

A LAYERED METAMODEL FOR HIERARCHICAL MODELING IN UML

CHEE-YANG SONG* and DOO-KWON BAIK[†]

*Department of Computer Science and Engineering, Korea University,
#1, 5-Ka, Anam-dong, Sungbuk-ku, Seoul 136-701, South Korea*

**songyang@kt.co.kr*

[†]baik@software.korea.ac.kr

Submitted 3 November 2001

Revised 12 March 2002

Accepted 12 July 2002

As software is becoming larger and more complex, it is increasingly important to use the hierarchical modeling approach. Unfortunately, however, UML does not specify each metamodel with hierarchy for model by modeling phase. Thus, most UML-based methodologies do not address the hierarchical modeling for model. As a method for supporting hierarchical modeling on UML, this paper proposes a layered metamodel which defines hierarchically modeling elements of model according to the modeling phase. We describe each metamodel with hierarchy for models in UML, then present the hierarchical integrated metamodel combined with each metamodel by three modeling phases (conceptual phase, specific phase, and concrete phase). Therefore, designers are able to construct the hierarchical model by applying the metamodel with hierarchy. Using the hierarchical metamodel enables designers to improve the usability of UML and reusability of application model.

Keywords: Hierarchical modeling; layered metamodel; UML; modeling technique.

1. Introduction

Recently, with the increasing size, complexity, and variety of software, it is difficult to understand a software system in its entirety. Accordingly, a development method following on incremental approach from an abstract level to a concrete level is necessary. That is, developing a software application hierarchically in terms of the different levels of abstraction for designing complex and large-scale software is a critical issue.

UML is being widely used as a standard language and a universal technique for modeling object-oriented applications and component-based applications. The metamodel for defining UML models has mainly been studied in OMG UML. The structure of the UML metamodel is organizing its models into a logical package for all models systematically [1–3]. However, this involves a complex structure in the

syntax and semantics for defining the models. Most other studies [4–7] have brought into focus the precise semantic definition for UML models using core elements and extension mechanisms to apply in various fields. Also, the work in integrating the modeling language and the formal language has been done to specify the UML models formally [8, 9]. Nevertheless, very little research has been paid to the metamodel providing hierarchical modeling for the UML models.

The UML metamodel has many problems in modeling software hierarchically:

- Metamodels in OMG UML do not define each individual metamodel for each model, owing to its focus on the entire structure while considering all models. Each metamodel is needed since developers actually design an application model independently with an individual UML model. Furthermore, UML does not provide a precise description of each metamodel that specifies hierarchically the modeling elements or constructors by the abstraction levels of the modeling life cycle. Just as in [10], it divides the use case model into three hierarchical structures and classifies the class model into two development phases, without providing for both the specification by the metamodel and the hierarchical structure for the other models in UML, such as statechart model, subsystem model, etc. It may confuse designers concerning the UML usage, and lead to a deterioration of usability for the UML models.
- Likewise, most UML-based methodologies [10–14] do not support hierarchical modeling by applying the hierarchical metamodel for UML models. It may depreciate reusability for application model. For modeling complex, various, and sizable software, each metamodel has to be specified by adding the extended elements of the model corresponding to the modeling phase. For example, in making a class diagram, it is generally applied from the conceptual analysis phase to the concrete design phase. That is to say, the conceptual analysis phase requires the class model to be designed with the modeling elements of class, attribute, and relationship among the classes. Meanwhile, the design phase requires the class model to be designed by including the elements of attribute type, operation, tagged value, and parameter to the previous class diagram. Thus, it requires the hierarchical metamodel specified with the suitable design elements according to the modeling phase.

Accordingly, hierarchical modeling for UML models is needed for the following reasons: Firstly, an incremental and abstract modeling approach is required as software is becoming more large-scale. Secondly, it is difficult for designers to understand or capture the entire and detailed structure of an application system all at once in the initial analysis stages. That is, it is useful to quickly obtain understanding for application. Finally, hierarchical modeling enhances reusability of the application model.

In summary, although hierarchical approaches through division of the modeling phase, task, and activity supports development methodologies well, a hierarchical

method for designing models is not provided. So to speak, the UML does not specify each metamodel with a hierarchy individually. The UML-based methodologies do not provide a hierarchical modeling based on the hierarchical metamodel for UML.

To address this problem, this paper proposes a hierarchical metamodel that constructs an application model hierarchically in terms of three modeling phases. The aim of our work is to enhance the usability for UML and reusability for the application model. This is made possible: (1) by specifying each metamodel which defines hierarchically the modeling elements conducted for each modeling phase; (2) in terms of building hierarchical integrated metamodels by modeling phase. The main concepts used in this paper are metamodel and hierarchy (Conceptual modeling phase, Specific modeling phase, and Concrete modeling phase). We have successfully applied our method to application domains such as the C_MDR (Component_MetaData Registry) as a case study for addressing the effectiveness of the hierarchical modeling approach to the software development.

The remainder of this paper is organized as follows. Section 2 sketches our approach concepts. Section 3 deals with the building of each metamodel with a hierarchy, considering both the abstract level and the concrete level. Section 4 describes how each metamodel is combined to build the hierarchical integrated metamodel in terms of three modeling phases. Section 5 illustrates how the hierarchical metamodel is applied for the C_MDR domain as a exploratory case study. Section 6 represents an assessment for the proposed metamodel with the UML metamodel and other development methodologies. Finally, Sec. 7 summarizes this paper with a contribution and a future plan.

2. Our Approach

The approach described in this paper is used to assist designers in the creation of software using UML. Our goal is to support the hierarchical metamodel that can produce a hierarchical model. This paper addresses a hierarchical approach by using metamodel defining the suitable modeling elements for UML models according to modeling phase. To provide it, the approach proceeds as follows:

- Create each *metamodel* with a *hierarchy* for the UML model
- Build the hierarchical integrated metamodel combined with each metamodel

The concept of metamodel and hierarchy is used to create the hierarchical metamodel. Generally, the metamodel has been used to define the modeling language [15] and to specify the application model in various domain fields [16–19]. In the context of this paper, we define the metamodel as a conceptual framework that describes the core elements and relationships underlying the field of UML for defining the abstract syntax and the static semantics of target models precisely and concisely. To provide the syntax and static semantics to the metamodel, the syntax is specified with the basic element including the target model (or diagram) and the relationships among the elements. The static semantics is expressed by representing

the multiplicity constraints and type of relationship among the elements.

The hierarchy is organized as a hierarchical structure for some target items. It is important to note that the hierarchy separates design models for good software reuse. This concept can be classified in terms of different levels of granularity and degree of abstraction. Thus, the layered architecture types of the hierarchy by granularity are defined through a sequence of refinements as follows: architecture layer, component layer, and object layer [20]. The hierarchy types in terms of abstraction are categorized into Conceptual modeling phase, Specific modeling phase, and Concrete modeling phase in accordance with the development life cycle. The hierarchy concept is used to build both the metamodel and the modeling procedure.

The principles for constructing the metamodel consist of:

- Constructing the hierarchical metamodel precisely based on the modeling phase with core elements and relationships of the model frequently used in practice.
- Building the hierarchical integrated metamodel based on three modeling phases.
- Modeling based on design elements in the metamodel.

3. Each Metamodel with a Hierarchy

To build each hierarchical metamodel, this section describes the hierarchical division for each model, rules for constructing the metamodel, and construction of each metamodel with a hierarchy. The target models for building each metamodel are represented by grouping the three aspects in Table 1.

