreover, on the method link user can set parameter values for the method that is invoked. To set the link properties/parameters first "select the method link" then "right click the method link and select Properties menu from popup menu".

Through properties tab;

1. It allows displaying parameter values.
2. It allows user to select a pair of values consisting of class type and method, which the declared class owns and whose return type can be assigned to the parameter type.
3. It allows user to define and edit values both for java reference and java primitive types manually.

When user selects "Domain Operations->Generate Domain Source Code"; the java code mapping to the entire picture is generated and displayed. In case user may want to run the code displayed then if there is a compilation or runtime problem, user is warned, else the code runs.

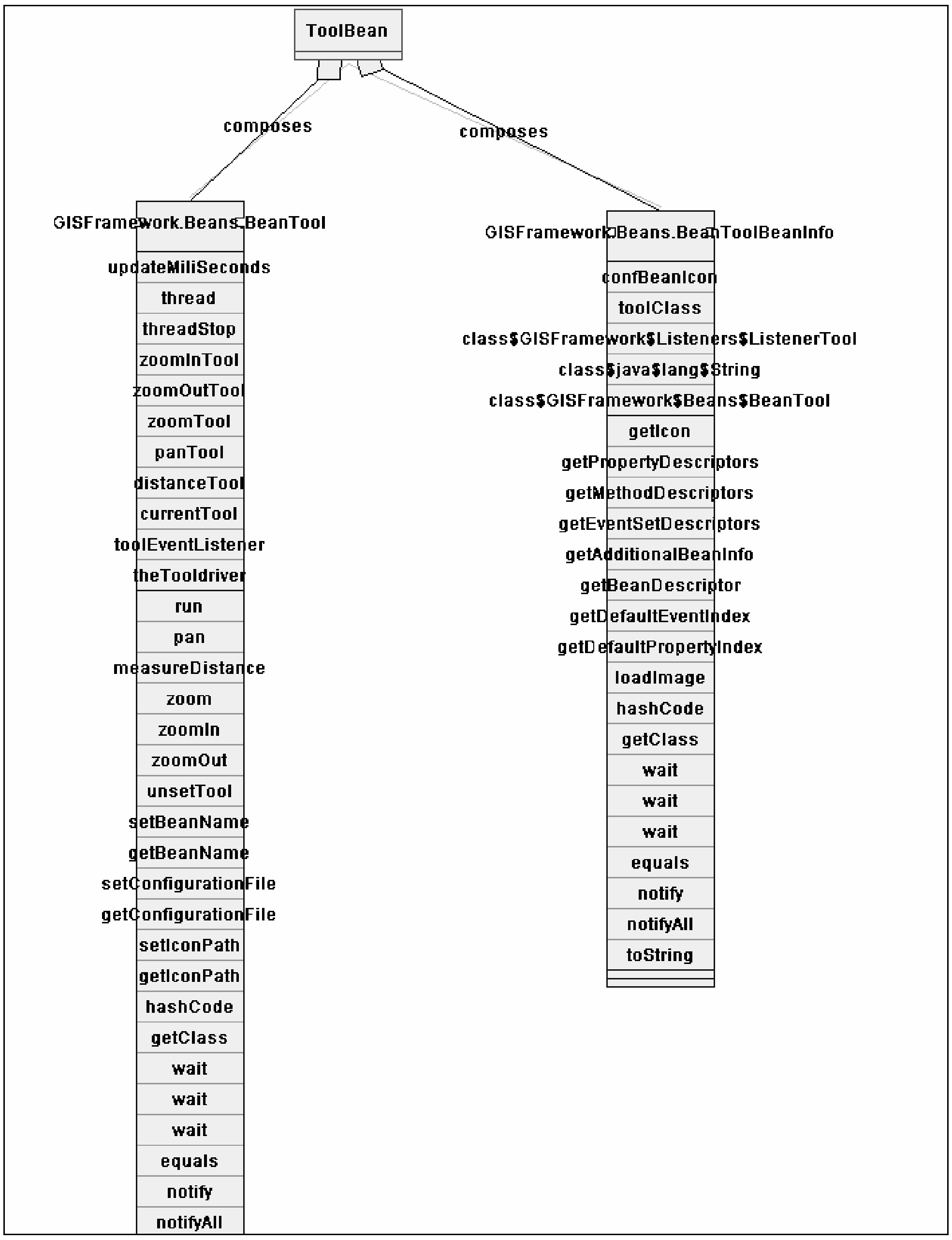


Figure 4.4 Representation of a Sample Component in Framework

Some data structures are designed and coded for COSECASE to allow domain creation. Some key classes and their usages are given below.

"ADTOO2COMComponent" is coded for porting java beans to COSECASE environment. Every bean is created as an *"ADTOO2COMComponent"* instance.

This class has five properties. First one is an array to hold the relations from this class to other classes, the other is a class object that refers to the ported bean, the third one is a file descriptor which refers to the full path of the jar file containing the bean, the fourth one holds an instance *"AKComponent"* that is used for the graphical view of the component, and the last one is the name of the component. The class declaration is shown below:

```
public abstract class ADTOO2COMComponent {
   private Vector relations;//holds the relations created from this component
   private Class javaClass; //class of bean ported
   private File containingJarFile; //full path of jar file containing the bean
   AKComponent graphicalCOSEView;
   String name;//name of the component
   /*Standard set and get methods*/
}
```

"ADTOO2COMInterface" is coded for showing interfaces of components. As soon as the java bean is ported to COSECASE, the interfaces that are implemented by components are discovered and initiated as *"ADTOO2COMInterface"* instances. This class has the variables to hold properties, event-in functions, event-out functions, and other functions. The class declaration is shown below:

```
    public abstract class  ADTOO2COMInterface {

        Vector properties;

        Vector methods;

        Vector eventsIn;

        Vector eventsOut;

        /*Standard set and get methods*/

  }
```

By using *"ADTOO2COMInterface", "ADCOM2OOMethod", and "ADTOO2COMComponent"* classes the transformation of a jar file is shown in **Figure 4.5.**

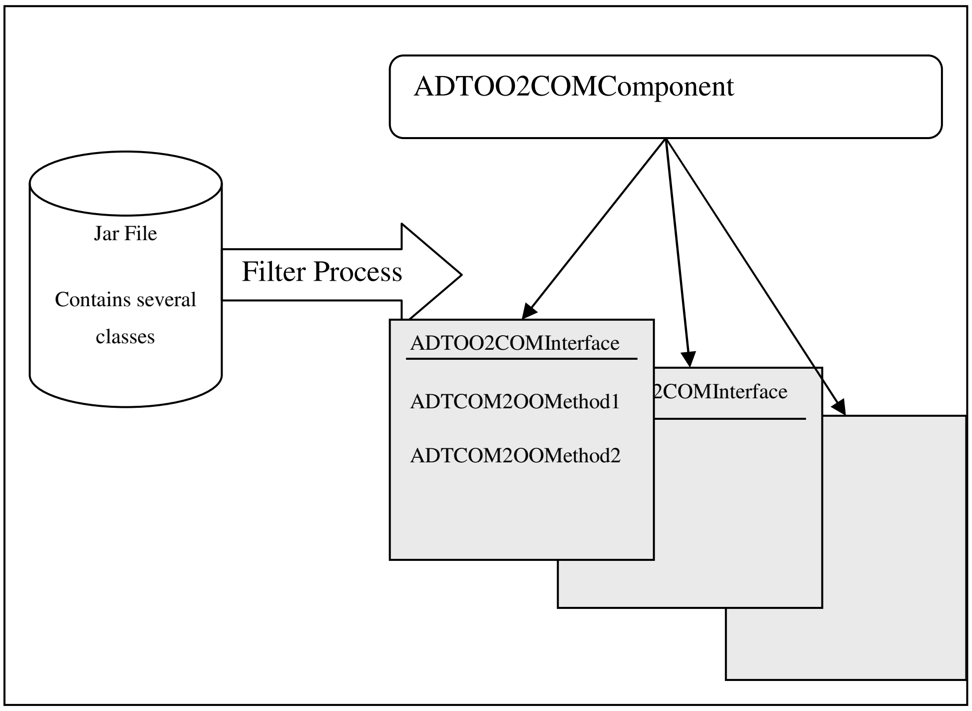


Figure 4.5 Transformation of a Jar File

"ADTOO2COMRelation" class is used for holding the relations inside the beans. These can be event-links, method links, inheritance, composition, and represents. But in the implementation part even-links and method links are not used. Because jar files includes compiled classes, the interaction inside beans is somewhat not possible to discover. Also represents and composition relations are not used because they are not needed. The class declaration is shown below:

```
  public abstract class   ADTOO2COMRelation {

     ADTOO2COMBaseComponent startRelationOwnerComponent;

     ADTOO2COMBaseComponent endRelationOwnerComponent;

     AKLink2 graphicalCOSEView;

     /*Standard set and get methods*/

}
```

"ADTCOM2OOMethod", "ADTCOM2OOParameter", and *"ADTCOM2OOParameterValue"* classes are used for code generation. Also these classes are serialized to an XML file to allow saving the relations created in the domain creation part.

"ADTCOM2OOParameter" class is used for saving user defined parameters. It has three properties. First one is the parameter sequence number in the related function, the second one is the value of the parameter which is an instance of *"ADTCOM2OOParameterValue*", and the last one is the type of the parameter. The class declaration is shown below:

```
public abstract class   ADTOO2COMRelation {

    int parameterSeqNum = -1;

    ADTCOM2OOParameterValue parameterValue;

    Class paramtype;

     /*Standard set and get methods*/

}
```

"ADTCOM2OOParameterValue" holds the user defined parameter value that is used for user defined function calls. The parameter can be a function call from any component, an instance of a component, or a basic type as integer or string. To allow all of these, this class has the properties such as component name, function name, instance or function call descriptor and value for basic types. The class declaration is shown below:

```
public abstract class    ADTCOM2OOParameterValue {

    String jarName; //name of the jar

    Class aClass; //name of the class

    Constructor constructor; //used as constructor or function call descriptor

    Method method; //method property if a function call is used

    String value;//holds the simple type parameter

     /*Standard set and get methods*/

}
```

In addition, the *"XMLDomainData"* class is coded to hold the array of components used in the domain and the relations between these components.

4.3. Creating Domain from Implemented Java Beans

First part of creating an application generator is to apply the domain design to the application generator. To do this *create domain* functionality is added to the COSECASE tool. The functionality serves a user interface to user for loading beans that belong to a specific domain and defining basic relationships in that domain.

To achieve domain creation, COSECASE extended with domain operations menu. This menu is shown in Figure 4.6. The first element of domain operations menu is "Create Domain".

