International Journal of Software Engineering and Its Applications
Vol. 7, No. 3, May, 2013

Designing of Framework for Mobile Applications Assets Management

Haeng-Kon Kim

*School of IT Engineering, Catholic University of Daegu,
Kyungsan-si, Kyungbuk, 712-702, Korea
hangkon@cu.ac.kr*

Abstract

Framework is a new programming paradigm which makes it possible to reuse the software architecture. Framework defines the wired-in interactions the default behavior of common applications as well as well defined components. Framework even provides a customizing mechanism. In this paper, we have developed an easy-to-customize framework to enhance the reusability. The aim of our customizing environment is to store the meta information in the framework repository to support the reuse of our MAM (Mobile Assets Managements) framework. Our framework repository contains the structural characteristics and the behavior characteristics of components, as well as interactions among the components. In this paper, the framework-customizing toolkit for mobile applications has been constructed. It helps the process of the framework reuse to be systematic by supporting the following: (1) understanding the framework itself, (2) refining the components, (3) changing the related components automatically, and (4) configuring the mobile applications.

Keywords: Mobile Applications, framework, customizing toolkit, software reuse

1. Introduction

The software library consists of well-defined components. The reusers find appropriate components from the library and put them with newly defined code into one program. If needed, reusers modify the components to meet the new requirements or to adapt with others. However, the library reuse has some limitations since the software architecture can't be reused. It is very difficult to reuse the design concepts and domain knowledge, while these are much more important things in reuse [2]. In addition, it is not easy to find proper components and to integrate them with others since it requires a deep understanding of domain. The reason why we need to reuse is not that we have no programming skills, but that the analysis and understanding of the requirements is difficult.

Framework is a new programming paradigm which makes it possible to reuse the software architecture. Framework defines the wired-in interactions the default behavior of common applications as well as well defined components [9, 11]. Framework even provides a customizing mechanism. Therefore, we don't need to write a code for integrating modules. Instead, we need to write the codes to be called from the framework. Framework is a promising way to understand the whole application domain and to reuse much large-scaled software parts.

This is the reason why we explored to develop a MAM (Mobile Product Contents Managements) framework. After conducting domain analysis, we defined a large amount of domain classes and their interactions between them so that the framework proposed in this study should provide basic and common functionality of a MAM system. So, when the requirements are not too specific, our framework can be a good application. In this case where a company has specific requirements, the application developer should customize our framework. When we develop new application by reusing the framework, we can maximize the reusability. However, the framework reuse is still a difficult process. Our framework defines more than two hundred domain classes and even more user interface classes. Therefore, the reusers are confronted with the difficulties in understanding the framework itself and identifying which components are to be redefined.

In this research, we have developed an easy-to-customize framework to enhance the reusability. The aim of our customizing environment is to store the meta information in the framework repository to support the reuse of our MAM framework. Our framework repository contains the structural characteristics and the behavior characteristics of components, as well as interactions among the components. The application developers can use our framework environment for (1) understanding the framework in the view of static structure and dynamic behavior, (2) extending or redefining the domain components, (3) extending or replacing the user interface classes, and (4) setting up the configuration of new applications. This paper discusses the design and the detailed structure of the customizing toolkit. In the section two, we survey the characteristics of the framework and explain our views to the customization of the framework. The design concepts and the architecture of our customizing toolkit are described in the section three. The state of the art of our implementation and conclusion are represented in section four and five, respectively.

2. Related Works

In this section, we describe the definitions of the general framework and framework customization. We will also define the scopes and characteristics of our MAM framework.

2.1. Framework Definition

As stated earlier, the framework is a new programming paradigm that can ensure software reuse. Framework is a template for working programs since it is a set of classes and interactions among the classes that embodies an abstract design for solutions to a family of related problems in one domain [9, 12]. Frameworks are not simply collections of classes but also wired-in interactions between the classes that provide an infrastructure and architecture for the developers. To provide an infrastructure, we define the template classes that are the common and general classes to a group of related applications, and define the hot spots to a specific application. These hot spots are redefined or extended by the specific application developer later. It can be said that applications are successful when they satisfy the requirements of the customer. However, framework can be said to be successful when it can be easily customized for several different requirements. Therefore, designing a framework is a little different from designing an application. We need more general and stronger abstract design concepts for designing a framework.