Table 1. Target models for constructing each metamodel in UML.

| Aspect | Target model |
|------------------|--|
| Static model | Subsystem diagram, Package diagram, Component diagram, Deployment diagram, Class diagram |
| Behavioral model | State Chart diagram, Interaction diagram (Sequence diagram, Collaboration diagram) |
| Functional model | Use case diagram, Activity diagram |

The metamodel is expressed in class diagram. Each metamodel is constructed with mandatory elements and relations in the target model. In order to construct the hierarchical metamodel, each metamodel is divided into two or three metamodels for the target model in Table 1 according to three modeling phases. Each metamodel is combined as source constituents for building an integrated metamodel into three modeling phases.

3.1. Hierarchical division of each model

In order to support hierarchical modeling, each metamodel should be built to meet the abstraction degree of modeling phase. That is, each metamodel has to be divided

hierarchically by adding the extended elements corresponding to the concreteness degree of the development life cycle.

The partition criteria of each metamodel by the modeling phase is based on:

- Applying the essential use scope of the model itself. For example, the package diagram in Table 2 consists of the hierarchical metamodels being used up to the Conceptual modeling and Specific modeling phase, since both have a grouping of model elements (including package unit) and process unit of elements (including class unit) that are large at the abstraction level.
- Correspondence based on the division level of element of the use case model and class model described in [10].
- Same leveling compared with the elements in other models.

Based on these criteria, Table 2 represents the models' application by partitioning into two phases, and Table 3 shows the models' application by division into three phases. Therefore, not all modeling works is completed at the concrete phase, in some cases modeling at the specific phase can be finished (e.g., in the case of the subsystem model).

Table 2. Two phase-applied models.

| Model/ Phase | Interaction Model | Usecase Model | Component Model | Subsystem Model | Package Model | Deployment Model |
|---|--|---|--|--|--|---|
| Conceptual modeling (requirement analysis) | | System _level usecase metamodel | Conceptual _level component metamodel | Conceptual _level subsystem metamodel | Conceptual _level package metamodel | Conceptual _level deployment metamodel |
| Specific modeling (preliminary design) | Specific _level interaction metamodel | Subsystem _level usecase metamodel | Specific _level component metamodel | Specific _level subsystem metamodel | Specific _level package metamodel | Specific _level deployment metamodel |
| Concrete modeling (detail design) | Concrete _level interaction metamodel | | | | | |

3.2. Construction of each model with a hierarchy

Models in UML have various elements and relationships separately. Therefore, some rules are needed that can be applied to a variety of cases in common for deriving the elements and the relationships between the models with consistency and clarity. The criterion for extracting the elements and the relationships is based on the standard specified in OMG UML V1.4. The following describes the related rules:

Table 3. Three phase-applied models.

| Model/Phase | Usecase Description | Class Model | State Chart Model | Activity Model |
|---|---|-----------------------------------|-------------------------------------|-----------------------------------|
| Conceptual modeling (requirement analysis) | High_level usecase description metamodel | Concept_level Class metamodel | System_level State chart metamodel | System_level activity metamodel |
| Specific modeling (preliminary design) | Essential_level usecase description metamodel | Specific_level class metamodel | Usecase_level state chart metamodel | Usecase_level activity metamodel |
| Concrete modeling (detail design) | Real_level usecase description metamodel | Implemental_level class metamodel | Class_level state chart metamodel | Function_level activity metamodel |

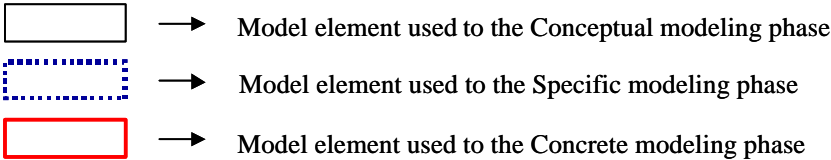


Fig. 1. Element notation of the metamodel over three modeling phases.

- Rule 3.1.** The selection of elements in the target model takes a necessary, core, and widely used element of the many elements in a simple manner. That is, the mandatory elements in the target model are captured. An element is expressed to one class.
- Rule 3.2.** The relationship among elements is represented as association, aggregation, composition, inheritance, and multiplicity.
- Rule 3.3.** The hierarchical metamodel is constructed by adding the extended elements of the current phase to the metamodel of the previous phase.
- Rule 3.4.** Stereotype is used for separating the duplicate elements over phases. Some models are applied with the same notation of model according to the different levels of granularity in the modeling target, for instance, activity model and statechart model.

Each metamodel is built individually for ten models in UML as shown in Table 1. As a partial illustration of the constructed metamodel, the commonly used subsystem metamodel, use case metamodel, and class metamodel are shown. As a notation legend of the hierarchical expression used in common for each metamodel over the three modeling phases, Fig. 1 shows the notation for identifying the hierarchy of each metamodel using the different notation to the element.

When UML models are applied for developing an application according to the modeling process, each metamodel is referred hierarchically. Then, the designed model of the application is verified by each metamodel.

3.2.1. *Subsystem metamodel*

The purpose of the subsystem construct is to provide a grouping mechanism for specifying a behavioral unit of a physical system. The subsystem model is designed at the level of software architecture for the initial requirement analysis. The hierarchical metamodel required in the subsystem model should meet the modeling levels from subsystem to operation for element being constructed. The subsystem is obtained in Conceptual modeling phase. On the other hand, the operation can be driven from the interaction model being constructed by the scenario in the use case description. However, the interaction model is built in the Specific modeling phase generally. Therefore, it needs two hierarchical metamodels to apply the model at the Conceptual modeling phase and the Specific modeling phase. The conceptual-level subsystem metamodel defines the modeling level to capture the elements of the subsystems and relationships among subsystems at the conceptual level. The specific-level subsystem metamodel specifies the modeling level with all elements owning the subsystem model by newly adding operations, specification elements, and realization elements.

With respect to the constitution elements of the subsystem model, the subsystem consists of specification elements and realization elements. Here, the appended elements for more details require the subsystem name, interface and relationship among the subsystems, useCase and operation for detail of the specification elements, collaboration for mapping between the specification elements and the realization elements, and component for representing a subsystem with components. They become the elements and relationships of each metamodel for the subsystem model.

The subsystem metamodel to support the Conceptual modeling phase (elements drawn as rectangles) and the Specific modeling phase (elements drawn as dotted rectangles) is illustrated in Fig. 2 where the subsystem model contains the elements of the subsystem and relationship by a composition with a multiplicity of 1 to 1..*. The relationship between subsystem and interface is expressed by an aggregation relationship labeled “has” relation label, because the ‘interface’ element can be shared. The relationship between the subsystem and the component is represented by inheritance relationship since the component is a type representing the inner part of a subsystem. Therefore, designers can easily build the subsystem model using the elements and relationships included within the hierarchical subsystem metamodel.