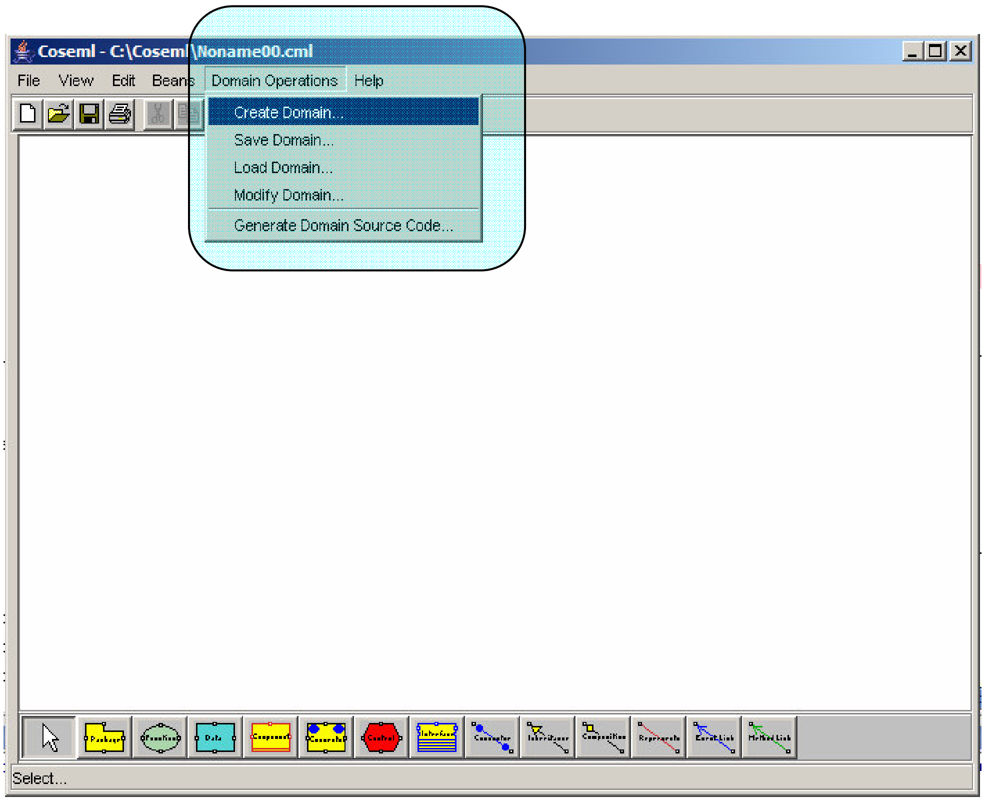


Figure 4.6 Domain Operations Menu

When create domain menu item is selected by the user, an empty screen is created and application component is initially put on the screen. This application component plays a chief role in the domain creation part since code generation will be done according to the interaction between the application component and other core components. User can add external components for the component repository by selecting the components placed in the left side in the window. The component repository listed in the left side contains the components residing in a special folder defined in the configuration file. The component repository used for the GIS domain is shown in **Figure 4.7.**

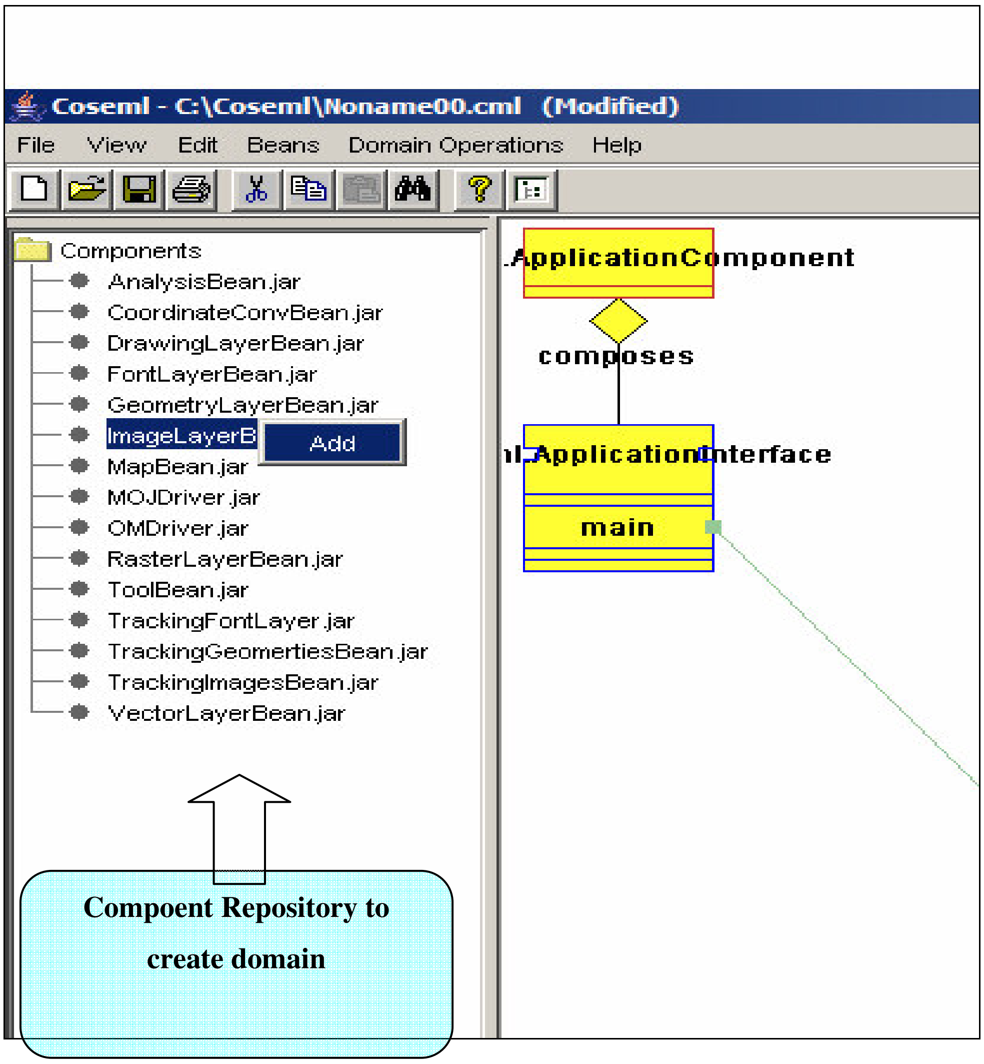


Figure 4.7 Components used for Creation a Domain

The components that are selected and added to the domain are converted to COSEML representations and displayed on the screen. All the interfaces that are implemented by the bean are shown on the screen. This operation is done recursively to show all the functions that a component serves with its interfaces. After all the components for the domain are added, the relations between components and the application component are defined. To define a relation in a domain, *method links* are used. *Event links* are not used since event mechanism is implemented by using the listener design pattern.

To create a relation between components on the screen, *"event link"* is selected and the start and end locations of this link are defined. The orders of relations are the same as the defining order. When a relation is defined, the next sequence number assigned for it. Therefore, the sequence numbers are given in an increasing order. The representation of a sequence number in a method link is shown in **Figure 4.8.**

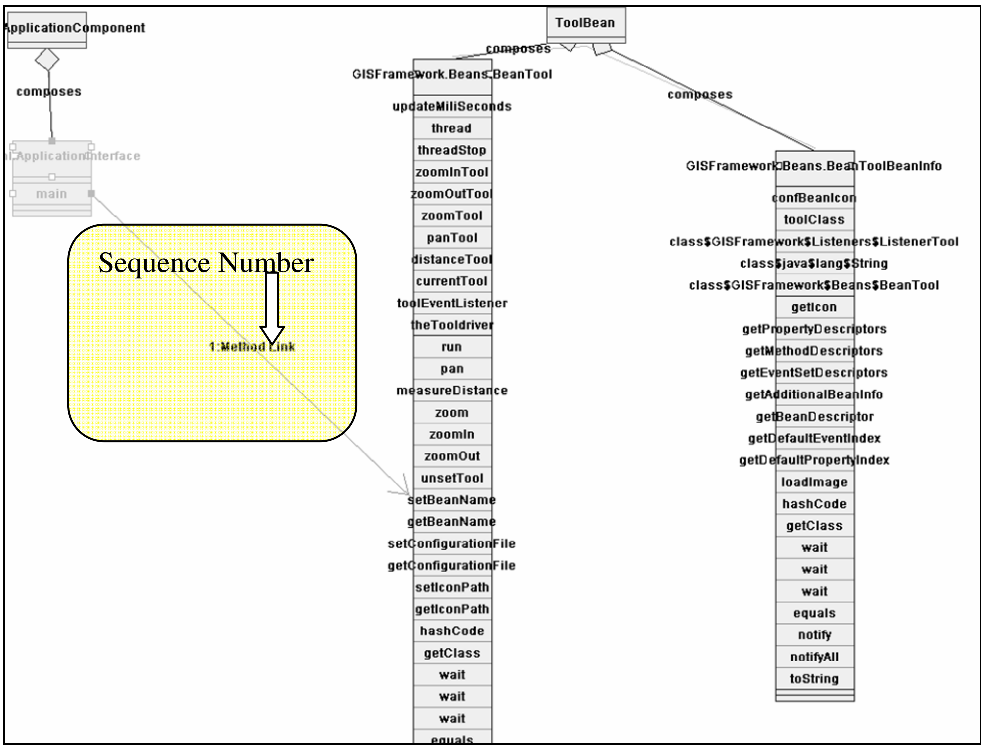


Figure 4.8 Representation of Sequence Number in Design Mode

The parameters that a function's signature has must be defined to save a full domain. To define the parameters between interfaces and the application, a *vector* property is added to method link in COSECASE. This property holds the array of parameters. A user friendly GUI for defining parameters is also implemented.

A parameter can be an instance of a component, a return value of an interface function of a component, or a basic type like string or integer. The types of the parameters are presented in the signature of the function. The parameter type is checked and a set of selection is offered to user. This selection is done by finding the same types in the functions or constructors of components that are added to the domain. Then the user can select the parameter. **Figure 4.9** displays the parameter definition screens.