2.2. Framework Category

We can categorize the framework by problem domains: application, domain, and system framework [9, 16]. Application frameworks abstract the concepts applicable to a wide variety of programs. It doesn’t have any real user functionality and has no program template for concrete application. Application frameworks are generally designed for horizontal markets such as GUI. Domain framework encapsulates classes in a particular problem domain and is developed for vertical markets. System framework provides the common functionality related with low-level ones such as hardware devices. It allows the system to be extended to add new kinds of hardware devices.

In addition, the framework can be customized in two ways: architecture-driven and data-driven framework [12]. In the architecture-driven the framework, framework can be customized by inheritance. It is also referred to as the white-box framework. White-box framework doesn't provide whole classes for entire applications but it provides just common structure. We can extend or refine the behaviors of the components by inheriting and overriding them to complete. Data-driven frameworks rely on object composition for customization. Clients define the application by using different compositions of objects. It is also called black-box framework, and consists of classes for the complete domain. It defines the abstract parts and candidate parts for filling the hot spots. We tend to think composition is easier than inheritance. However, the development of the white-box framework is said to be easier than the development of the black-box framework.

2.3. Advantages of Frameworks

The benefit of the framework is that it makes it possible to reuse a higher level of code and design than other reuse approaches. This advantage enables the creation or reconstruction of an application, which were very difficult and limited in the library-based reuse. The first benefit is that the framework provides the infrastructure and the architectural guidance so that an application developer can easily understand the domain and just write specific codes for new applications. The framework can reduce coding, testing, and debugging efforts. The framework itself also has a mechanism for extending functionality [9, 11]. In the framework, hot spots are defined for further deployment. Abstract classes in the framework provide the well-defined interfaces to map with several different implementations for each application. Deferred implementation can be extended without changing the existing framework. As a result, the framework reduces the efforts of maintenance.

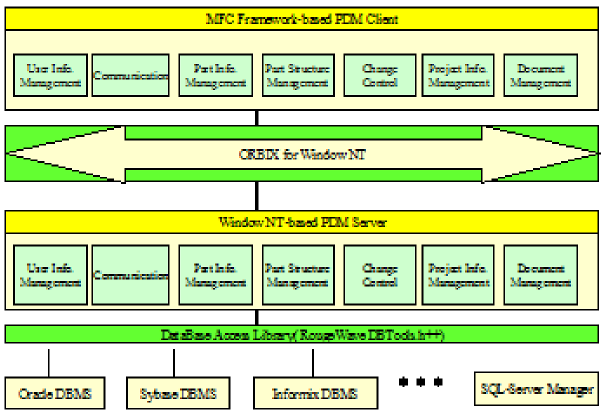


Figure 1. Our Target Environment of PDM System

2.4. Our Approach to the Framework

The number of people who agree with the fact that object-oriented technology (OOT) can provide productivity to software development process has been increased. However, many developers still hesitate to move to OOT since they need a significant amount of time and effort to apply OOT to their practical problems. We believe that the framework based on OOT could deliver more powerful and natural solutions to application developers who doesn't understand what OOT is, but has great knowledge of the domain.

Our framework is a domain framework and the target domain is MAM (Mobile Assets Management). The framework proposed in this study is called NextMAM which enables developers to leverage OOT with the framework that supports the MAM domain. NextMAM provides well-defined customizing mechanisms for framework reuse as well as domain classes and domain processes. Until now, NextMAM has a white box framework. There are still hot spots to be filled for specific requirements by application developers. In some parts, NextMAM shows the characteristics of the black box framework. Developers just need to decide which combinations of our candidate domain concepts are required for their application. The purposes of NextMAM are twofold: (1) provides the well-defined MAM framework to easily create new MAM applications, (2) provides an integrated development environment for the framework reuse. In this paper, major discussions are focused on the second purpose.

