ソフトウェア工学 103−10
（1995．3．9）

- A Domain-Oriented Specification Language "kusanagi" -

Toshiya Hikita
4th C&C Operations Unit 2nd Systems Development Division
2nd application systems development department NEC corporation
5-21-6 Shiba,Minato-ku,Tokyo 108,Japan
Voice:81-3-3456-7518 Fax:81-3-3456-5087 Email:hiki@asd2.sdd2.mt.nec.co.jp

Masao J. Matsumoto
C&C Software Development Group NEC corporation
Email:matumoto@ccs.mt.nec.co.jp

Abstract. In recent years, particular CASE tools/languages called the application builders has enabled major part of programming process in software development to be automated. This paper discusses the approaches to simplify specifications giving for application builder which is based on concepts and terms of business application domain. We proposed a domain-oriented specification language called "kusanagi" using knowledge of business application domain. At last this paper reports that productivity has been largely improved by kusanagi in real software developments.

1. Introduction

In recent years, particular CASE tools/languages called the application builders has enabled major part of programming process in software development to be automated. With these tools, developers can capture and realize much more requirements of user than could before, and they can shift the development process style from waterfall model to spiral model, to follow up promptly requirement changes.

Main author has playing a systems engineer, and has developed application software using application builder for several years. Now, the authors can not think developments without application builder. But, we don't think that all problem solved with it. It is always necessary to try hard to reduce human works, meaning to minimize the volume of specifications giving for the application builder. What is the minimum specification? If the size of specification is zero, it is a package software itself. It can't be customized. If the size is equal to a source program, it is a high level programming language. It means lower productivity. The answer is in between them. Reuse is one of the most promising approaches. These are two basic and important techniques for reusability: reuse of the specifications in specification description language and reuse of code fragments under thus formed specifications with program synthesizer which transforms automatically the specifications to an executable code[MAT91].

This paper discusses the approaches to simplify the specifications giving for an application builder using concepts and terms of such particular business application domain. An example of simplified specifications is domain-oriented specification language "kusanagi". Domain specific approach has trade-off in system analysis/design process as compared domain generic approach(See fig. 1). Principal advantage of domain specific approach is reusability of common design works in the domain. Kusanagi is a technique to reuse of specifications and code fragments in business application domain. At last this paper reports some evaluation results showing how effectively kusanagi coud be used in real software developments.

	Domain generic approach	Domain specifc approach
Range of application	Large	Small
Design flexibility	Large	Small
Design complexity	Complex	Simple

fig. 1. Trade-off of domain specific approach

2. Basic concept of "KUSANAGI"

Kusanagi is a specification language on the basis of the object-oriented

analysis/design techniques[RUM91][EMB92]. Kusanagi used concepts in these works: object, operation, relationship, etc. And it made design model with these concepts. The principal aim of kusanagi is to reuse the specifications. If the specifications are complicated, they must be hard to reuse, because it is difficult to find, understand and adopt them. Here are two approaches to simplify the specifications: domain-oriented and dual level reuse.

The first is domain-oriented approach. Restriction of range or bounds of problems prompt sometime a drastic progress. This approach is to solve problems by using some concepts of business application domain. The authors defined domain as unit of business such as sales management, accounting, production control, etc. and super-domain as business application domain. Kusanagi specialized in this super-domain. In kusanagi, major behavior of business process is to record transactions, and business application systems are composed of three kinds of object classes: MasterFile(means a master file as defined in MIS), TransFile(a transaction file) and ReportFile(a report file). Each object classes has predefined several operations. These operations are prepared by a program synthesizer in the support tool of kusanagi as code fragments. Section 3 discusses this in more detail.

The second is introduction of the dual levels of reusable artifact: object and framework. Framework is a set of objects related each other. It is made up by objects and relationships between objects. Reuse of objects is local reuse, but reuse of framework is global reuse. With kusanagi, systems designer can build two libraries: object library and framework library. Reuse of framework less difficult the reusability. Section 4 discusses this in more detail.

3. Domain-oriented approach

Kusanagi restricts the objects and the relationships in business applications as follows. These restrictions simplify systems design process and to understand for business user.

3.1 Objects

Kusanagi has only three kinds of objects: MasterFile, TransFile and ReportFile. These three kinds are defined as abstract classes. Abstract class is a class who has not their instances. All objects in kusanagi will be defined subclasses of these abstract classes.

We first explain about usage of these objects and how they are different from each other. The MasterFile is an object using for storage of information. It corresponds

with the "thing" such as client, merchandise etc. And it is referenced by persons or sections concerned. The TransFile is an object using for transmission of information. It corresponds with "relation between one thing and another things" as order ,claim, etc. Thus it is used for communication between sections in a company or a company and its client. The ReportFile is an object using for analysis of information. It corresponds with "information" such as monthly-report, sales-results, etc. And it is printed on papers to deliver to each section.

We next explain about structure of these objects and how they are different form each other, using concepts of relational model[COD70] such as attribute, function dependency, primary key, external key, etc. MasterFile has a flat structure with primary key and other attributes having function dependency(FD) to primary key. The primary key of MasterFile aforementioned is those that not be included in external keys. TransFile has a "body" and "details". The body-part has a primary key called masterkey, and other attributes in body-part having FD to primary key. The detail-part has a primary key, called slavekey constituting masterkey and over one of external key as primary key of related MasterFile or other TransFile. The detail-part of TransFile is a complex entity in entity-relationship model. Let's call elementary attribute and process attribute as data come from external resource of system such as human, and data produced by elementary attributes. ReportFile is constituted by process attributes only. In many systems, ReportFile will be implemented as a operation of TransFile. But, in kusanagi a ReportFile is considered an independent object. Because, ReportFile has often one for many relationships with TransFiles.