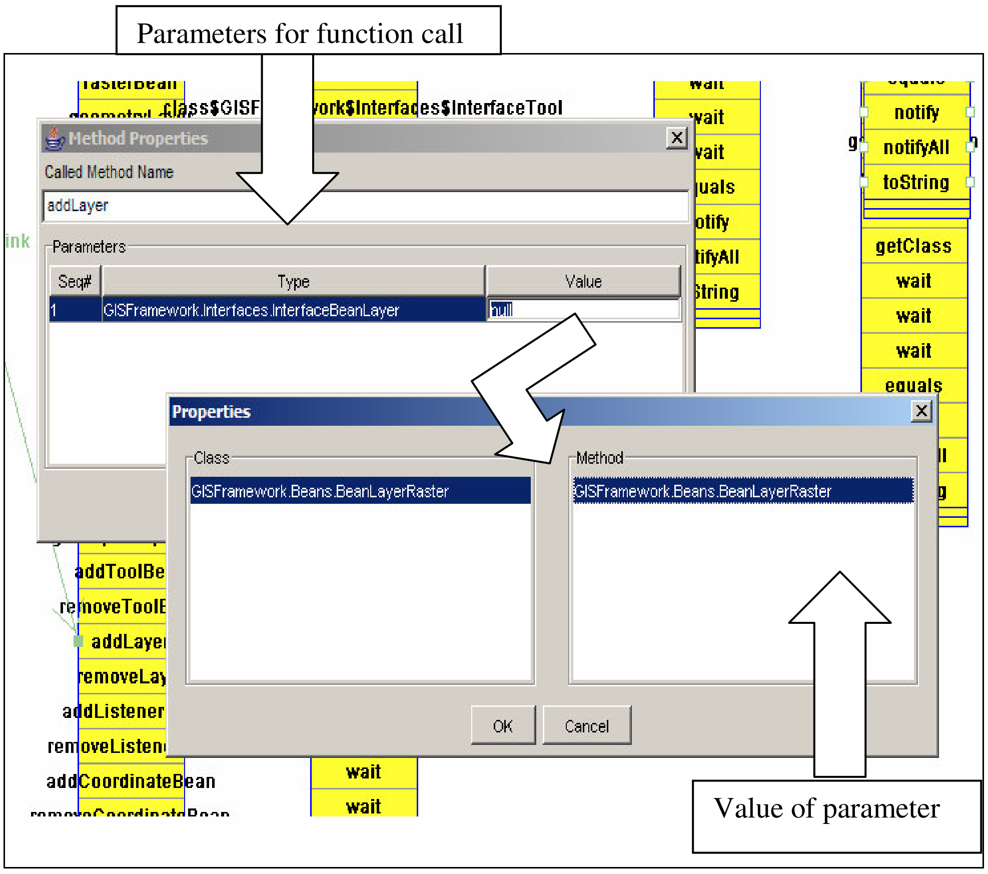

Parameters for function call
Method Properties
Called Method Name
addLayer
Parameters
Seq#
Type
Value
1
GISFramework.Interfaces.InterfaceBeanLayer
null
Properties
Class
GISFramework.Beans.BeanLayerRaster
Method
GISFramework.Beans.BeanLayerRaster
OK
Cancel
Value of parameter

Figure 4.9 Parameter Definition Screens

After the parameters and the orders of function calls are defined, the components and XML representation of relations are saved in a directory. After the domain is created once any developer can load and use default relations for code generation for several times.

4.4. Modifying Domain

After a domain is created, several changes can be made to the domain. To reflect these changes *modify domain* function is used. When modify domain function is selected the domain created by domain engineer is loaded and all the relations are shown automatically. Then the engineer can modify and save the last state of the domain. After the domain is saved, the last state of the domain will be used for all engineers developing applications by using this framework.

4.5. Loading Domain

The application developer will use *load domain* function. Since the main idea of this thesis to offer default relations in a domain, application developer can use this function to make a smart start for the application. When load domain function is selected, the domain list is presented to the user. Our approach offers a generic design for offering default relationships. Different domains can be created and can be loaded at any time. The user first selects the domain of interest. As soon as the user selects the domain, the components in the component repository will be displayed at the left side of screen.

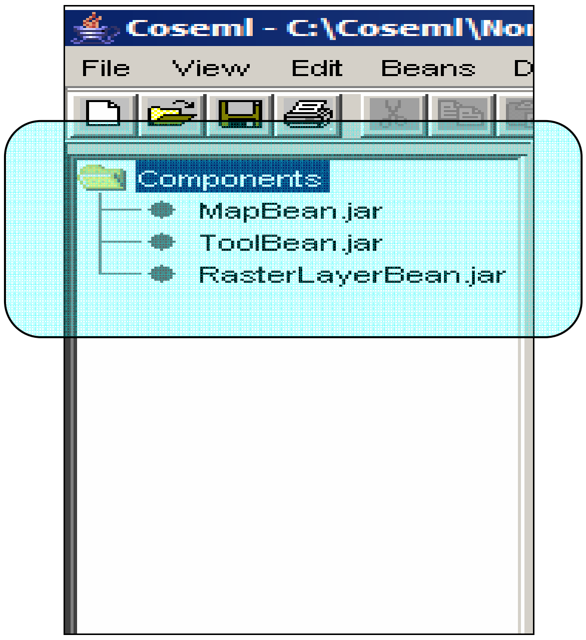


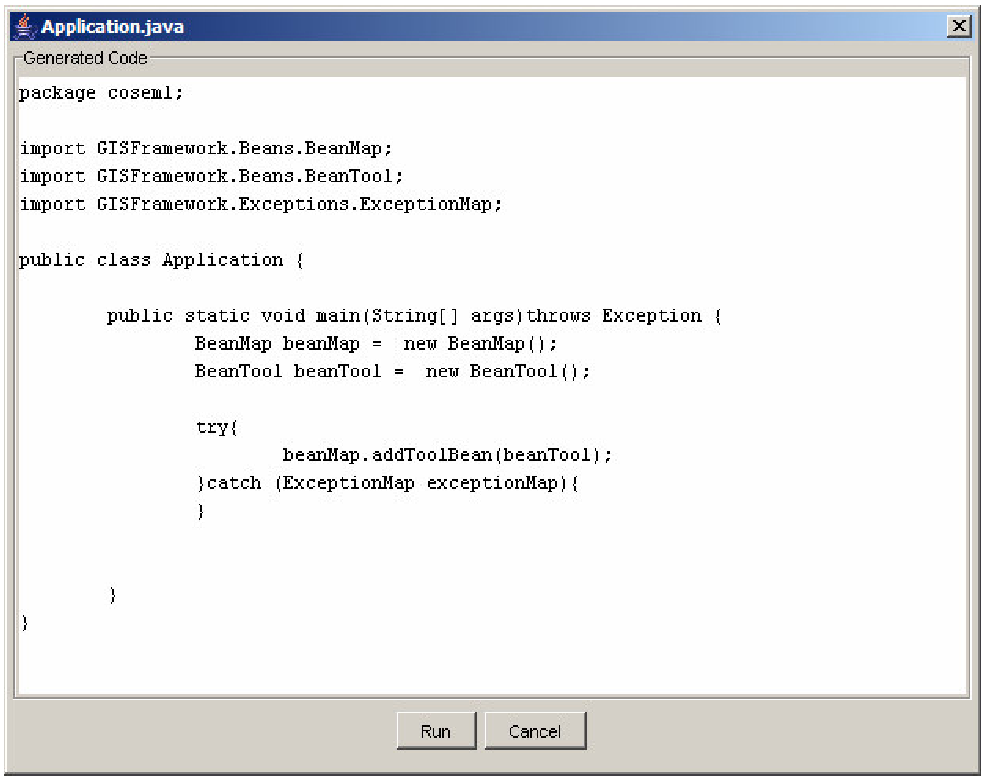
Figure 4.10 Components Available for Code Generation

The user can use any of these components for custom applications. When user adds a component, any relation between this component and the application will be displayed automatically. User can delete this method link or user can add some new method links for the custom application. The sequence numbers will be increased in increasing order as in domain creation part. Likewise, parameter definitions can be done with the same method used in the domain creation part.

4.6. Code Generation

Code generation is the final part of this work. Although our main problem is not code generation, we claim that our approach offers a generic way for code generation.

After an application engineer extends the default relations offered by the framework, he can get the source code of the application by using the *generate domain code* function. The java code is generated from components and relations displayed in the screen. In addition, user can extend the code displayed in the screen just like using a text editor. When the *run* function is selected, the source code is compiled and run. If an error occurs in the compilation part, the user is warned. Figure 4.11 shows the code for a sample application generation.



```
package coseml;

import GISFramework.Beans.BeanMap;
import GISFramework.Beans.BeanTool;
import GISFramework.Exceptions.ExceptionMap;

public class Application {

        public static void main(String[] args)throws Exception {
                BeanMap beanMap =  new BeanMap();
                BeanTool beanTool =  new BeanTool();

                try{
                        beanMap.addToolBean(beanTool);
                }catch (ExceptionMap exceptionMap){
                }


        }
}
```

Figure 4.11 A Sample Code Generated from Framework

Code generation approach is somewhat like similar approaches. First the components used for the custom application are determined and these components are imported to the source code by using the import statement. Then the instances of components used in the application are created to get the instances of these components. Finally by using relations between the application component and other components, function calls are placed into the main function in the source code. The generated code is also saved to the disk for future use.

4.7. GIS Domain Creation

The framework offers a generic approach and this approach can be applied to any domain. To demonstrate the success of the methodology and the framework, GIS domain is selected.

The components used for GIS domain and their definitions are given below.

a. "*BeanMap"* component forms the main user interface component of the GIS domain. The main interaction is supplied by this component. It is the basic map component. The layers are added to this component to show different raster and vector maps.

b. "*BeanTool"* component is implemented for the custom GIS operations like zoom in/out, extend, and scale. This bean is added to MapBean by using its addToolBean interface function. This bean calls some pre-defined functions in the time of fired events. These pre-defined functions form an interface and this interface is implemented by the "*MapBean*" component.

c. "*BeanLayerSymbolsFont*" component is used for drawing symbols that are displayed as fonts. This layer is added to the "*MapBean*" component as a layer.

d. "*BeanLayerSymbolsGeometry*" component is used as a custom drawing layer. This layer provides the ability of drawing different geometries in the need of custom application. It is added to the "*MapBean*" component as a layer.

e. "*BeanLayerSymbolsDrawing*" component is used as a symbol-drawing layer. This layer is added to the MapBean layer and interaction is obtained by mouse events.

f. "*BeanLayerSymbolsImage*" component is used as a custom image-drawing layer. This layer provides the ability of drawing images to use as a geometry or as a symbol. It is added to the “MapBean” component as a layer.

g. “*BeanLayerVector*” component is used to show vector layers. This component is added to the “*MapBean*” component as a layer. The path of the vector data (map) must be set to this component before it is related with the “*MapBean*” component.

h. “*BeanLayerRaster*” component is used to show raster layers. This component is added to the “*MapBean*” component as a layer. The path of the raster data (map) must be set to this component before it is related with the “*MapBean*” component.

i. “*BeanTrackingFontLayer*” component is the tracking form of the “*BeanLayerSymbolsFont*” layer.