3.2.2. *Use case metamodel*

The use case model shows the relationship among useCases within a system and/or the interaction between the system and their actors through usage scenarios of

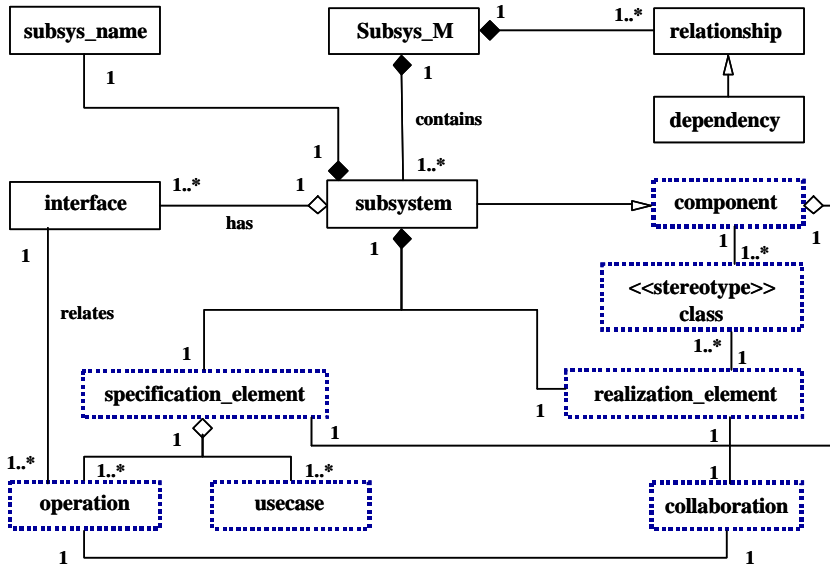


Fig. 2. Subsystem metamodel providing the two modeling phases.

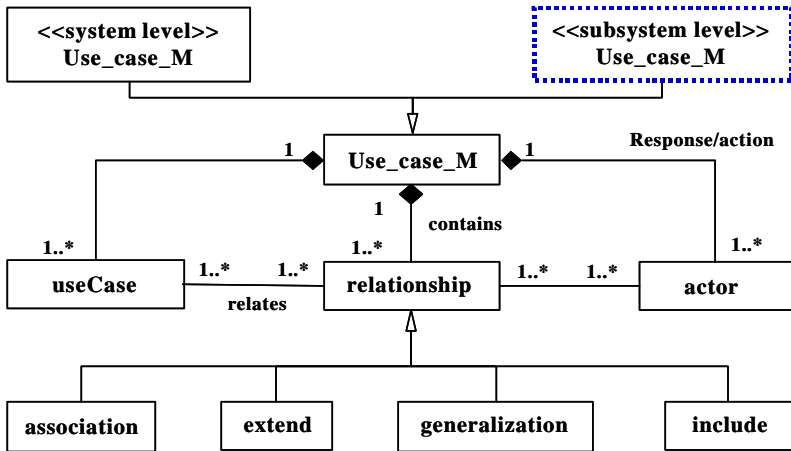


Fig. 3. Use case metamodel for use case model supporting two modeling phases.

system. This model consists of the use case diagram and use case description to design a further detailed design. The use case description stands for the semantics of use case, and it is divided into three modeling phases according to the degree of abstraction, such as High-level, Essential, and Real use case descriptions described in [10]. To build the hierarchical metamodel, as shown in Fig. 3, the metamodel for the use case model is divided into two metamodels so that it can be applied at the Conceptual modeling phase supporting the modeling at the system level and the

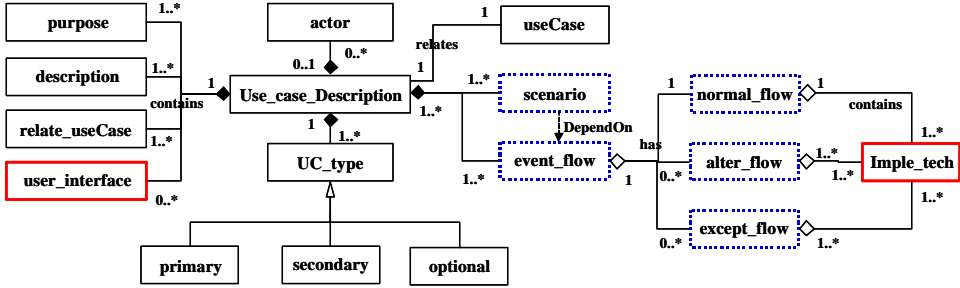


Fig. 4. Use case metamodel for use case description providing three modeling phases.

Specific modeling phase providing the modeling at the subsystem level of granularity, without adding its modeling element (not changing its express notation). It is expressed separately using a $\ll\text{system level}\gg$ stereotype and $\ll\text{subsystem level}\gg$ in the metamodel.

Each metamodel for the use case description, not specified in UML, creates a hierarchical metamodel divided into three modeling phases as shown in Fig. 4. For instance, the elements (Real use case description) indicated by thick rectangle, such as `user_interface` and `implementation_technology`, are designed at the Concrete modeling phase. Use case description has many elements like `useCase`, `purpose`, `actor`, `description`, and so on. The relationship between `scenario` and `event_flow` is represented by `dependOn`. Elements of precondition and post condition are not included in its metamodel since these are not mandatory elements.

3.2.3. Class metamodel

The class model describes the types of objects within the system and the relationships (association and subtype) among them. The model also shows attributes and operations of the objects it describes. In [10], it is divided into two modeling phases according to the abstraction degree of the software life cycle. In this article, the metamodel for the class model creates each metamodel by providing three phases for covering up to the implemental viewpoint in order to support an implemental property (such as language type) in greater detail and more precisely. The `conceptual_level` class diagram should be represented with little or no regard for the software that might be implemented. The `specific_level` class diagram shows the interfaces of the software, not the implementation. The `concrete_level` class diagram actually has the classes and is laying the implementation bare.

The metamodel for the class model is composed of many modeling elements such as class types, relationship types, visibility, attribute, operation, etc., as shown in Fig. 5. For specifying the different types of class (such as the class type by role and abstraction degree), various stereotypes are used in its metamodel, such as the $\ll\text{concrete}\gg$ class and $\ll\text{business_concept}\gg$ class. They are designed during the Specific modeling phase. The relationship between them and the class ele-

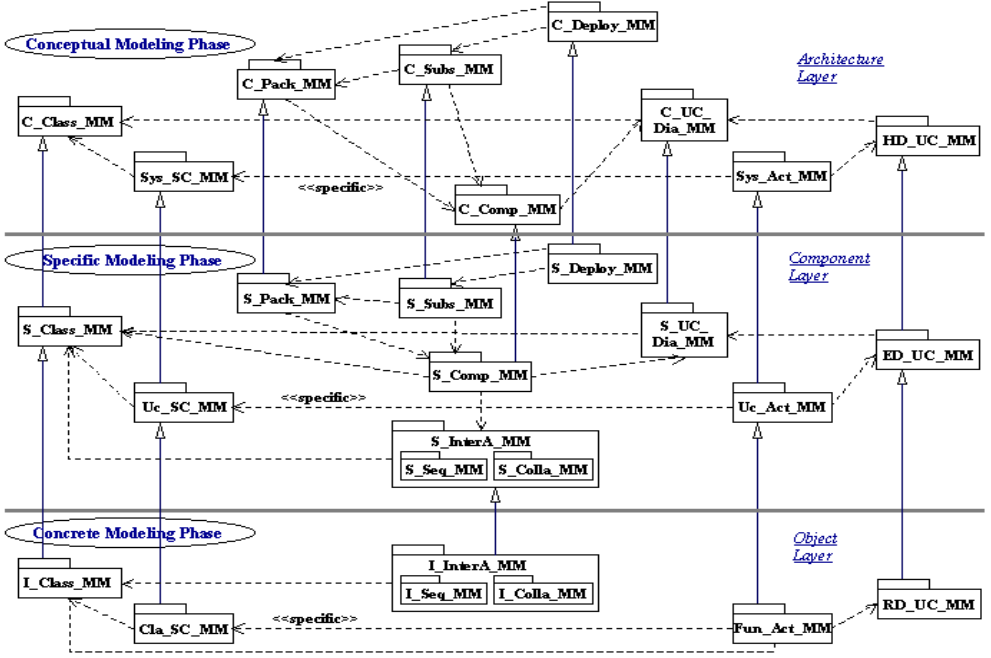


Fig. 6. Constitution and relationship of the integrated metamodels.

phase. The modeling work in the conceptual modeling phase is performed with the models included in this phase. The full name for abbreviations in Fig. 6 can be seen in Tables 2 and 3.