2.5. Explanation of Next MAM

In this section, we will discuss how NextMAM has been developed in order to explain our customizing environment. Figure 1 shows the scope and architectural features of NextMAM. NextMAM is a Windows NT-based distributed framework and uses Orbix for system interoperability and applies the RougeWave library for Database interoperability. NextMAM has been constructed following the framework development process: domain analysis, system analysis, applying and writing design patterns, and implementing and verifying a framework for their extendability and reusability.

We understand MAM domain the investigation of the literature about the MAM domain has been conducted through existing applications and general references such as Metaphase, Matrix, Work Manager and MAM buyer’s guide. In addition, existing MAM systems were analyzed to capture the basic functionality.

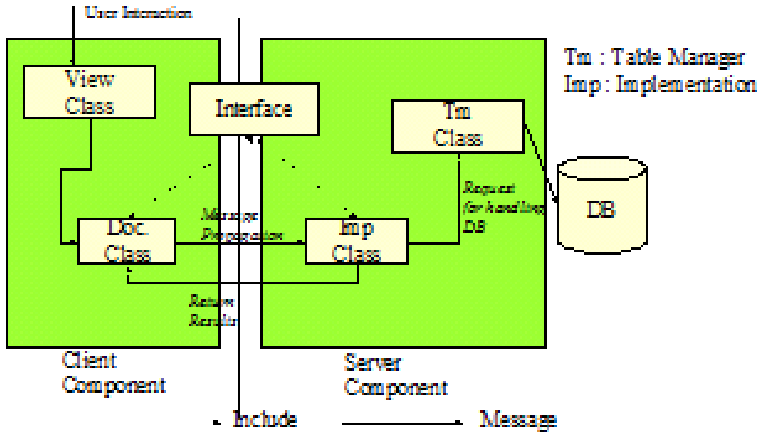


Figure 2. Our Strategy for Module Structure

To develop the framework, we analyzed several MAM systems, which had been used at different companies. Through this process, we identified which parts were common across the different customers and which parts should be specific to each customer. We described the MAM domain as an abstract and integrated business object model. This business model defines three important abstract classes: MAMData, MAMBehavior, and MAMRelation. MAMData is a data-centric object whose subclasses are part, mail, user, counterpart and material among others.

It contains many attributes to describe the item managed and processed in the MAM system. Its behaviors are kept simple, which are related with manipulating the attributes, setting or retrieving. MAMData doesn't know the complex processes such as validation, revise, and transfer. These classes should be the subclasses of MAMBehavior that define the behaviors to manipulate the MAMData. So MAMBehavior is referred to as a process-centric object whose attributes are needed just for its behavior not for its own characteristics. Of course, one MAMData has many associations with MAMBehavior for their processing. MAMRelation is an abstract class whose derived classes are Binary, Ternary, AttachTo, NextTo, and AssignWith, and so on. This abstract class knows how to handle the relationships among the classes and the derived attributes from the association. These three abstract classes enable NextMAM to extend the business concepts without modifying other parts.

NextMAM compromises three other frameworks: MFC framework for user interface, Orbix for system interoperability, and Rougwave library for persistent service. This made it difficult to develop NextMAM to be an independent and general framework. Our UIDP programming model is a solution to make the coupling between frameworks low. This model defines four kinds of independent classes: (1) user interface class, (2) interface class between client and server, (3) domain class, and (4) persistent class. Each class has distinct responsibilities: (1) the user interface class and view class are responsible for handling user interactions, (2) interface class, interface defined in IDL file, handles the interface between the Imp class, the domain class, is client and server components. (3) the Imp class, the domain class, is responsible for processing the business concepts. Actually, MAMData, MAMBehavior and MAMRelation belong to Imp class, (4) persistent class knows how to connect with and handle databases. Figure 2 shows our UIDP programming model. Currently we are implementing NextMAM with 228 domain classes and 3 service objects: workflow management, relationship management, and transaction management.