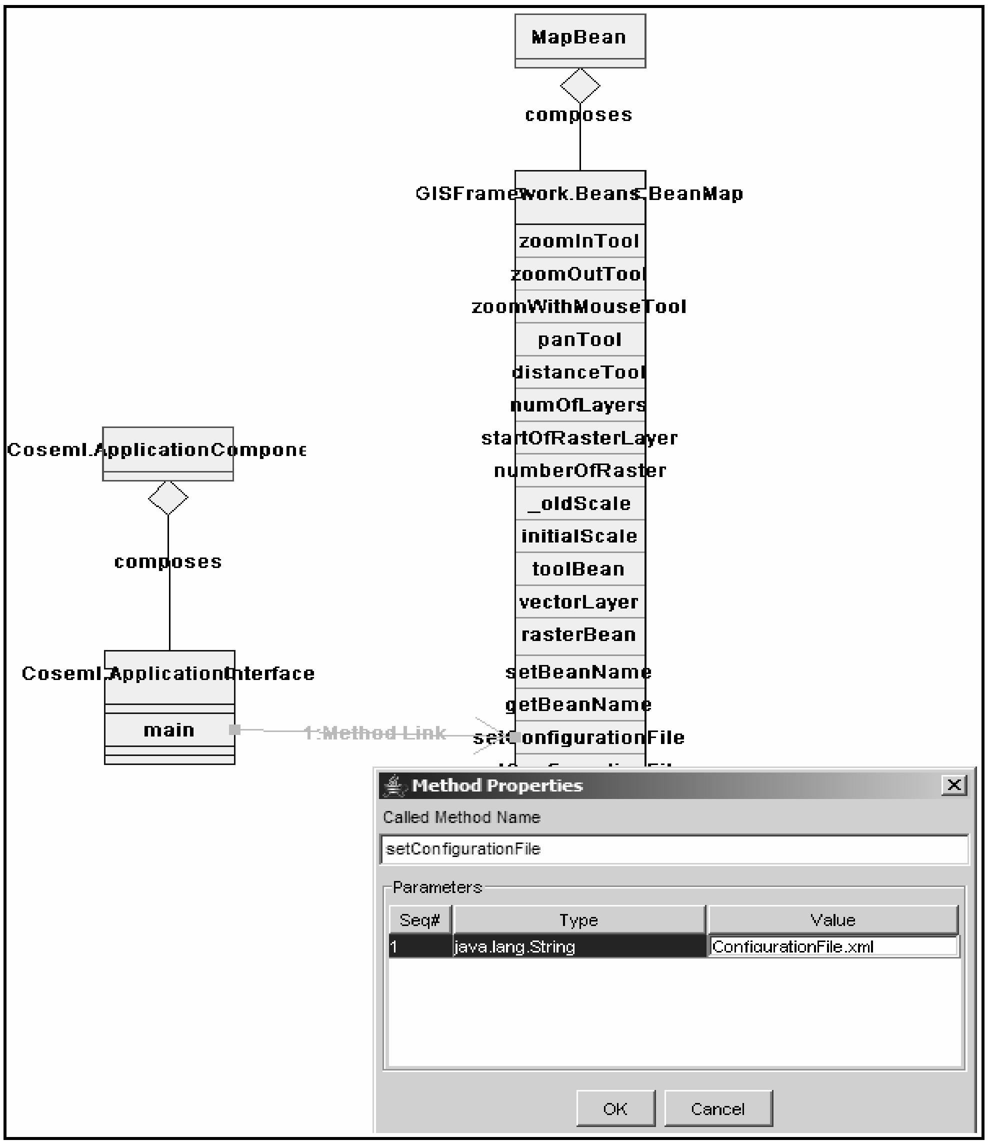
j. “*BeanTrackingGeometriesLayer*” component is the tracking form of the “*BeanLayerSymbolsGeometry*” layer.

k. “*BeanTrackingImagesLayer*” component is the tracking form of the “*BeanLayerSymbolsImage*” layer.

l. “*BeanCoordinateConverter*” component makes the coordinate conversions.

A relation document is prepared for the GIS domain to define and save the relations by using COSECASE tool. The relations that are defined for GIS domains and their representation in the framework are given below.

RELATION 1: The implementation of the domain allows using different COTS. For all custom applications developed in the GIS domain, the configuration file property must be set for *MapBean* component. This configuration file is used to select the COTS library used for individual function calls. A link between application and *BeanMap’s SetConfigurationFile* function is created with the sequence number “1”. This function takes a *String* type parameter. The value of this string is set to “c:\\ConfigurationFile.XML” initially. Representation of RELATION 1 is shown in **Figure 4.12**.



Called Method Name

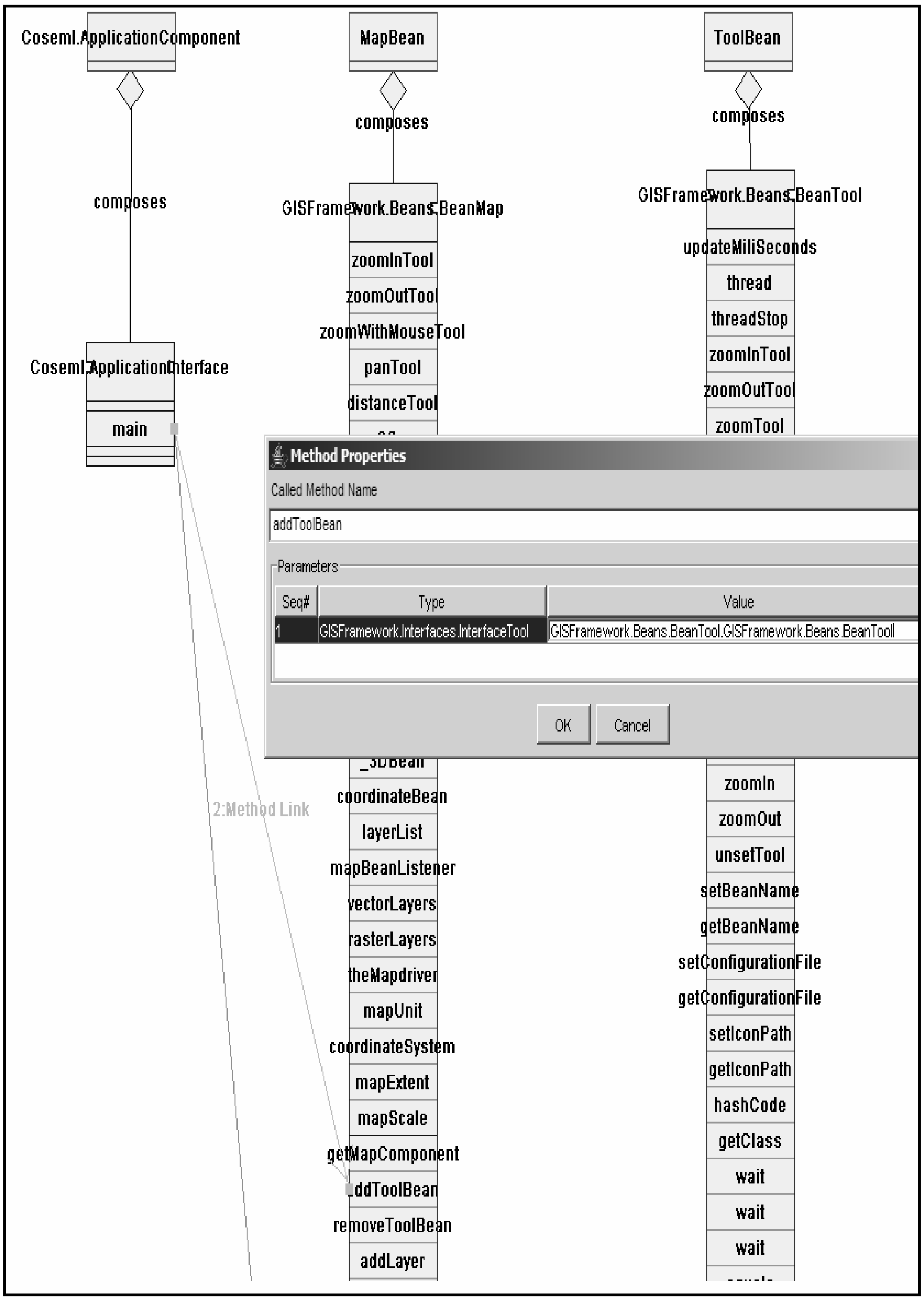
setConfigurationFile

Parameters

Seq#	Type	Value
1	java.lang.String	ConfigurationFile.xml

Figure 4.12 Representation of Relation 1

RELATION 2: To use a tool for custom GIS functions like pan and zoom, an instance of *ToolBean* class must be passed as parameter to *BeanMap*'s *addToolBean* function. The parameter for relation must be set as the constructor function of the *ToolBean* component. When a custom GIS operation is completed, an event is fired from the *ToolBean* component to the *BeanMap* component. Representation of RELATION 2 is shown in **Figure 4.13**.



Called Method Name

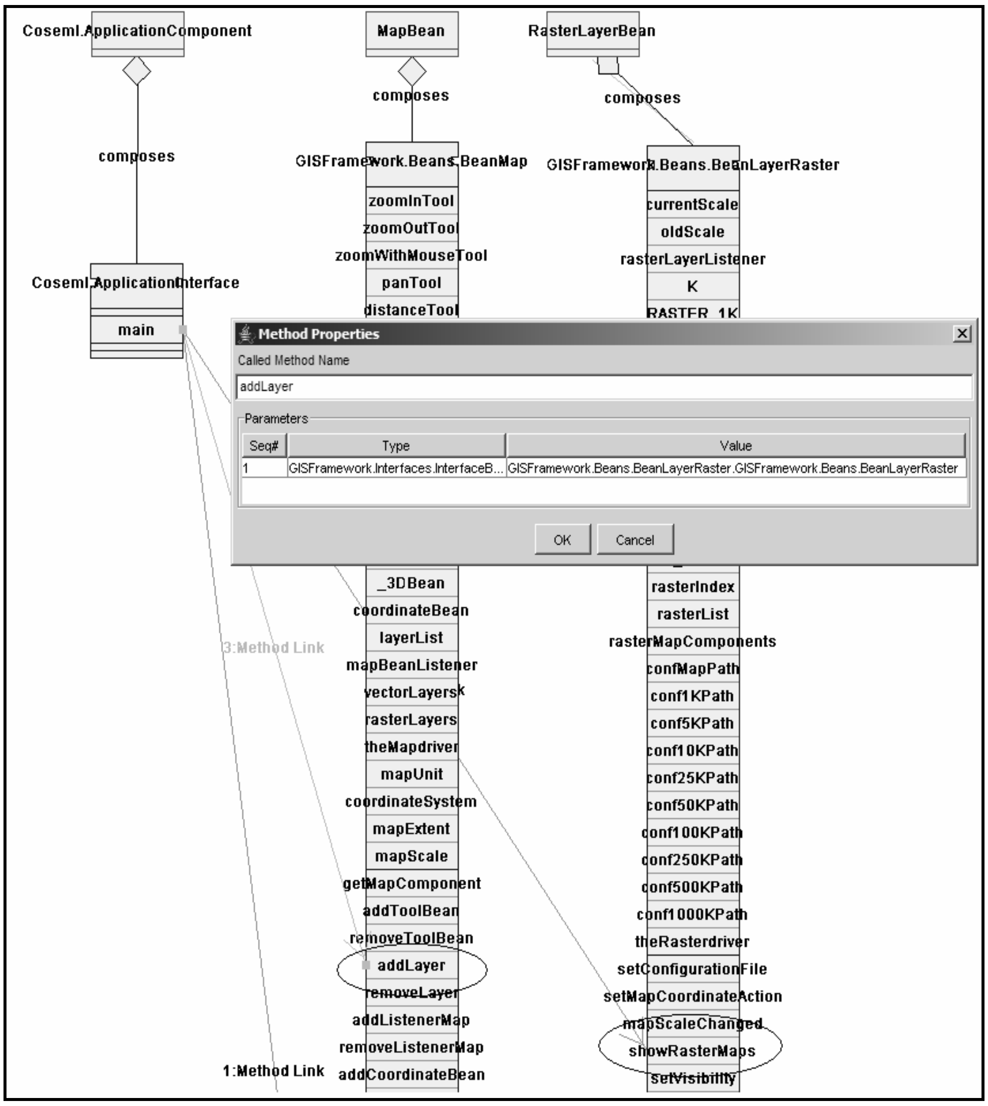
addToolBean

Parameters

Seq#	Type	Value
1	GISFramework.Interfaces.InterfaceTool	GISFramework.Beans.BeanTool.GISFramework.Beans.BeanTooll

Figure 4.13 Representation of Relation 2

RELATION 3: To use a raster map, an instance of the *BeanLayerRaster* component must be passed through the *addLayer* function of the BeanMap component. The name of the raster map can be changed according to the application and this property can be updated while defining special relations for the custom applications in the time of load domain. In addition, to show the raster map the *showRasterMaps* function of the *BeanLayerRaster* component must be called. Representation of RELATION 3 is shown in **Figure 4.14**.