Integrating each metamodel requires analysis of the relationship between each metamodel and among the elements. At first, the relationship among the metamodels is shown in Fig. 6 with the package diagram. Here, the mapping of each metamodel among the phases (or layers) represents the inheritance relationship, since each metamodel in the lower phase is inherited from each metamodel in the upper phase. The mapping between each metamodel in the same phase is expressed by the dependency relationship based on the interactive inclusion specified in the UML.

Meanwhile, with respect to the relationship among the elements, the mapping between each metamodel actually stands for the connection of the related elements and needs relationship analysis among the elements. The relationship is derived from each metamodel. For instance, the subsystem metamodel (see the part marked oval in Table 4) shows the relationship between ‘component’ element and ‘subsystem’ element, for example, their relationship has an inheritance since a subsystem is a kind of component. The subsystem is a subclass of package and one class has one sequence model. Within these relationships, Table 4 depicts the relationship matrix as a road map among the core elements in each metamodel.

Table 4. Relationship matrix among elements in each metamodel.

| Model | Deployment | Subsystem | Package | Component | Use case | Class | Activity | StateChart | Interaction |
|-------------|---|---|---|---|--|--|---------------------------------------|--|----------------------------------|
| Deployment | | | 1 : 1..* node ϕ -- package | 1 : 1..* node ϕ -- component | | | | | |
| Subsystem | | | subsystem --> node | subsystem --> component | 1 : 1..* subsystem ϕ -- useCase | 1 : 1..* subsystem \diamond -- class | | | |
| Package | 1 : 1..* package -- ϕ node | 1 : 1..* package \triangleleft -- subsystem | 1 : 1..* package \diamond --package | 1 : 1..* package ϕ -- component | 1 : 1..* package ϕ -- useCase | 1 : 1..* package \diamond --class | | | |
| Component | 1..* : 1 component -- ϕ node | component \triangleleft -- subsystem | 1..* : 1 component -- ϕ package | 1 : 1..* component \diamond --component | 1 : 1..* Component -- useCase | 1 : 1..* component \diamond -- class | | | |
| Use case | | 1..* : 1 useCase -- ϕ subsystem | 1..* : 1 useCase -- ϕ package | 1..* : 1 useCase -- component | | | | | |
| Class | | 1 : 1..* class -- \diamond subsystem | 1 : 1..* class -- \diamond package | 1 : 1..* class -- \diamond component | | | | 1 : 1..* class ϕ -- state | |
| Activity | | | | | | | | 1 : 1..* object ϕ -- state | 2..* : 1..* object -- link |
| StateChart | | | | | | 1..* : 1 state -- ϕ class | 1..* : 1 state -- ϕ object | 1 : 1..* state \diamond -- state | |
| Interaction | | | | | | | 1..* : 2..* link -- object | | |

4.2. Hierarchical integrated metamodel

Based on the constitution of the integrated metamodels and the relationship mapping among metamodels, we can build the hierarchical integrated metamodel by combining each metamodel in terms of modeling phase. As the modeling phase progresses from the abstract level to the concrete level, the elements of the target model append further additional elements or system granularity for modeling attains smaller size. The integrated metamodels are constructed with the core elements in each metamodel to maintain simplicity. Thus, all elements in each metamodel need not be included in the integrated metamodel, because we can always reference each metamodel.

The hierarchical integrated metamodel creates three integrated metamodels by three modeling phases. For example, Fig. 7 illustrates an integrated metamodel constructed with eight metamodels at the Conceptual modeling phase.

The statechart metamodel and the activity metamodel include all elements having their models, because these models are modeled with different granularity according to abstraction level of application system without change of modeling elements over three modeling phases.

Mapping between metamodels is combined by the connection among the core elements. Here, the included elements belong to the conceptual level in each metamodel. The designer in this phase can perform the modeling precisely and verify the constructed application models easily with the elements and relationships specified in Fig. 7.

The integrated metamodels for the specific modeling phase are built with extended elements of each model or small granularity of system for more detail modeling.

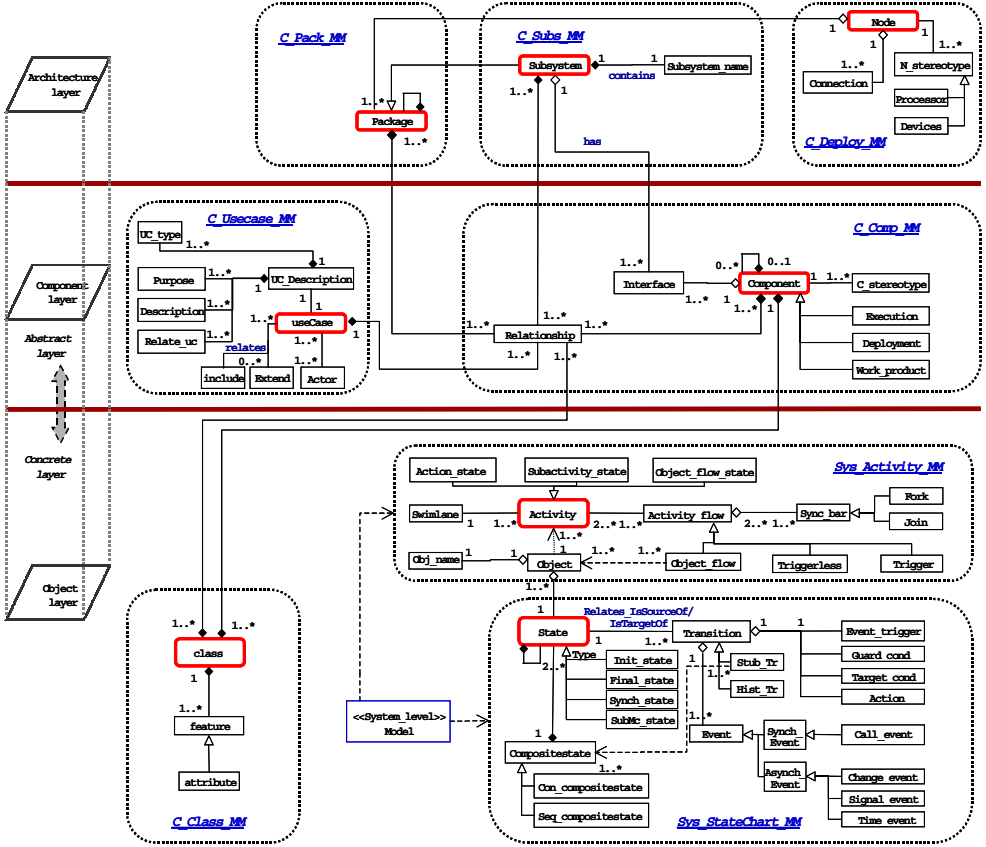


Fig. 7. The integrated metamodel in Conceptual modeling phase.