3. Desing of Customizing Toolkit

As mentioned earlier, the framework can be a promising way to reuse the higher design concepts and code. The problem is that framework reuse is not an easy process. Several obstacles make our customers deploy NextMAM to their application. The first one is that NextMAM defines such a large amount of domain classes that application developers encounter some problems in understanding a framework. It means they have some difficulties in identifying what parts are not fitting with their requirements and are to be customized. As we described earlier, we combined several commercial frameworks with our domain framework. It requires customers to know how to customize other frameworks or how to extend them with new frameworks. This is the second obstacle. The last one is that the framework has no integrated visual environment for its customizing. What they define is a natural mechanism for extending the framework. These are why we have started to develop an integrated environment for customizing. We have overcome these obstacles with the following principles:

(1) We must visualize the framework in terms of the behaviors of components as well as the structure of the framework. Understanding the framework with our visualization tool is the first step to customizing.

(2) For decoupling the existing framework and our domain framework, we extended the UIDP model by applying a mapper pattern whose intent is to decouple the classes defined in different abstract layers. We have three kinds of mapper for the UIDP. First one is for between the user interface and the domain class. UI/Domain mapper is responsible for

delegating user events to domain classes. The second one is for between client and server components. C/S mapper defines which client component is associated with the server components. The last one is a mapper between the domain and the persistent class. Domain/DB mapper is responsible for schema mapping since it is based on the relational database.

(3) Understanding, customizing, and generating an application can be done in an integrated customizing environment.

3.1. Customizing Scenario

Figure 3 shows the context of our customizing toolkit. The application developer can extend the domain classes by inheriting the existing one defined in the framework and also can define new domain classes by inheriting the MAMData abstract class. If necessary, user can extend or replace the user interface classes or simply modify the appearances of the existing user interface. In consequence of the customization of a business model, we can modify the database schema or characteristics of fields defined in the table. processes, our customizing toolkit has been developed.

(1) Understanding the Framework As we described in the earlier section, the framework can be an application by itself, and consists of well-defined components. What we can refine are these well-defined components to meet for a specific requirement. To accomplish this, we should identify which components are to be extended or replaced with new one. The framework is so large that it is not an easy task to identify the components to be customized. Class hierarchy and such a large class diagram can't help us easily understand the framework. To understand the framework, we should identify the default behaviors of the framework and components as well as the structures of them. For a more complete understanding, we mapped the behavioral characteristics with static features. This allows that if you discover some behaviors to be refined, then you could go to a model corresponding to those behaviors.

(2) Extending or Redefining the Behaviors of a Component

When the behaviors of components are not satisfied with current requirements, we should customize the framework. In such a case we should redefine the implementation of components. Of course, what we can refine are hot spots defined in the framework. As a result, we inherit the identified components and define a new implementation for the hot spots.

(3) Extending the Business Concepts

Business concepts need to be extended when we identify new business objects. In NextMAM, many important and common business objects of MAM domain are defined. Nonetheless, an every application can require a few different concepts. This means that, the same business concepts, or even the same user can have different structures and different features. A workCategory might be required in one application. However, in other applications, it might not be used. This is why we should customize the existing components for defining the new business concepts. In NextMAM, we defined the MAMBehavior class as an abstract class for process-centric objects.

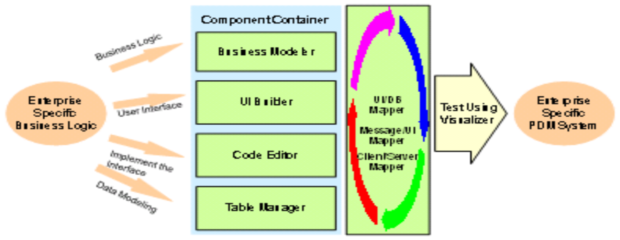


Figure 3. The Context of our Customizing Toolkit

When we should define the new process or new strategy of an application, we can extend the MAMBehavior or its subclasses. It is different from extending the behaviors of components. Extending the business concepts requires the creating or customizing of the related user interface and database scheme.

(4) Extending, Redefining and Replacing the User Interface

In many cases, the reason why we need the customization is that users tend to like to have a different 'look & feel'. According to the user requirements, the appearances of the user interface can be updated without changing the domain model. Of course, a new user interface can be defined for already existing domain concepts.