Called Method Name

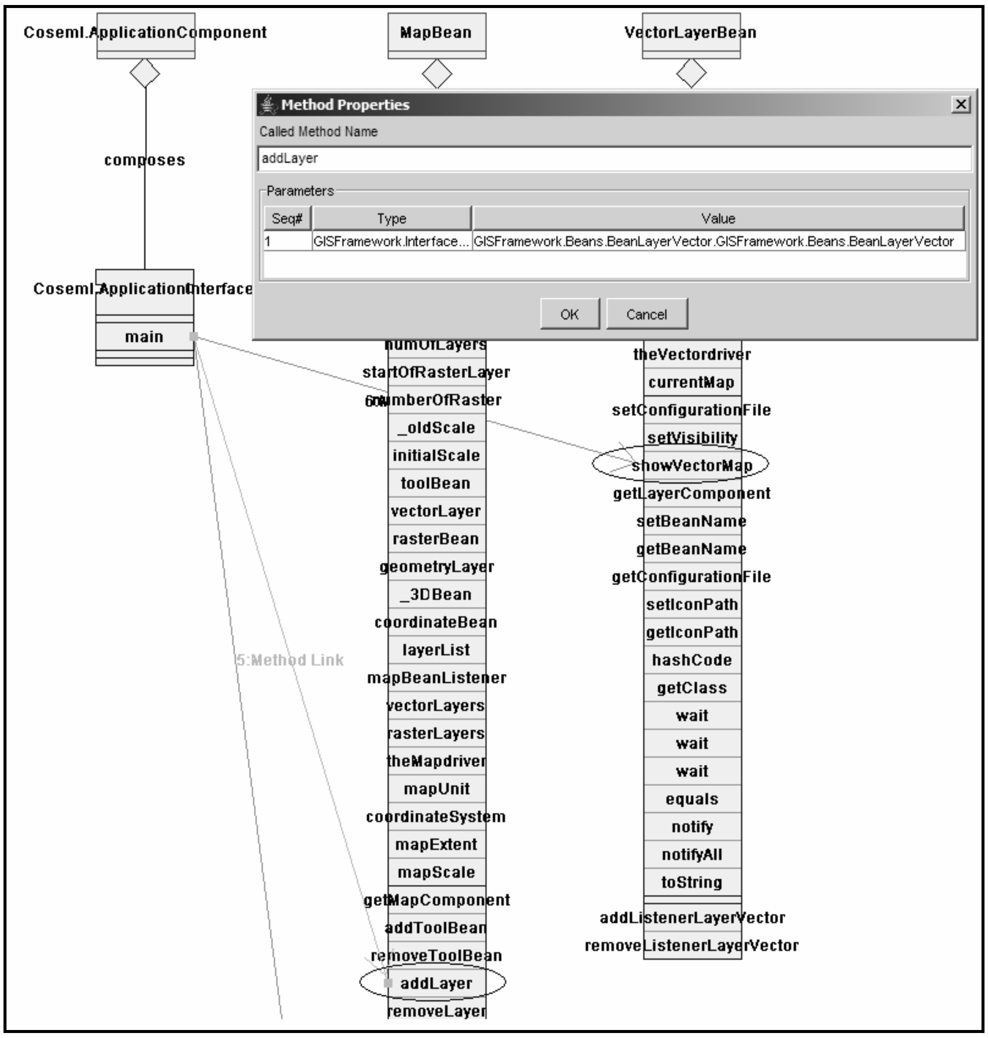
addLayer

Parameters

Seq#	Type	Value
1	GISFramework.Interfaces.InterfaceB...	GISFramework.Beans.BeanLayerRaster.GISFramework.Beans.BeanLayerRaster

Figure 4.14 Representation of Relation 3

RELATION 4: To use a vector map, an instance of the *BeanLayerVector* component must be passed through the *addLayer* function of the BeanMap component. The name of the vector map can vary between applications and this property can be set while defining special relations for custom applications in the time of domain loading. In addition, to show the raster map the *showVectorMaps* function of the *BeanLayerVector* component must be called. Representation of RELATION 4 is shown in **Figure 4.15.**


Method Properties

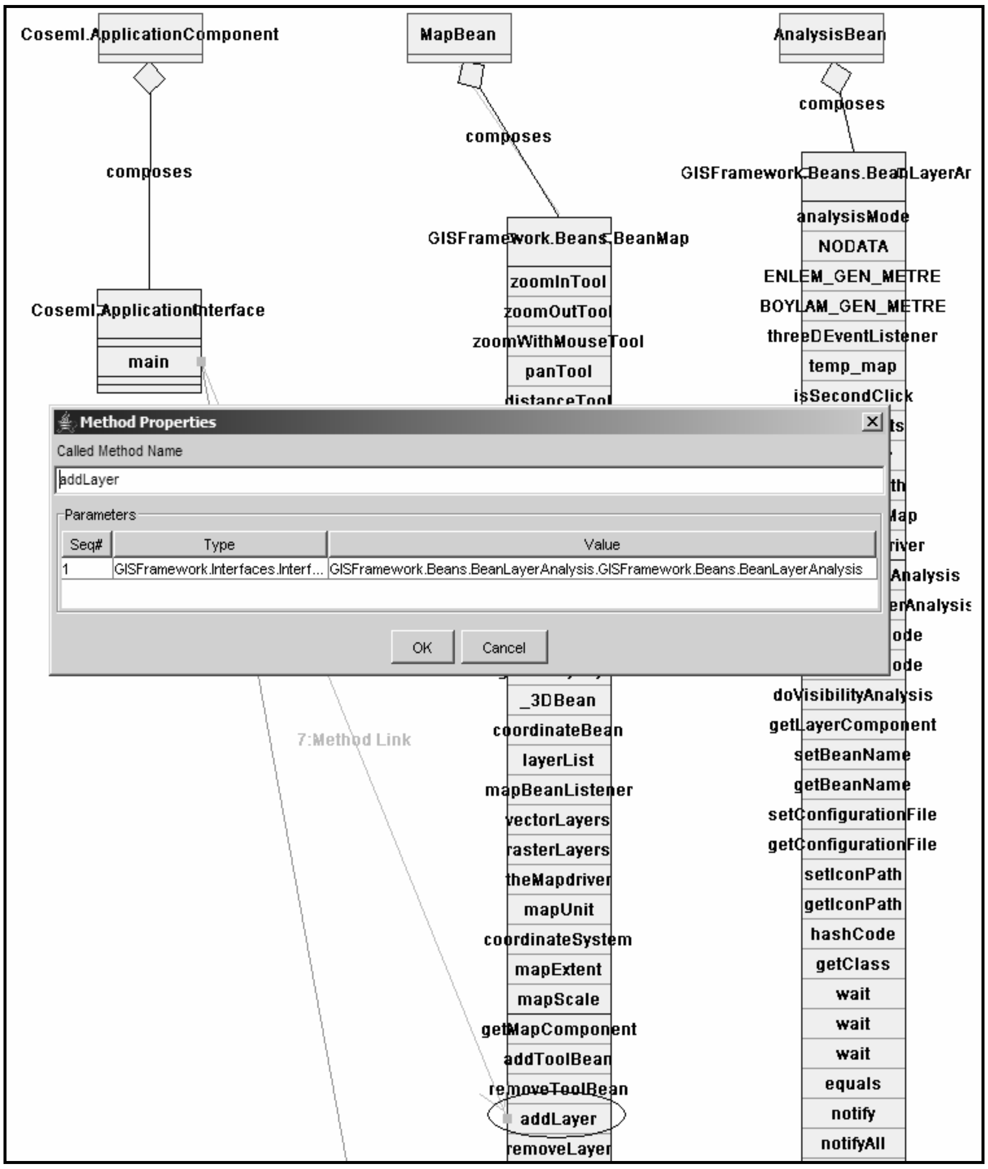
Called Method Name: addLayer

Parameters

Seq#	Type	Value
1	GISFramework.Interface...	GISFramework.Beans.BeanLayerVector.GISFramework.Beans.BeanLayerVector

Figure 4.15 Representation of Relation 4

RELATION 5: To achieve a 3D analysis in GIS domain the *BeanLayerAnalaysisLayer* component must be used. A *BeanLayerAnalaysisLayer* component must be added as a layer to the *BeanMap* component by using the *addLayer* function. To use this component for analysis, some more function calls are needed. Nevertheless, these function calls are not set as default relations because it is believed that they must be implemented in the custom application logic. Representation of RELATION 5 is shown in **Figure 4.16.**