In order to enhance understandability for all UML models, one unified integrated metamodel, which combines all metamodels at the Concrete modeling phase, is shown in Fig. 8. It can be used as a blueprint for UML. If all information representing this integrated metamodel is managed through a repository of the Meta-CASE Tool, it can realize the automatic verification for application models. This metamodel is also used for checking consistency and traceability of the application models.

5. An Exploratory Case Study

An exploratory case study illustrates the practical use of the hierarchical metamodel for a software design. Its purpose is to ensure reliability of the approach mechanism and the procedure of activities. The case study shows the hierarchical approach using each metamodel and the integrated metamodel by modeling phase.

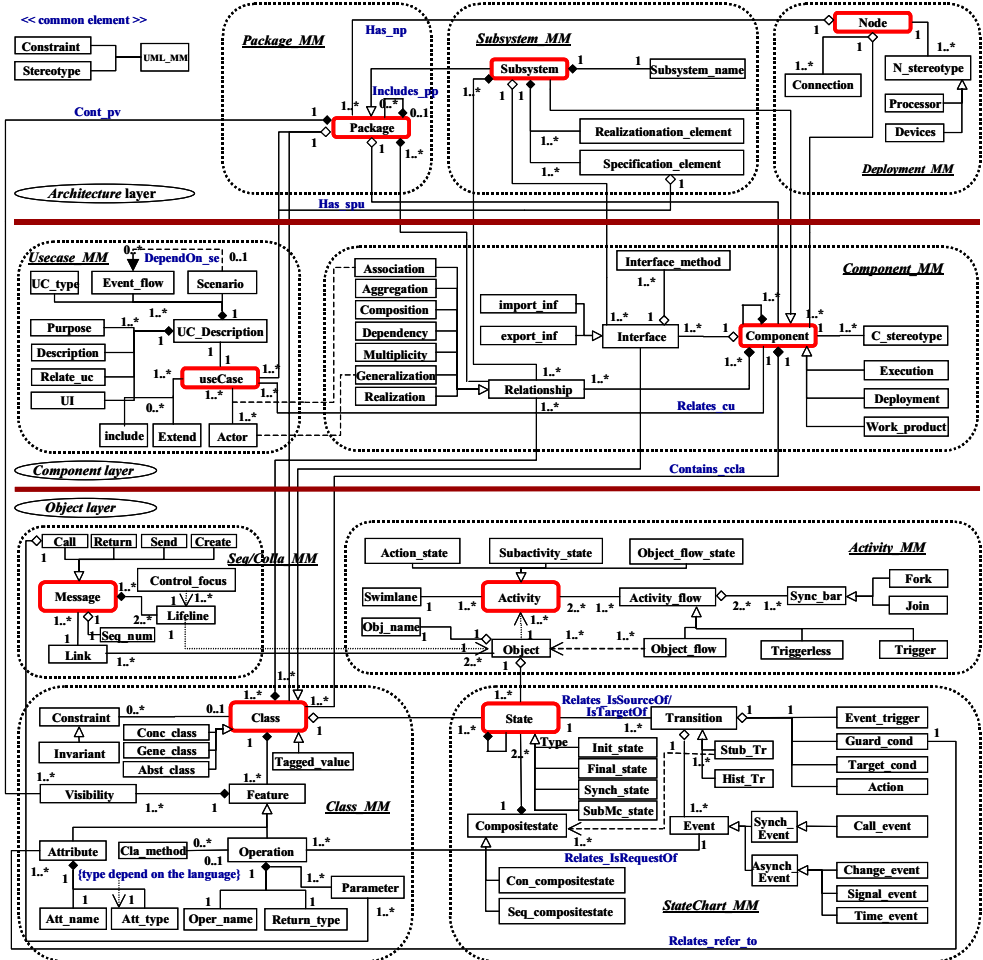


Fig. 8. One unified integrated metamodel for UML of all models.

We design a C_MDR application to share and to circulate standardized information for commercial components [21]. The C_MDR system is a tool to support registration and management of the standardized metadata for component products. There are three service systems, namely the component metadata management service, the component category code management service, and the analysis information management service. The models constructed are the subsystem model, interaction model, component model, etc., as shown in Table 1. Here, as an example, for the models constructed to each metamodel in Sec. 3, we show this for the subsystem model, use case model, and class model based on the hierarchical metamodel.

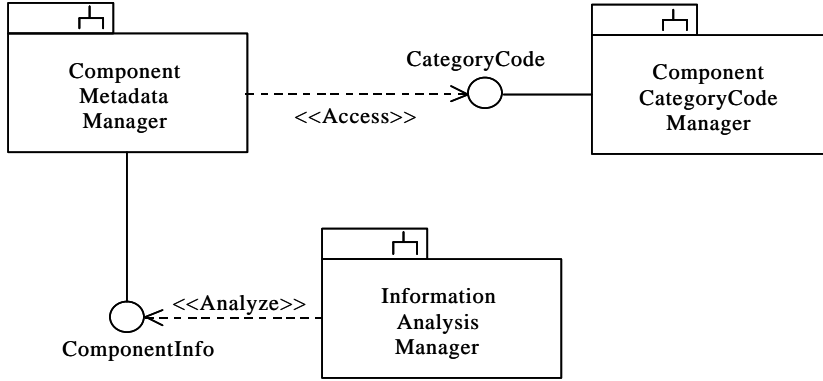


Fig. 9. Subsystem model in Conceptual modeling phase for C_MDR.

5.1. Conceptual modeling phase

The Conceptual modeling phase analyzes the domain requirements for C_MDR. At first, conceptual_level subsystem model is made in terms of referencing and verifying with the conceptual_level subsystem metamodel. After that, conceptual_level use case model and conceptual_level class model are designed using the previous artifacts and verified by using the modeling elements of each hierarchical metamodel. For the subsystem model, the modeling elements in Conceptual modeling phase consist of subsystem, interface, and dependency among subsystems in terms of each hierarchical metamodel in Fig. 2. With these elements, Fig. 9 illustrates a conceptual_level subsystem model organized with three subsystems for C_MDR.

The use case model is designed as a use case diagram and use case description using each hierarchical metamodel. Based on the subsystem model in Fig. 9, the use case model (constructed with the modeling elements of useCase, relationship, and actor at the system level in the metamodel) for the C_MDR system is built as three useCases: the 'Manage Component Metadata', 'Manage Component CategoryCode', and 'Manage Analysis Information', as shown in Fig. 10.

The use case description is constructed with respect to all useCases in Fig. 10 separately with the elements of useCase, actor, purpose, and relate_useCase. Thus, three use case descriptions are created. As an example, Table 5 illustrates High-level use case description in this modeling phase for the 'Manage Component Metadata' useCase.

For the class diagram, the modeling elements of the class diagram in the Conceptual modeling phase consist of class, attribute, and association among classes as specified in the conceptual_level class metamodel, only the business classes (component class, configuration class, etc.) and attributes are captured for C_MDR using the artifacts of the use case diagram (Fig. 10) and use case description (Table 5). With these as the modeling elements, Fig. 11 represents the class model designed at the conceptual level.