3.2. Framework Repository

For framework customization, we constructed the framework repository separately from the framework itself. Our framework repository defines the meta-information and meta-model for the NextMAM framework. The structure of the framework repository is shown in Figure 4. In the repository, we store four kinds of information: MAM framework model, its meta-model, mapping information, and the framework itself as a default application. Domain model, user interface model, interface model, and persistent model consist of the MAM framework model. We applied our UIDP model to construct the repository. Each model is defined with the object model and object collaboration diagram. For some dynamic components of the domain model, the dynamic model is also defined. These models are the results of NextMAM development process and are used to understand and extend the framework. The meta-information is needed for these MAM models to create new application

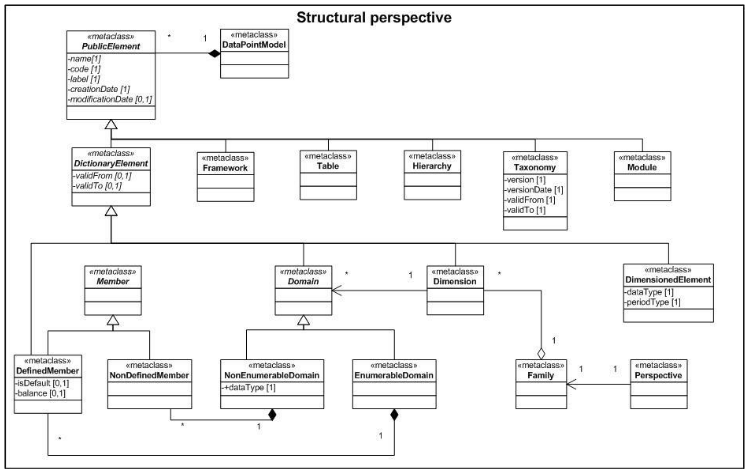


Figure 4. Structural Meta Model for MAM

as a result of customizing the framework. The important information of each model is identified and stored. For example, we define the meta-information for persistent classes as TableName, Name, Data_type, Length, Precision, Scale, Null_Allowed, and Is_primary. Table 1 shows the detail scheme for meta-information as in Figure 4.

One of the most important goals of our customizing environment is that the application developer can easily customize a model without considering the related other models. When a model is changed, our customizing toolkit adjusts the related models automatically using the mapping information defined in the repository. The framework repository contains the framework itself as a default application and a basis for customizing. Upon completion of the customizing process, our toolkit generates the new application based on the framework. Figure 5 show the Module Structure and Association with Repository.

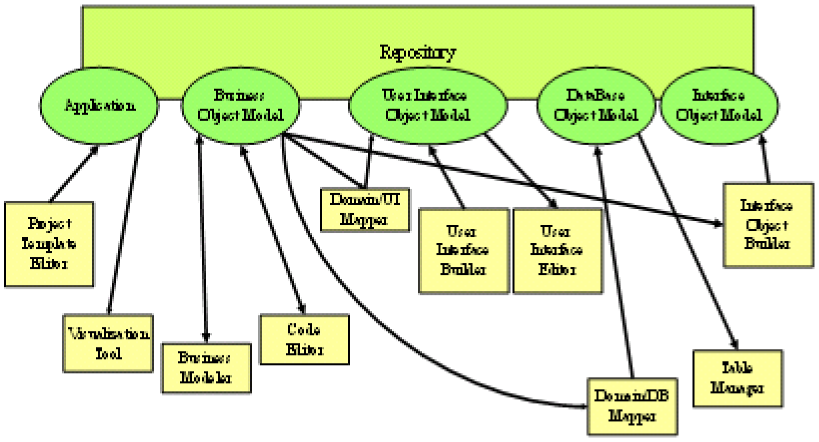


Figure 5. Module Structure and Association with Repository

3.3. Identification and Role Model Creation

Figure 5 shows the modules of our customizing toolkit. The major modules of the toolkit include the Business Modeler, the Table Manager, the UI Editor, the Code Editor, and the Visualizer. Figure 6 demonstrates the system structure and working flow among each module. It shows the structure of modules and their association with the repository.