Called Method Name

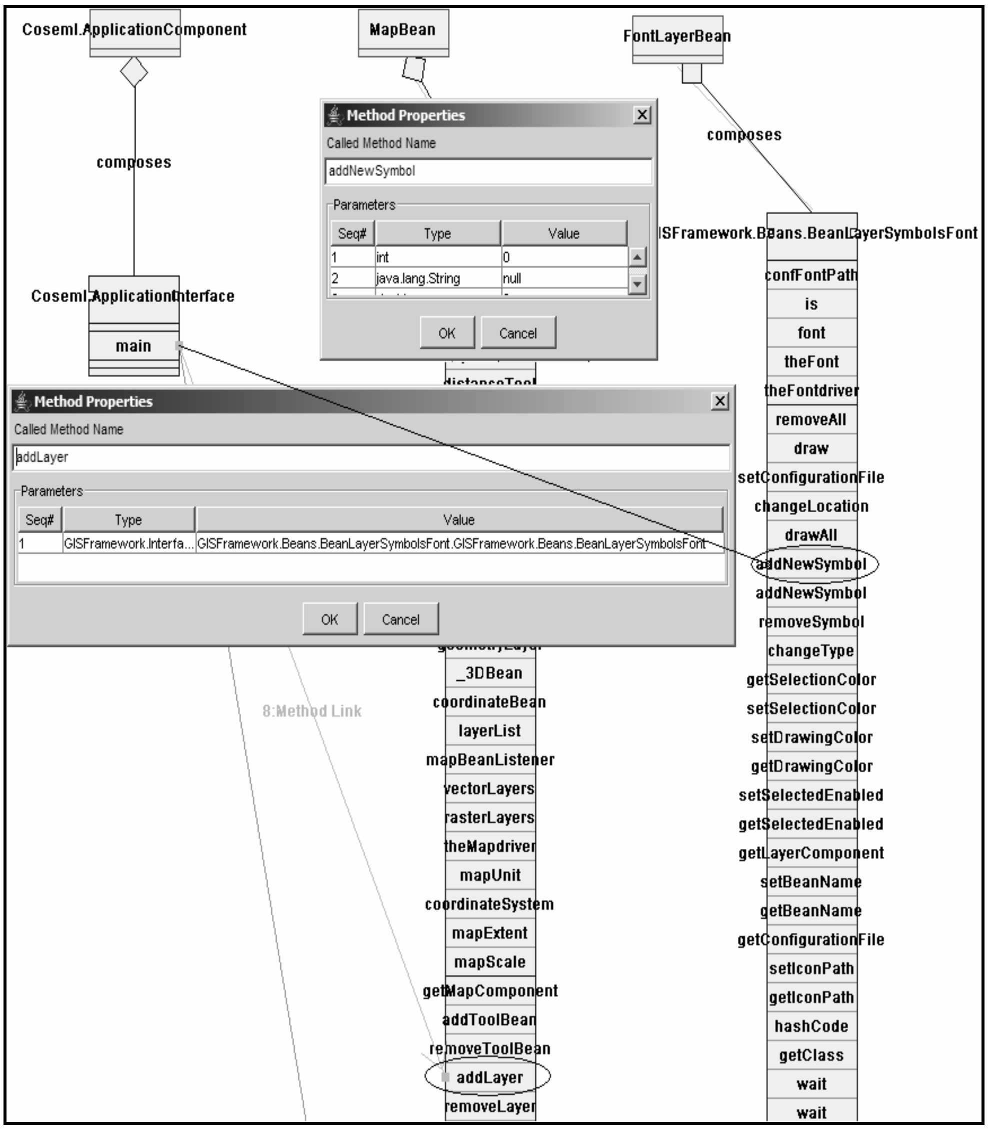
addLayer

Parameters

Seq#	Type	Value
1	GISFramework.Interfaces.Interf...	GISFramework.Beans.BeanLayerAnalysis.GISFramework.Beans.BeanLayerAnalysis

Figure 4.16 Representation of Relation 5

RELATION 6: Like most of the components the *BeanLayerSymbolFont*, the *BeanLayerSymbolGeometry*, the *BeanLayerSymbolDrawing* components must be added to the BeanMap component as a layer. To achieve this *addLayer* function of the *BeanMap* component must be called with an instance of a *symbol layer component*. Then a symbol can be added to this layer by using the *addNewSymbol* function with custom parameters. Representation of RELATION 6 is shown in **Figure 4.17.**


Method Properties

Called Method Name: addNewSymbol

Parameters

Seq#	Type	Value
1	int	0
2	java.lang.String	null

Method Properties

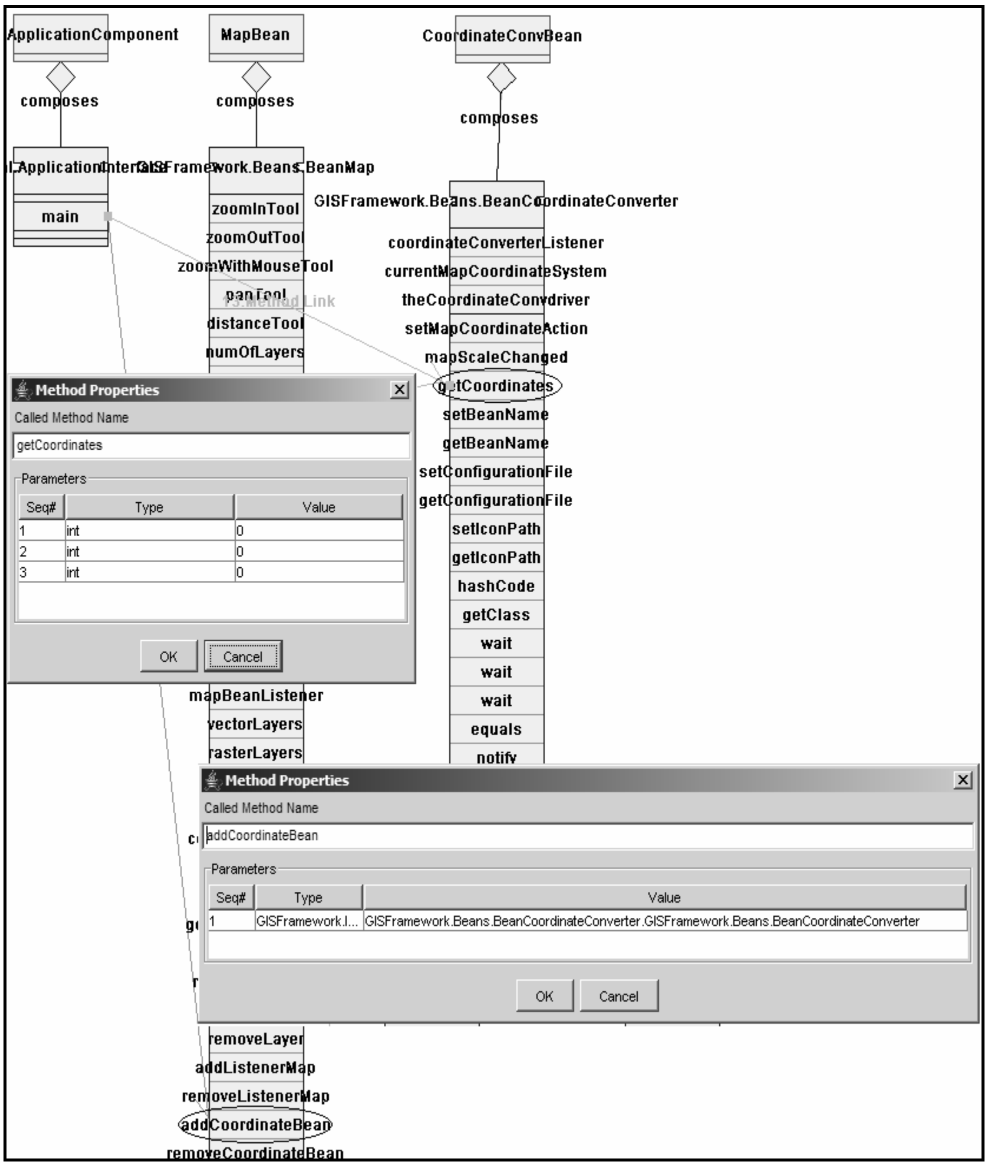
Called Method Name: addLayer

Parameters

Seq#	Type	Value
1	GISFramework.Interfa...	GISFramework.Beans.BeanLayerSymbolsFont.GISFramework.Beans.BeanLayerSymbolsFont

Figure 4.17 Representation of Relation 6

RELATION 7: *BeanCoordinateConverter* component is used for coordinate conversions in the GIS domain. This component must be added to the BeanMap component by using addLayer function of the BeanMap component. After the Coordinate Converter Bean is added to the Map Bean, it registers itself with the Map Bean and the Map Bean registers itself with the *BeanCoordinateConverter* automatically. Moreover, to achieve a coordinate conversion the *getCoordinates* function of the *BeanCoordinateConverter* component must be called with custom parameters. Representation of RELATION 7 is shown in **Figure 4.18.**



Called Method Name

getCoordinates

Parameters

Seq#	Type	Value
1	int	0
2	int	0
3	int	0

Called Method Name

addCoordinateBean

Parameters

Seq#	Type	Value
1	GISFramework.I...	GISFramework.Beans.BeanCoordinateConverter.GISFramework.Beans.BeanCoordinateConverter

Figure 4.18 Representation of Relation 7

The BeanMap component is the main component of the GIS domain. The user interaction in a GIS application is obtained by using this component. Other components mostly interact with this component. However, these interactions are not presented directly. Instead of direct function calls between components, components passed as parameter through function calls. Moreover, the interaction between components in the GIS domain is satisfied by using event mechanisms. The components are first added to the MapBean component, which implements different interfaces that are needed to manage other components. There are two different behaviours of these components. Some components fire events asynchronously from the MapBean like mouse events, the other components fire events in the case of a request come from the MapBean component.

After adding all of the components and defining the relations, the domain is saved with the name “GIS”. Then any application engineer can load and use this domain for custom application development. The framework will offer the default relations defined here automatically when application engineer uses any of the component resides in the created domain.

Same basic applications are developed by using the domain information created here. Every time the relations are defined straight, a running application is developed.

4.8. A Sample Domain

Although the approach defined here works for the GIS domain, another domain is needed to show that this approach can be applied to other domains. Therefore, a simple domain consisting two components is defined for demonstration purpose. This domain reflects the main idea of most of the applications. A database and a GUI component are included in this domain.

The components used for Test domain and their definitions are given below.

a. “*DatabaseBean*” component is used for database access. This component has the functionalities to connect to a given database and to acquire data from the connected database. Data in database relates to the customer information. Queries can be constructed and filtered with the name of the customer.

b. "*GUIBean*" component is the GUI part of this test domain. It has a text field and default functionalities inherited from the Swing library.

A relation document is prepared for the test domain to define and save the relations by using COSECASE tool. The relations that are defined for test domains and their representation in framework are given below.

RELATION 1: The implementation of the domain allows using to connection to different databases. The name of the database, the user name, and the password can be given as parameters to the *DatabaseBean* component. For this purpose first of all the database connection parameters can be set by using *setDatabaseParameters* function with the default parameters *Customers* as the name of database, "*sa*" as the name of user, and "*sa*" as the password of user.

RELATION 2: The name of the customer to query from the database must be set by using *setCustomerName* function. The name of the customer can vary across different applications so the parameter is given as "*Eray*" initially but can be changed at the time of custom application development.

RELATION 3: Finally, to display the customer info acquired from the database must be displayed in the *GUIBean's* text field. The text field property of the *GUIBean* must be loaded with the return value of the *getCustomerInfo* function of *DatabaseBean* component.