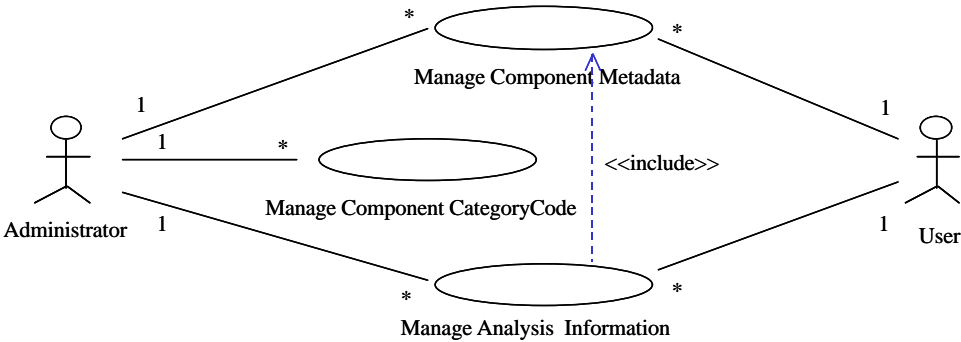


Fig. 10. Use case model in conceptual modeling phase for C_MDR.

Table 5. High-level use case description in conceptual modeling phase for the ‘Manage Component Metadata’ useCase.

| | |
|----------------|---|
| useCase Name | Manage Component Metadata |
| Actor | Administrator, User |
| UC_Type | Primary |
| Purpose | Registering and managing the metadata for component products. |
| Description | <ul style="list-style-type: none">• Administrator registers the standardized metadata for a new component product into database by category code scheme.• Administrator updates the stored component metadata for the changed products.• User searches the component metadata using various retrieval services. |
| Relate_useCase | Manage Analysis Information |

The designed models are verified by each metamodel applied individually in this modeling phase. The relationship of elements between the produced models is verified by the integrated metamodel for this phase in Fig. 7.

5.2. Specific modeling phase

The Specific modeling phase performs the preliminary design for C_MDR. By utilizing artifacts or outputs of the Conceptual modeling phase, the application models are made with the extended elements of this modeling phase specified in each hierarchical metamodel. As for the use case model, the use case diagram in this modeling phase is the same as the use case diagram in the Conceptual modeling phase without appending any modeling elements. But this phase further refines the use case diagram of the Conceptual modeling phase with the granularity unit of a smaller target size than that of the Conceptual modeling phase, as shown in Fig. 3. Hence,

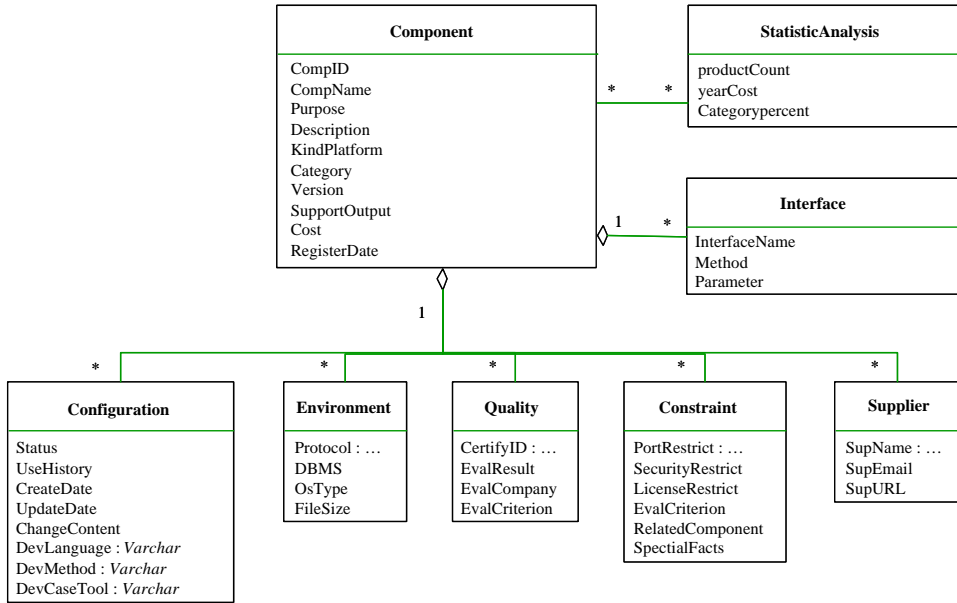


Fig. 11. Class model in conceptual modeling phase for C_MDR.

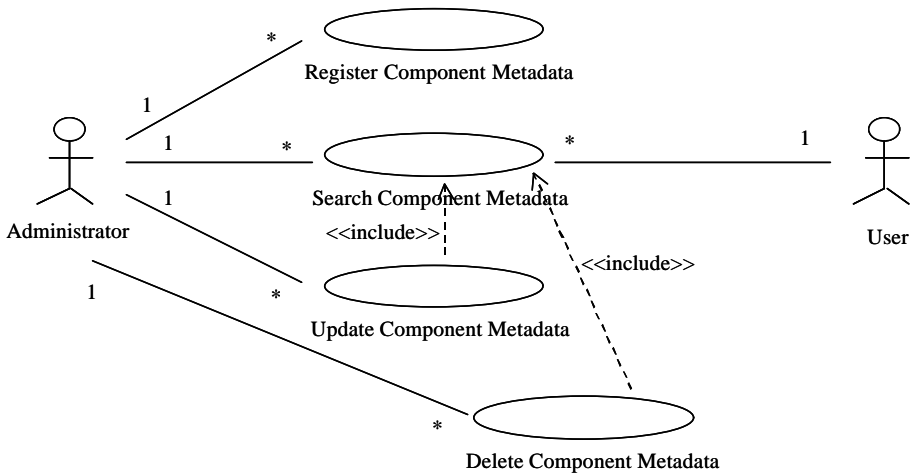


Fig. 12. Use case model in Specific modeling phase for the 'Manage Component Metadata' useCase.

it creates the use case diagram individually using the same modeling elements for each useCase constructed in conceptual phase. The use case diagram designed in this modeling phase is shown in Fig. 12, as an example for the 'Manage Component Metadata' useCase in Fig. 10. Its diagram is fulfilled in this modeling phase.

However, use case description is designed by appending the elements of scenario and event_flow (normal_flow, alter_flow, and except_flow) based on the Essential Use case description metamodel as shown in Fig. 4. As an example, Table 6 represents the use case description for the ‘Manage Component Metadata’ useCase in terms of concrete specification more than that of conceptual modeling phase.

For the subsystem model, the modeling elements in the Specific modeling phase are composed of its elements by appending specification elements (operation and useCase), realization elements (collaboration), and components to the elements in the Conceptual modeling phase. The parts for the specification elements and realization elements in subsystem model are modeled using the artifacts of the use case model in Fig. 12. In this modeling phase, the subsystem model is designed with respect to all subsystems in Fig. 9 separately. Therefore, three subsystem models are created. As an example, the designed subsystem model of this phase is shown in Fig. 13 for ‘Component Metadata Manager’ subsystem. The subsystem model divided into two modeling phases is completed in this modeling phase.