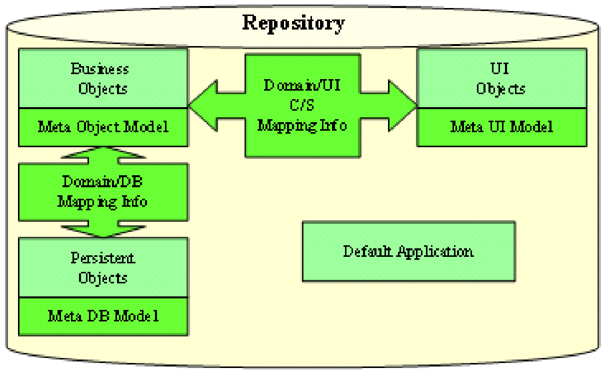


Figure 6. The Structure of the Framework Repository

4. Desing of Customizing Toolkit

We have completed the development of Toolkit Version 1.0 based on the framework repository. We developed the modules defined in the earlier sections: Business Modeler, UI Editor, Code Editor, Table Manager, Visualizer, IDL generator, Code Generator, Schema Mapper, and Component Mapper, etc. The running environment of the toolkit is shown in Figure 7 and the explanation of important modules is as follows.

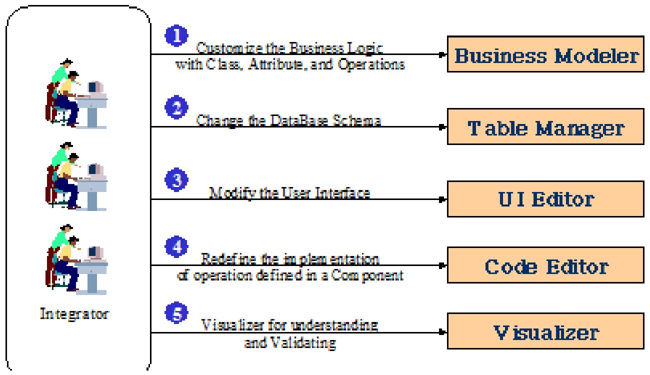

Integrator
1 Customize the Business Logic with Class, Attribute, and Operations
Business Modeler
2 Change the DataBase Schema
Table Manager
3 Modify the User Interface
UI Editor
4 Redefine the implementation of operation defined in a Component
Code Editor
5 Visualizer for understanding and Validating
Visualizer

Figure 7. The Customizing Processy

4.1. Business Modeler

Application developers define or redefine the business concepts with business modeler. Our business modeler has similar functionality with other object modeling tools. It can define the objects with attributes and operations, and define the relationship such as aggregation, inheritance, and association. While defining the object, application developers can define or redefine the mapping information with the mapper. The mapping information can be defined when the attributes and operations are defined. Attributes are defined with their own characteristics and mapping information. Mapping information is defined for user interface class and persistent class. To define the new behavior, a code editor is provided. The IDL generator produces the interface class between client and server components. The code generator also generates the server component based on the customization. It means that new subclass which inherits components is defined, and internally complied and plugged into the new application. Until now, we needed the compilation of a newly defined code. Finally, our toolkit registers newly defined server components into the implementation repository.

Communication does not only take place between agents, but can also occur between an agent and its environment. Agents can be implemented with sensors and effectors to allow the agent to interact with its surroundings. A less direct model of agent communication has to do with one agent effecting a change on the environment and other agents perceiving this change and acting accordingly. Instead of sending a message directly to another agent, an agent may use it effectors to change the environment in a way that can be perceived and interpreted by the sensors of other agents. This method is not as effective as message communication since it assumes that all agents will interpret this environmental change in the same manner. However, it does offer a significant advantage if message passing capability between agents is lost but coordination between agents is still required. This model highlights which, why and when agents/roles need to communicate leaving all the details about how the communication takes place to the design process. The communication model is typically refined through several iterations as long as new interactions are discovered. It can be conveniently expressed by means of a number of interaction diagrams. This model is interaction centric and shows the initiator, the responders, the motivator of an interaction plus other optional information such as the trigger condition and the information achieved and supplied by each participant. The Figure 8 shows the example of our customizing toolkit.