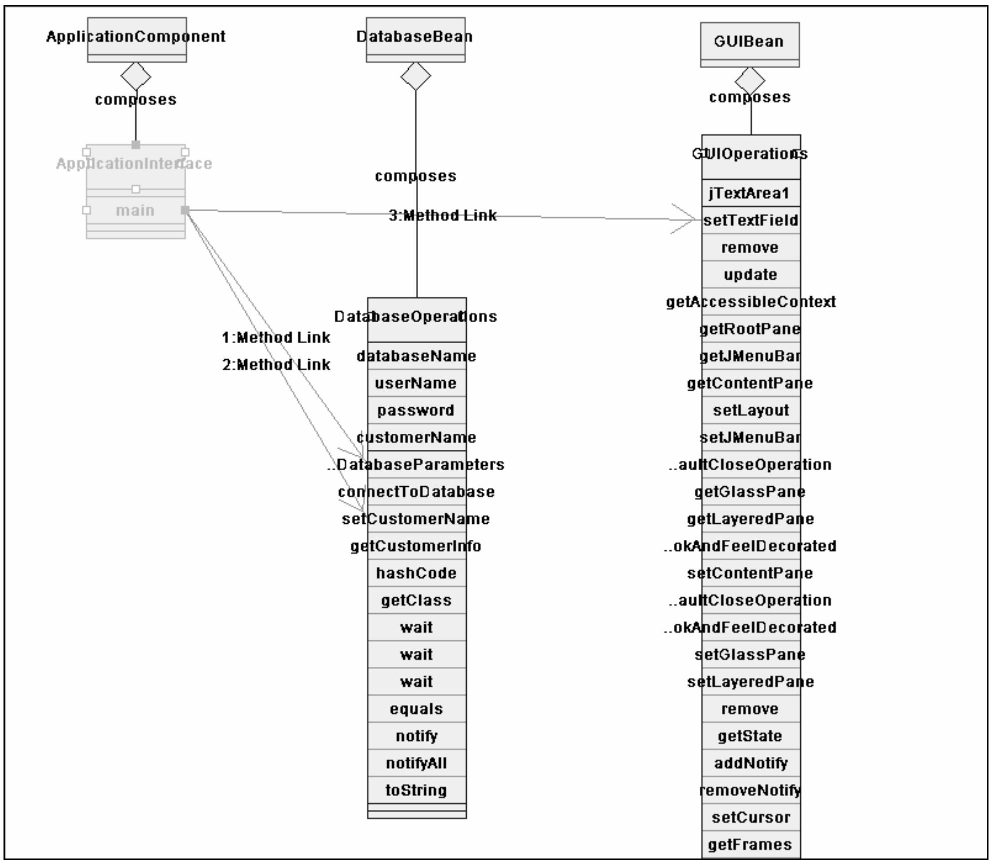
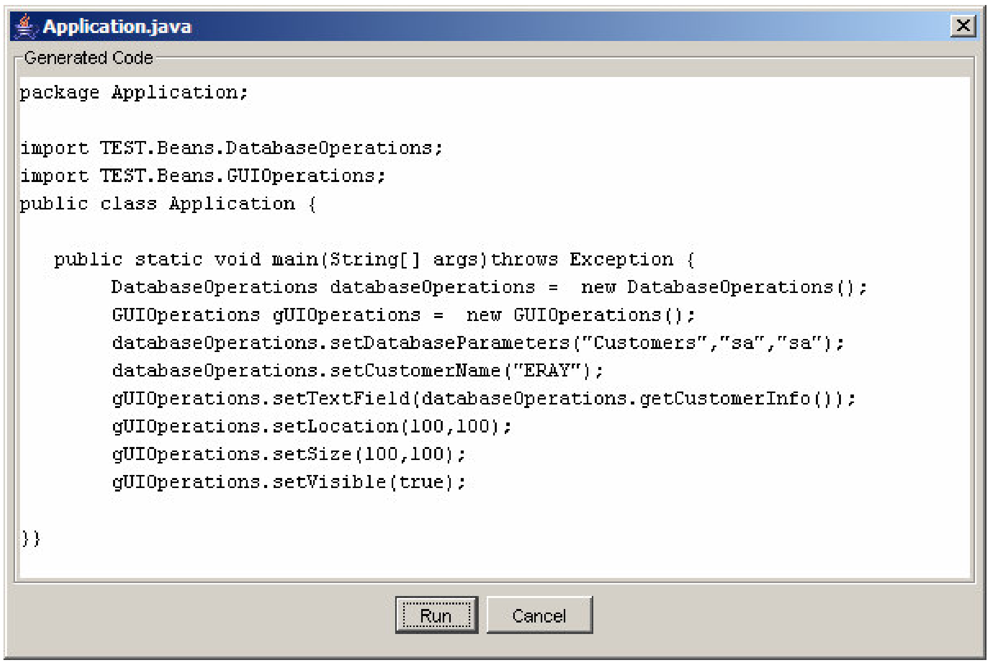


Figure 4.19 Representation of Default Relations in Test Domain

After adding two components and defining the relations, the domain is saved with the name “Test”. The defined relations for test domain are shown in **Figure 4.19**. Then an application is generated by loading the domain. This application has some more relations defined in the domain loading part. These relations are defined to set the location and size of the GUI frame. Then the source code is generated from defined relations and the application is run. The source code and the application generated by using the saved domain are shown in **Figure 4.20** and **Figure 4.21.**



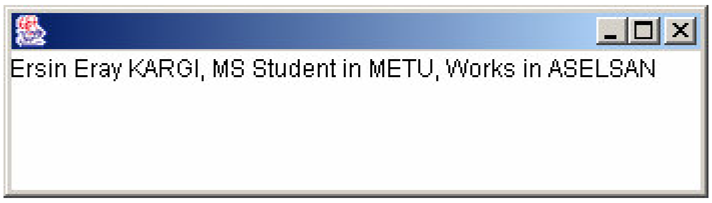
```
package Application;

import TEST.Beans.DatabaseOperations;
import TEST.Beans.GUIOperations;
public class Application {

   public static void main(String[] args)throws Exception {
        DatabaseOperations databaseOperations =  new DatabaseOperations();
        GUIOperations gUIOperations =  new GUIOperations();
        databaseOperations.setDatabaseParameters("Customers","sa","sa");
        databaseOperations.setCustomerName("ERAY");
        gUIOperations.setTextField(databaseOperations.getCustomerInfo());
        gUIOperations.setLocation(100,100);
        gUIOperations.setSize(100,100);
        gUIOperations.setVisible(true);

}}
```

Figure 4.20 The Source Code Generated in Test Domain



```
Ersin Eray KARGI, MS Student in METU, Works in ASELSAN
```

Figure 4.21 Application Generated from Source Code

4.9. Evaluation of Other Related Work

Code generation by using pre-fabricated components is an improving area for software engineering community. Since the benefits of developing applications by using tested, utilized program parts are high, this approach gains importance nowadays. A lot of efforts have been spent on this issue.

Different approaches are introduced in [35], [36], [41], and [42]. Our approach suggests a generic approach which can be used for individual domains. Also different approaches are implemented for application development by using components. But the framework approach goes one step further by offering default relations to help the application development. The GUI used here allows the component oriented design and the implementation despite similar attempts mostly concentrate on just putting the components together.

In some of these approaches an eXtended Markup Language (XML)-based file defines the construction of the component. But in our approach the components are used from a pre-constructed library and the construction of components is discovered by using the byte code. This gives us the advantage of saving time by automatically discovering the structure of a component. Rather than defining the structure of components in XML-based files, our approach offers defining the default relations in an XML-based file. Also our approach allows porting java beans into a component-oriented design. All of the approaches described in here do not allow porting some kind of pre-compiled code into a component-oriented design. Rather they start with defining the components in the component oriented world.

In [37], the approach used for integrating components has some similarities with our approach. But in this approach components are integrated with adapter components. The adapter components are not needed in our case. In our approach by using an application component the interaction between components is achieved by direct function calls. An advantage of using the adapter components is automatically type casting between compatible types. In our approach only the parameters in the same type are offered. But the casting is available by user definition.

Also the GIS Java Beans library is an output of [8]. In [8], steps of domain analysis for the GIS domain are introduced. The main idea discussed is, pass between jar files to Java Beans. And a sample demonstration part is implemented to show that the Jar files produced can be used for component-oriented approach.

One more benefit of our approach is that it acts as a bridge between the domain engineer and application engineer. By creating a domain, a domain engineer can transfer domain information to the application engineer easily. Also an application engineer can improve the domain by using his own relations. But of course this idea must be tested with future-use.

CHAPTER 5

CONCLUSION

A generic approach for an easier use of a domain is generated and implemented in this thesis. In addition, steps of domain engineering are applied for the GIS domain. The artefacts of this domain are used to validate the approach. Integrating java beans into COSECASE world was a cornerstone for this thesis. Since java beans are automatically integrated, there is no need to define the structure of components. All the methods, properties are automatically discovered inside the jar files.

Defining the structure for saving and offering relations and implementing it in COSECASE, constituted considerable effort. The approach allows users to define default relations and save these relations with the related components to form a domain. Then these default relations are offered to the user for custom application development. The structure is also defined suitable for code generation by allowing modifications the default relations offered by the framework. Demonstration of the approach in the GIS domain is also completed. In addition, a simple test domain is created for demonstration purpose. This domain is saved and used in the framework to show that our approach is generic and can be applied to other domain.

5.1. Work Conducted

In the beginning of this thesis, domain analysis is realized for the GIS domain. Some custom applications are inspected and knowledge from domain experts is collected. For the analysis part, FODA (Future Oriented Domain

Analysis) technique is used. Some studies for the domain analysis with FODA are also examined to be well informed with the technique.

After completing the domain analysis part, an approach to help with code generation by offering default relationships is developed. Although our demonstration domain is the GIS domain, this approach is developed as generic as possible. For the user interface part of the thesis, the COSECASE tool is used. Since this tool offers component oriented modelling the pass between java beans to the COSECASE environment is completed. At the early steps of the framework implementation, the full structure of java beans is ported into the COSECASE. This means all the classes in the beans displayed as components and their interfaces and relations are shown in the screen. However, to hide the unnecessary details of a component the framework is implemented with the package filtering ability. The name of package to be used is read from a configuration file to achieve a generic implementation. The interfaces that bean classes implement are shown as a different interface to achieve easy definition of relations.

Relations are defined between components. After developing some custom applications by using components, it is realized that the interactions mainly take place between the main application and components. The interactions between components are manipulated by the main application. So a default application component is added in the domain creation part to enable code generation.

The relations between components and the application component are saved by using XML. This text data is created by the application but there is no obstacle to define these relations outside the application and port into the application or manipulate it outside after creation.

To show that this approach presents an easy way for automatic code generation, a code generation part is also implemented in this thesis. For the code generation part, the need for defining the sequence of function calls appeared. A sequence number property is added to method links in the COSECASE tool. This also helps the modelling part to define the sequence of relations between components. Then for a working and compiling code, the need for defining the parameters arises. To help define parameters in the relations a user-friendly approach is implemented. This implementation helps by offering functions or objects to parameters by pairing the types.

For the code generation part, the user can extend the default relations by adding his own relations after domain is loaded. This part is implemented to get a running java code. Instances of components are created and function calls are written in the given sequence by using the defined parameters. This approach has demonstrated success for simple GIS applications.

While implementing the framework user-friendliness was an important point. Application generation can be conducted with the minimum user interaction since some default relations are offered by the framework.

The approach offered here, opens the way of easy code generation for component-oriented software engineering. Since the sequence of function calls and the definitions of parameters can be achieved by using the GUI, there is no difficulty for code generation.

Our implementation is firstly demonstrated in GIS domain and showed success. In addition, to show that the approach can be applied to any domain, a sample domain is created and this domain is used for application development in the implemented framework.

All the code written into the COSECASE tool can be used easily by other ideas like reverse engineering. Some improvements and corrections are completed for the COSECASE tool. The additions and corrections completed for the COSECASE tool are listed below.

- A new package is added to manage the importing of java beans to COSECASE for domain operations. In addition, the data structures to save and load the domain are covered in this package.

- Some classes are expanded to save relations defined for a domain in an XML file. These classes allow XML based serialization.

- A new class is added to allow domain operations like *creating domain*, *saving domain*, *loading domain, modifying domain,* and *generating domain code*.

- A new class is added to check for jar files to get the packages, classes, and details of these classes. This class is implemented by using *Reflection API, which* allows getting the structure of a compiled java class.