The class diagram in this modeling phase concretes the class model in Conceptual modeling phase with extended elements, such as attribute type, operation, method, visibility, stereotype, UI interface class, etc. In this modeling phase, the

Table 6. Essential Use case description in specific phase for the ‘Manage Component Metadata’ useCase.

| | | |
|--|---|--|
| useCase Name | Manage Component Metadata | |
| Actor / UC_Type / Purpose / Description / Relate_useCase | Same with High-level Use case Description | |
| Normal Flow | <div><div><div>** Register component metadata **</div><div>[Actor Action]</div><div>[N-1] Administrator requests the registration for a new component</div><div>[N-3] Administrator types specification information, category information, quality information, environment information, etc for component product on UI. On completion of item entry, then request storing of this component metadata.</div><div>[N-6] After conformation of successive registration, Administrator performs the registration for another components repeatedly.</div><div>** Update component metadata **</div></div><div><div>[System Response]</div><div>[N-2] displays the UI form for registration. The kinds and category scheme for component are presented.</div><div>[n-4] checks the duplication of component name. If no errors, creates the objects for component metadatas, then makes the records of component, configuration, quality, supplier etc, store this information into DB.</div><div>[N-5] presents a success message to administrator.</div></div></div> | |
| Alter Flow | None | |
| Except Flow | <div><div>[E-1] If the duplication of component name occurs, then generates error messages.</div><div>[E-2] If there occurs some error when data is stored to DB, then generates error messages for this.</div></div> | |

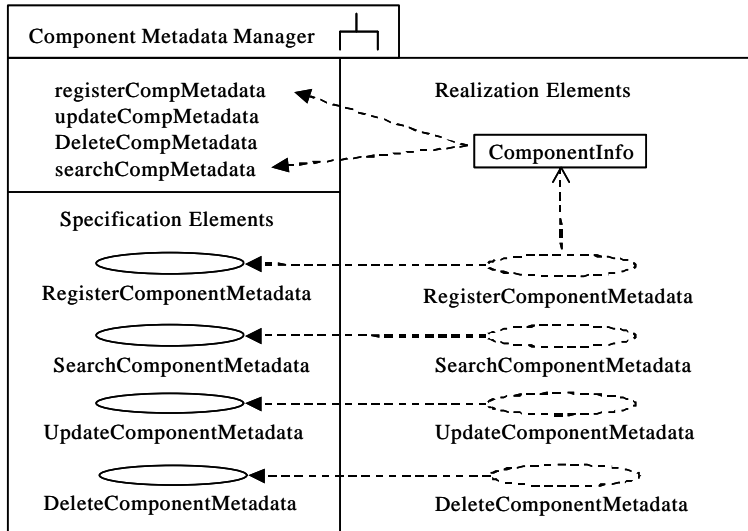


Fig. 13. Subsystem model in Specific modeling phase for the ‘Component Metadata Manager’ Subsystem.

class model identifies the UI class, DB class, etc. as shown in Fig. 14 where the parts expressed in italics indicate the newly designed elements in this modeling phase.

5.3. Concrete modeling phase

The Concrete modeling phase performs the concrete design in detail for the C_MDR application. This phase accomplishes the use case description and the class model. All the models produced so far are verified at the end by the one unified integrated metamodel, depicted in Fig. 8 for checking the consistency among the application models.

For the use case model, we create a Real use case description by using additional elements like *user_interface* and implementation technique (such as input and output items, and so on), and by concreting the more normal flow with the implementation view to the Essential use case description of the previous phase. The Real Use case description for this modeling phase is shown in Fig. 15 with an added ‘user interface’ modeling element as an example.

The class model in the Concrete modeling phase builds the model with more concrete modeling elements, such as class and *attribute_type* by taking into account implementation properties to the class model in the Specific modeling phase. The implementation properties represent language dependent elements (such as parameter and return type), database dependent elements, and so on.

As a result, other models in UML can be designed hierarchically using the layered metamodel in this way. This case study has shown a hierarchical modeling precisely from the Conceptual modeling phase to the Concrete modeling phase using

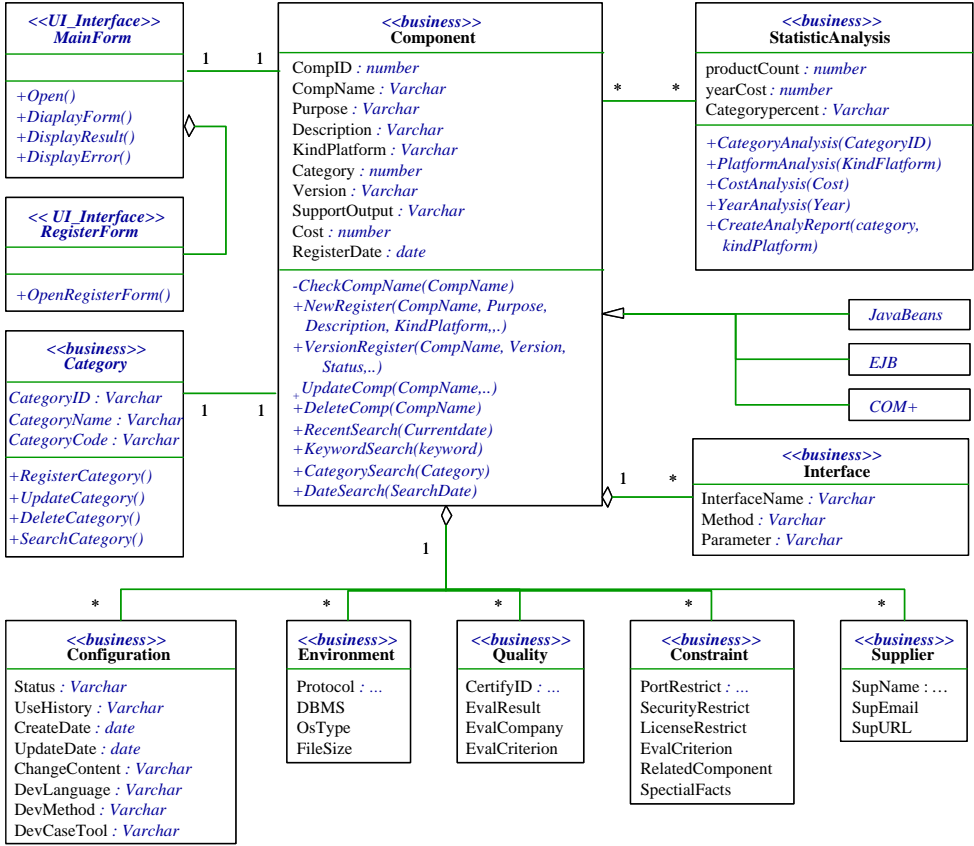


Fig. 14. Class model in Specific modeling phase for C_MDR.

the proposed modeling elements of the hierarchical metamodel. Furthermore, the metamodel is used in the referencing and verification of the designed models.

6. Assessment

With various experiments, we assess the proposed hierarchical metamodel with UML metamodel and existing methodology in terms of usability, reusability, and extendability. Usability is a measuring criterion of how many elements in the metamodel are used in software development. We knew that the elements within the proposed metamodel were actually used more than that of existing UML metamodel in designing software, such as *business_concept* class and *implementation_concept_class* in the class metamodel with precise division over modeling phases. Figure 16 illustrates the usage rate of modeling elements according to the scale of software system. As the system scale grows, the elements of the proposed metamodel are used rather than that of UML metamodel.

| | |
|--|--|
| useCase Name | Manage Component Metadata |
| Actor / UC_Type / Purpose / Description / Relate_useCase / Normal Flow / Alter Flow / Except Flow | Same with Essential Use case Description |
| User Interface | |

Fig. 15. Real Use case description in concrete phase for the 'Manage Component Metadata' useCase.

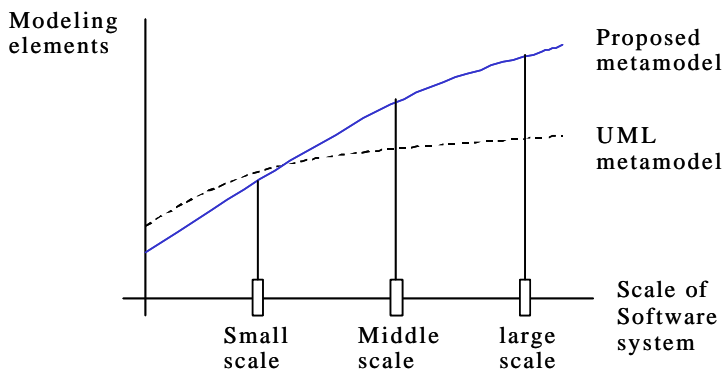


Fig. 16. Comparison results for usability through modeling elements.