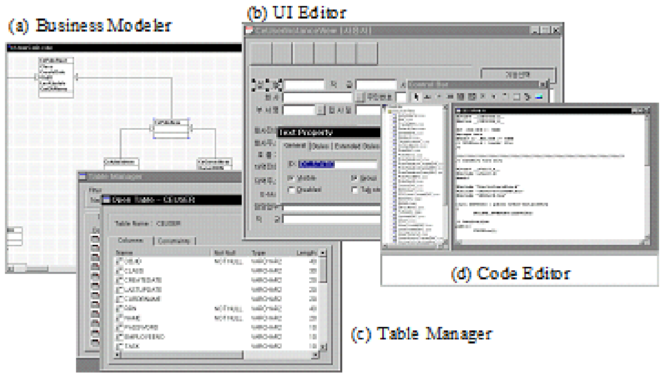


Figure 8. Example of our Customizing Toolkit

4.2. UI Editor

User interface classes can be customized by UI editor, whose functionality is the manipulation of the user interface based on the framework repository, and modification, replacement or creation of the elements defined in the user interface. The UI editor is based on the direct manipulation principal. An application developer can customize the user interface directly and these changes will be applied to the user interface model stored in the framework repository. With UI editor, we can add the elements for newly defined attributes and the operations of the domain model, and modify the appearance of existing elements. Then, we generate new UI components without changing other components. In this case, we don't need re-compiling for new user interfaces since our framework repository simply handles the user interface components.

4.3. Table Manager

When we define business concepts, we already define the mapping information between the domain object and the persistent object. Then our mapper can automatically generate a table schema based on this mapping information. However, when an application developer still wants to define the detailed features of persistent classes, our table manager can help this process. In fact, our table manager creates, updates, deletes and retrieves persistent class as well as its instances and fields. Thus, it is not necessary for the application developer to know how to handle the specific database. These are the responsibilities of the table manager.

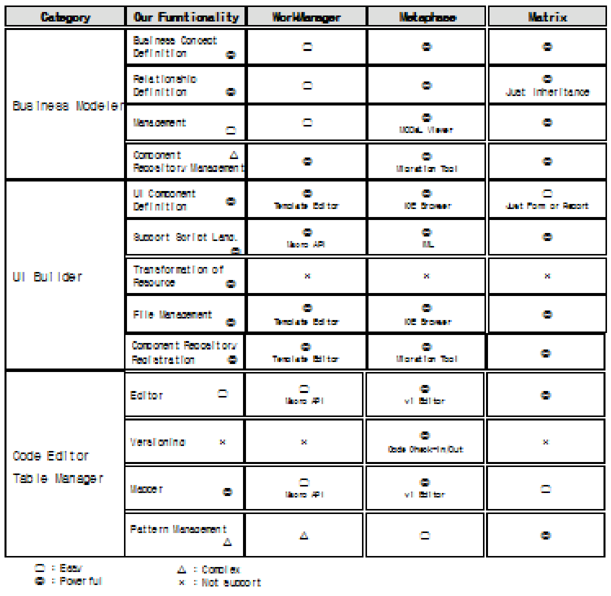
4.4. Visualizer

The first goal is to help the understanding process when an application developer starts to customize the NextMAM framework. Our components defined in the framework repository are applied to the dynamic run-time object design pattern suggested by NanoSoft [11]. We modified OVCore library for our purpose. We changed ClassBroker for handling the class description as class key such as class ID. In addition, we also modify CobjectInfo to manage our MAMObject. We can easily understand the dynamic behavior without executing the framework by visualizer since the framework repository contains the run-time objects. In addition, application developers can identify what component will be customized while they understand the framework because the framework repository contains the mapping information between run-time object and real component.

4.5. Evaluation

Some practical comparison would contribute to the justification of the NextMAM toolkit, while several previous MAM frameworks also have their customizing utilities. In this section, we describe the results of the comparison of our customizing toolkit with others. We compared our toolkit with WorkManager, MetaPhase, and Matrix. The result is summarized in Table 1.