- Classes that are used to hold links like method links and event links are updated. This update allowed these links to be saved in a data structure for saving in an XML file.

- New classes are added to control the parameter definitions of method links. These classes also include the user-interface part for allowing the user to define the values for parameters. While defining the values of parameters, the framework offers some functions and properties for selection by using the information acquired from used components.

- Some classes are updated for correction in deletion of nodes. Before the corrections when nodes are deleted, the relations defined for this node are not deleted and cannot be used. With the correction when a node is deleted the relations defined for this node are also deleted.

- Some classes are updated to allow display of sequence numbers in method links.

- COSECASE's main frame is updated to add domain operations menu. This menu allows user to attain domain operations such as creating domain, saving domain, modifying domain, and loading domain.

- A new dialog class is added to show the generated code and to allow user's changing the code.

5.2. Comments

Although demonstration in two different domains is completed, we cannot claim that this approach can be applied to all of the domains. Some experiments in real world must be conducted. The tests by using the framework showed that custom application development is faster with this framework. Of course, these tests are conducted for small applications and there is no guaranty that it will be faster for enterprise applications.

5.3. Future Work

This thesis offers a generic design for offering default relations and code generation. Code generation part of the thesis can be extended. The implementation in COSECASE allows one instance of a component for an application. If more than one instances of the same component are needed by an application, this functionality might be added. Control flow is currently managed by the main application. This results in a centralized control structure. A distributed management of the control flow can be added as a future capability.

In addition, this work opens a way to reverse engineering in component-oriented approach. In the early stages of this thesis, the java bean components were fully reverse engineered and transferred into COSECASE. This reverse engineering part does not cover the interactions inside a bean since the source code does not exist. The infrastructure partially built in this thesis can be expanded to a fully capable reverse-engineering tool.

Message ordering was implemented but this capability should be supported by adding a "sequence diagram" capability to the tool.

Although our approach is applied to two domains, some more domains can be analyzed and transferred into the framework for further validation and for finding deficiencies.

REFERENCES

[1] Hassan Gomaa,"Developing Software Lines: Why Bother?", Addsison Wesley, 2004

[2] Kimberly Jordan, MJY Team, George Mason University, "Software reuse term paper for the MJY Team software risk management www site", SWSE 625, April 1997

[3] Bertrand Meyer, "Eifell: Reusability and Reliability", *IEEE Tutorial Software Reuse: Emerging Technology* edited by Will Tracz, IEEE Computer Society Press, IEEE Catalog #EH0278-2, ISBN 0-8186-0846-3, pp. 216-228, 1990.

[4] Maarit Harsu, "A Survey on Domain Engineering", *International Workshop on Component-Based Software Engineering*, May 1999

[5] Robert Krut, Nathan Zalman, "Domain AnalysisWorkshop Report for the Automated Prompt Response System Domain", *International Workshop on Component-Based Software Engineering*, May 1999

[6] Carnegie Mellon University Software Engineering Institute (SEI), "Domain Engineering", *http://www.sei.cmu.edu/domin_engineering/domain_emg.html,* March 2005.

[7] D.L. Parnas, P.C. Clements, and D.M. Weiss, "Enhancing Reusability with Information Hiding", *IEEE Tutorial Software Reusability* edited by Peter Freeman, IEEE Computer Society Press, IEEE Catalog #EH0256-8, ISBN 0-8186-0750-5, pp. 83-90, 1987.

[8] Ebru Özdoğru, "A GIS Domain Framework utilizing JAR libraries as components", *M.S. Thesis*, Middle East Technical University, May 2005.

[9] Clements Szyperski, "Component Software Beyond Object Oriented Programming", *Addison-Wesley*, 1998.

[9] Ilka Philippow, Detlef Streitferdt, Matthias Riebisch, "Design Pattern Recovery in Architectures for Supporting Product Line Development and Application", Modelling Variability for Object-Oriented Product Lines edited by M. Riebisch, J. O. Coplien, D, Streitferdt (Eds.). BookOnDemand Publ. Co., Norderstedt, pp. 42-57, 2003.

[10] "Software Reuse, Major Issues Need To Be Resolved Before Benefits Can Be Achieved", *United States General Accounting Office*, January 1993.

[11] Ruben Prieto Diaz, Gerald A. Jones, "Breathing New Life into Old Software", *IEEE Tutorial Software Reuse: Emerging Technology* edited by Will Tracz, IEEE Computer Society Press, IEEE Catalog #EH0278-2, ISBN 0-8186-0846-3, pp. 152-160, 1990.

[12] Vedat Bayar, A Process Model for Component Oriented Software Development, *M.S. Thesis*, Middle East Technical University, November 2001.

[13] Aydın Kara, A Graphical Editor for Component Oriented Modeling, *M.S. Thesis*, Middle East Technical University, April 2001.

[14] Jim Q. Ning, "A Component Model Proposal", *International Workshop on Component-Based Software Engineering*, pp. 13-16, Los Angeles, CA, USA, 17-18 May 1999.

[15] Kurt C. Wallnau, "On Software Components and Commercial ("COTS") Software", *International Workshop on Component-Based Software Engineering*, pp. 213-218, Los Angeles, CA, USA, 17-18 May 1999.

[16] Tricia Oberndorf, Lisa Brownsword, Carol A. Sledge, "An Activity Framework for COTS-Based Systems", *Technical Report, Software Engineering Institute*, October 2000.

[17] Cecilia Albert, Lisa Brownsword, "Evolutionary Process for Integrating COTS-Based Systems (EPIC): An Overview", *Technical Report, Software Engineering Institute*, July 2002.

[18] Gail E. Kaiser, David Garlan "Melding Software Systems from Reusable Building Blocks", *IEEE Tutorial Software Reuse: Emerging Technology*

edited by Will Tracz, IEEE Computer Society Press, IEEE Catalog #EH0278-2, ISBN 0-8186-0846-3, pp. 267-274, 1990.

[19] Geral Jones, “Methodology/Environment Support for Reusability”, *IEEE Tutorial Software Reuse: Emerging Technology* edited by Will Tracz, IEEE Computer Society Press, IEEE Catalog #EH0278-2, ISBN 0-8186-0846-3, pp. 190-193, 1990.

[20] Oh-Cheon Kwon, Seok-Jin Yoon and Gyu-Sang Shin, “Component-Based Development Environment: An Integrated Model of Object-Oriented Techniques and Other Technologies”, *International Workshop on Component-Based Software Engineering*, pp. 47-53, Los Angeles, CA, USA, 17-18 May 1999.

[21] D. Ansorge, K. Bergner, B. Deifel, N. Hawlitzky, C. Maier, B. Paech, A. Rausch, M. Sihling, V. Thurner, S. Vogel, “Managing Componentware Development –Software Reuse and the V-Modell Process”, *Proceedings of the 11th International Conference on Advanced Information Systems Engineering*, pp 134-148, 14-18 June 1999.

[22] K. Czarnecki and U. Eisenecker, “Generative Programming: Methods, Techniques, and Applications”, *Addison-Wesley* 1999.

[23] Rubén Prieto-Díaz, “DOMAIN ANALYSIS: AN INTRODUCTION”, *ACM SIGSOFT Software Engineering Notes*, pp. 47-54, April 1990.

[24] Rubén Prieto-Díaz, “Domain Analysis for Reusability”, *Proceedings of COMPSAC’87*, pp. 23-29, 1987.

[25] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, A. Spencer Peterson, “Feature-Oriented Domain Analysis (FODA) Feasibility Study”, *Technical Report, Software Engineering Institute*, November 1990.

[26] Sholom G. Cohen, Jay L. Stanley, Jr., A. Spencer Peterson, Robert W. Krut, Jr., “Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain”, *Technical Report, Software Engineering Institute*, June 1992.

[27] Robert Kurt, Nathan Zalman, "Domain Analysis Workshop Report for the Automated Prompt Response System Domain", *Technical Report, Software Engineering Institute*, May 1996.

[28] John D. McGregor, "The Evolution of Product Line Assets", *Technical Report, Software Engineering Institute*, June 2003.

[29] Paul C. Clements, Linda M. Northrop, "Salion, Inc.: A Software Product Line Case Study", *Technical Report, Software Engineering Institute*, November 2002.

[30] Rosario Girardi and Carla Gomes de Faria, "A Generic Ontology for the Specification of Domain Models", *Proceedings of 1st Workshop Component Engineering Methodology*, pp. 41-50, 24 September 2003, Erfurt, Germany.

[31] Thomas Cleenewerck, "Component-based DSL Development", *Proceedings of GPCE03 Conference*, Lecture Notes in Computer Science 2830, pp. 245–264, Springer- Verlag, 2003.

[32] Mark Johnson, "A Walking tour of Java Beans by Mark Johnson", *Java World http://www.javaworld.com/javaworld/jw-08-1997/jw-08-beans.html*, February 2005.

[33] Laurance Vanhelsuwe, "Mastering in Java", http:/*/www.javaolympus.com/freebooks/FreeJavaBooks.jsp,* October 2004.

[34] Esri GIS and Mapping Software, "Geography Matters*", White Paper http://www.esri.com/library/whitepapers/pdfs/geomatte.pdf*, September 2002.

[35] Vaclav Cechticky, Philippe Chevalley, Alessandro Pasetti, Walter Schaufelberger, "A Generative Approach to Framework Instantiation", *Proceedings of the 2nd International Conference on Generative Programming and Component Engineering*, pp. 267–286, Springer- Verlag, September 2003.

[36] Theo Dirk Meijler, Serge Demeyer, Robert Engel, "Automated Support for Software Development with Frameworks", *Proceedings of the 6th European Conference held jointly with the 5th ACM SIGSOFT International*

Symposium on Foundations of Software Engineering, ACM SIGSOFT Software Engineering Notes, Volume 22 Issue 6, pp. 123–127, November 1997.

[37] L.K. Jololian, F.J. Kurfess, M.M. Tanik, “Data, Functıon, And Control As Elements Of Component-Integratıon”, *Integrated Design and Process Technology,* IDPT-2003, pp. 194–204, June 2003.

[38] Vaclav Cechticky, Philippe Chevalley, Alessandro Pasetti, and Walter chaufelberger, “A Generative Approach to Framework Instantiation”, GPCE 2003, LNCS 2830, pp. 267–286, 2003.

[39] Jeff Gray, Ted Bapty, Sandeep Neema, Douglas C. Schmidt, Aniruddha Gokhale, and Balachandran Natarajan, “An Approach for Supporting Aspect-Oriented Domain Modeling”, Birmingham AL 35294-1170, http://www.isis.vanderbilt.edu

[40] Bernhard Schaetz, Alexander Pretschner, Franz Huber, Jan, “Model-Based Development, FAST product-line architecture process

[41] C. L. Heitmeyer, R. Jeffords, B. Labaw, “Automated Consistency Checking of Requirements”, ACM TOSEM 5(3):231-261. 1996.

[42] K. Bergner, A. Rausch, M. Sihling, “Componentware—the Big Picture.”, *20th ICSEWorkshop on Component-based Software Engineering*, 1998

[43] M. Zaremski, J. M. Wing, “Specification Matching of Software Components”, ACM TOSEM. 6(4):33-369. 1997.