Table 7. Comparison with the UML metamodel.

| Metamodel/ Measure Matrix | UML Metamodel | Hierarchical Metamodel |
|---|-------------------------------------|---------------------------|
| Systemic architecture of metamodel | Fully support | Support |
| Each metamodel | Partial support (subsystem etc.) | Fully support |
| Hierachical metamodel | No support | Support |
| Integrated metamodel by modeling phase | No support | Support |
| Understandability | Complex | Easy |
| Usability | Medium | High |

Table 8. Comparison with the existing methodologies.

| Method/ Measure Matrix | RUP | OSP | Advisor | Hierarchical Modeling |
|--|---------------|-------------------------------------|---------------|---------------------------|
| Systemic architecture of methodology | Fully support | Fully support | Medium | |
| Hierachical no modeling | No support | Partial support (class, useCase) | No support | Support |
| Checking of consistency and traceability | Difficult | Difficult | Difficult | Easy (using metamodel) |
| Reusability | Medium | Medium | Medium | High |

For reusability of application model, the constructed models are powerfully reusable according to each modeling phase at the various abstraction levels, by using the hierarchical structure of the metamodel. For extendability, the approach using metamodel type gives high extendability since the metamodel can be modified flexibly by updating variable elements for the change of UML model. We just modify the elements within target layer of updated elements owing to the hierarchical structure. The elements in the low layer do not change since it is inherited from the upper layer.

As shown in Table 7, the metamodel in UML supports well the precise description for the entire architectural structuring for all modeling elements in the UML models. However, we provide nicely a simplified hierarchical metamodel widely used in practice for the software development. Using each hierarchical metamodel, designers are able to develop the software hierarchically with regard to variety, large size, and complexity. Moreover, the designers can easily approach modeling separately per model. Hence, these improve the understandability and usability of UML by modeling based on the hierarchical metamodel.

Most methodologies apply the same model elements without regard to the design level of the development phase. The approach in this paper addressed this problem by a hierarchical modeling based on each hierarchical metamodel and the integrated metamodel. Table 8 demonstrates the excellence of hierarchical modeling using metamodels against existing methodologies. By using the integrated metamodels, our mechanism enables designers to check for consistency and traceability among the designed models which we have proved through a case study of C_MDR. Furthermore, the modular approach through a hierarchical design can enhance the reusability of software according to the abstract level.

7. Conclusion

This paper has presented a hierarchical metamodel for UML to support hierarchical modeling. We looked at each metamodel with a hierarchy by modeling phases for all models in UML, and then created three integrated metamodels combining each metamodel by the modeling phase. Moreover, one unified integrated metamodel was proposed as a blueprint for UML. For proving the effectiveness of the hierarchical metamodel, we have performed the hierarchical modeling through a case study with a C_MDR application system, and then compared it with UML metamodel and the existing methodologies.

The main contributions of the hierarchical approach for UML models are summarized as follows. First, Designers are able to make the hierarchical model by applying each metamodel with hierarchy. It enhances more reusability of application model. Second, UML models can further be used effectively, easily, and precisely. It enhances further usability for UML. Third, Consistency and traceability among the designed application models is promoted by verifying and using metamodel manually. For practical use, the metamodel can be embedded as a reference metamodel in the M2 layer on OMG UML 4-layer architecture, and be applied in practice to existing development methodologies.

For future work, we will further refine precisely the metamodels by embedding dynamic semantics. Then, the refined metamodel will be specified by a formal language like Object-Z. We will develop the tools to verify the consistency check by using the hierarchical metamodel.

References

1. Object Management Group, *OMG Unified Modeling Language Specification*, Version 1.4, September 2001, <http://www.omg.org>.
2. Object Management Group, *Meta Object Facility (MOF) Specification*, Version 1.4, April 2002, <http://www.omg.org>.
3. G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.
4. B. Henderson-Sellers, "Some problems with the UML V1.3 metamodel", in *Proc. 34th Hawaii Int. Conf. on System Sciences (HICSS-34)*, Maui, Hawaii, January 3–6, 2001.

5. A. Evans and S. Kent, "Core meta-modelling semantics of UML: The pUML approach", in *Proc. UML'99*, Colorado, USA, October 1999, pp. 140–155.
6. A. Schleicher and B. Westfechtel, "Beyond Stereotyping: Metamodeling Approaches for the UML", in *Proc. 34th Hawaii Int. Conf. on System Sciences (HICSS-34)*, Maui, Hawaii, January 3–6, 2001.
7. S. Kent, S. Gaito, and N. Ross, "A meta-model semantics for structural constraints in UML", in H. Kilov, B. Rumpe, and I. Simmonds, *Behavioral Specifications for Businesses and Systems*, Chapter 9, Kluwer Academic Publishers, 1999, pp. 123–141.
8. E. Robbins, N. Medvidovic, F. Redmiles, and S. Rosenblum, "Integrating architecture description languages with a standard design method", in *Proc. ICSE'98*, Kyoto, Japan, April 1998.
9. C. Y. Song and D. K. Baik, "An integrated metamodel and its formal specification in Z for component architecture", *Int. J. Computer and Information Science* **3** (2002) 137–145.
10. C. Larman, *Applying UML and Patterns*, Prentice Hall PTR, New Jersey, 2002.
11. P. Kruchten, *The Rational Unified Process: An Introduction*, Second Edition, Addison-Wesley, 2000.
12. Sterling Software Corporation, *Sterling Software Application Management Group, The CBD96 Standard Version 2.1*, Standards for Specifying and Delivering Software Components Using COOL: gEN, July 1998.
13. Compuware Corporation, *UNIFACE Development Methodology*, V7.2, 1998.
14. R. Veryad, "SCIPIO: Aims, principles and structure", *SCIPIO Consortium*, April 1998.
15. Metamodel.com, Metamodel, 2002, <http://www.metamodel.com>.
16. G. Nordstrom, J. Sztipanovits, G. Karsai, and A. Ledeczi, "Metamodeling — Rapid design and evolution on domain-specific modeling environment", in *Proc. IEEE ECBS'99 Conference*, Nashville, Tennessee, March 1999, pp. 68–74.
17. G. Nordstrom, "Formalizing the specification of model integrated program synthesis environment", in *Proc. IEEE Aerospace 2000 Conference*, Big Sky, Montana, March 2000, pp. 523–532.
18. K. C. Kang, W. S. Choi, and J. J. Lee, "A metamodel approach method for the description and analysis of software architecture", *Korea Information Science Society — Software Engineering Review* **13** (2000) 49–60.
19. K. L. Mills, "A knowledge-based method for inferring semantic concepts from visual models of system behavior", *ACM Transactions* **9** (2000) 306–337.
20. S. Dakhli, C. Toffolon, "A three layers software development method: Foundations and definition", in *Proc. 3rd IEEE International (ICECCS '97)*, Lake Como, Italy, September 1997, pp. 162–172.
21. C. Y. Song, S. B. Yim, D. K. Baik, and C. H. Kim, "A construction of the C_MDR (Component_Metadata Registry) for the environment of exchange the component", *Korea Information Science Society — Software Engineering Society Journal* **7** (2001) 614–629.