Table 1. The Meta Model for MAM



Category	Our Funtionality	WorkManager	Metaphase	Matrix
Business Modeler	Business Concept Definition ●	▭	●	●
	Relationship Definition ●	▭	●	● Just Inheritance
	Management ▭	▭	● MODEL Viewer	●
	Component Repository Management △	●	● Migration Tool	●
UI Builder	UI Component Definition ●	● Template Editor	● IDE Browser	▭ Just Form or Report
	Support Script Lang. ●	● Macro API	● IML	●
	Transformation of Resource ●	×	×	×
	File Management ●	● Template Editor	● IDE Browser	●
	Component Repository Registration ●	● Template Editor	● Migration Tool	●
Code Editor Table Manager	Editor ▭	▭ Macro API	● vi Editor	●
	Versioning ×	×	● Code Check-in/Out	×
	Mapper ●	▭ Macro API	● vi Editor	▭
	Pattern Management △	△	▭	●

▭ : Easy
● : Powerful
△ : Complex
× : Not support

5. Conclusion

While it is very difficult to reuse the design concepts and domain knowledge, the framework has known to be a promising way to understand the application domain and increase the reusability of the software.

In this study, we have developed MAM framework based on the object technology. We generalized several MAM applications to capture the domain object model. With a generalized object model, we apply to design patterns to document the framework showing what the hot spots are. It was based on our design of customizing toolkit. To support the reuse process of the framework, we have also developed the customization toolkit for our NextMAM framework.

NextMAM, based on the results of the comparison of the frameworks, turns out to be very powerful in UI Builder and Table Manager Category. It also demonstrates acceptable functions in Business Modeler, while Code Editor results in differentiated functions compared to others.

Acknowledgements

This work was supported by the Korea National Research Foundation (NRF) granted funded by the Korea Government (Scientist of Regional University No. 2012-0004489).

References

[1] O. H. Barros, "Business Information System Design Based on Process Pattern and Frameworks", BPtrends, Available: www.bptrends.com, **(2004)** September.
[2] E. Metak and J. Caputo, "Dynamic Runtime Objects: Building Applications Your User Can Modify at Runtime", Microsoft Systems Journal, **(1997)** July, pp. 49-74.
[3] G. P. Moynihan, A. Suki and D. J. Fonseca, "An Expert System for the Selection of Software Design Patterns", Blackwell Publishing, vol. 23, no. 1, **(2006)** February, pp. 39-52.
[4] E. Gamma, R. Helm, R. Johnson and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley/ACM Press, MA, **(1995)**.
[5] H. S. Hamza, "Improving Analysis Patterns Reuse: An Ontological Approac", Ontologies as Software Engineering Artifacts Workshop (OOPSLA'04), **(2004)**.
[6] T. George, W. Heineman and T. Councill, "Component-Based Software Engineering", Addison-Wesley, **(2001)**.
[7] J. C. Grundy, W. B. Mugridge and J. G. Hosking, "Constructing component-based software engineering environments: issues and experiences", Journal of Information and Software Technology, vol. 42, no. 2, **(2000)** January, pp. 117-128.
[8] G. Hu, "A Formal Specication of UML Class and State Diagrams", Proceeding of 9th ACIS International Conference on Software Engineering, Artificial Intelligence, Network & Parallel and Distributed Computer (SPDN 2008), Studies in Computational Intelligence, Springer, vol. 149, **(2008)** August, pp. 247-257.
[9] D. Ballis, A. Baruzzo and M. Comini, "A Rule-Based Method to Match Software Patterns Against UML Models", Proceeding of the 8th International Workshops on Rule-Based Programming (RULE'07), France, Paris, **(2007)**, pp. 239-248.
[10] N. B. Harrison, P. Avgeriou and U.Zdlin, "Using Patterns to Capture Architecture Decisions", IEEE Software, vol. 24, no. 4, **(2007)** July-August, pp. 38-45.
[11] U. Zdun, "Systematic Pattern Selection Using Pattern Language Grammars and Design Space Analysis", Software: Practice and Experience, vol. 37, no. 9, **(2006)**, pp. 983-1016.
[12] S. Henninger and P. Ashokkumar, "An Ontology-Based Metamodel for Software Patterns", Proceeding of the 18th International Conference on Software Engineering and Knowledge Engineering (SEKE2006), San Francisco, **(2006)** July 5-7, pp. 327-330.
[13] R. H. Wison, "A Scientific Routine for Stock Control", Harvard Business Review, vol. 13, no. 1934, pp. 116-128.