3.2 Relationships

Kusanagi supports five relationships: is-a, is-part-of, predicates, concerns, reports. Is-a is generalization-specialization relationship. Is-part-of is whole-part relationship. Predicates is a n-array relationship. Concerns and reports are 1:N relationships. Is-a and is-part-of use only for between same kind of objects. Predicates use for between one TransFile and over one MasterFile or TransFile. Concerns use for between one TransFile and one MasterFile. Reports use for between one ReportFile and over one TransFile or MasterFile.

4. Specification language

What is a specification language? We classify elements of code in programming language into five categories: data definition, constraints of data, relationships between data, control logic and styles. Styles are elements for human interface in this paper, e.g.,

screen layout, items in menu bar, etc. We suppose that the specification language doesn't include the elements for only implementation, i.e., control logic and styles. Control logic are for implementation. In kusanagi, skeleton which is reusable code fragment includes all variation of control logic needed. Of course, the human interface is very important factor to specify a software, but this factor depends on the implement environment. So, we preferred to make the styles with an interface builder as a kind of CASE tool.

The rests of categories are data definition, constraints of data and relationships between data. We subordinate these elements to data itself, in a object. The specification language includes not only these elements, but the relationships between modules (or objects). It is clear that these relationships can't encapsulate in a object. It is necessary the descriptions over objects. Kusanagi has two levels of description: object description and framework description which describes relationships between objects.

4.1 Object description

Object description is the specifications of the object that exist in a system. Object description consists attribute definition and method definition(See Fig. 2).

Attribute definition includes data definition, constraints of data and relationships between data as name of attribute, domain and deduction. The domain means type of attribute, range of value and integrity constraints of attribute. The integrity constraints are not null constraint, existence constraint, etc.. See (*A) in Fig. 2. The deduction means defaults value, retrieval value of other attribute ,calculation formula or SQL formula. See (*B) in Fig. 2.

Method definition includes name of method, method type and conditions for selecting instance. Method type are predefined, and restricted by kind of object. See (*C) in Fig. 2.

```
OBJECT:Client
        Client#[key]:char(4);NOTNULL  .....(*A)
        ClientName:char(40);NOTNULL
        METHOD:Client_mtn TYPE=Maintenance
        METHOD:Client_list TYPE=List COND={ALL}

OBJECT:Sales
        [BODY]
        Contract#[Masterkey]:char(6);NOTNULL
```

```
Client#:LIKE Client.Client#;EXIST(Client)
ClientName:
        ::Client.ClientName
Date:date
        ::DEFAULT(TODAY)
Charge#:LIKE Employee.Employee#;NOTNULL
Comments:char(100)
[DETAIL]
Merchandise#[Slavekey]:LIKE Merchandise.Merchandise#;NOTNULL
MerchandiseName:
        ::Merchandise.MerchandiseName
Quantity:integer;NOTZERO
        ::DEFAULT(1)
Price:decimal
        ::DEFAULT(Merchandise.UnitPrice*Quantity)  .....(*B)
METHOD:Sales_mtn TYPE=Maintenance
METHOD:Sales_prt TYPE=Print
METHOD:Sales_exp TYPE=Export COND={Date>=ARG(1)}  .....(*C)
```

Fig. 2 Example of object description in kusanagi

4.2 Framework description

Framework description is the specifications of relations between these objects(See Fig. 3).

```
FRAMEWORK:SalesManagement
        Corporate is-a MasterFile
        Client is-a Corporate
        Merchandise is-a MasterFile
        Sales is-a TransFile
        Sales predicates Client,Merchandise
        Employee is-a MasterFile
        Employee concerns Sales
        MonthlyRep is-a ReportFile
        MonthlyRep reports Sales
```

Fig. 3 Example of framework description in kusanagi

This example represent all information of an object-diagram(Fig. 4).

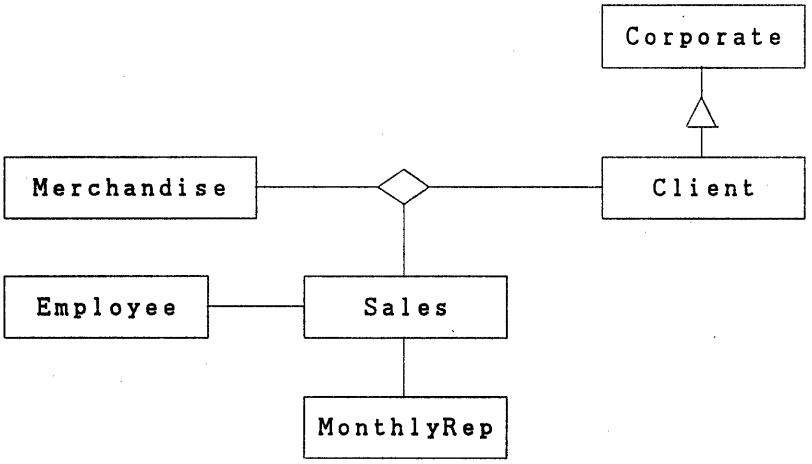

Fig. 4 Object diagram of SalesManagement system
(the description rules are based on [EMB92])

We can define is-a relationship between a framework and another. It means a description with differentiation be able to use(See Fig. 5)

FRAMEWORK:SalesPurchases is-a SalesManagement
Supplier is-a Corporate
Supplies is-a TransFile
Supplies predicates Merchandise,Supplier
Stock is-a ReportFile
Stock reports Supplies

Fig. 5 Example of framework description using differentiation

We get now a simple method for global reuse of specification designed.

5. Example in a real software development

Kusanagi was used for a real software development of tenant management

system. This system is a kind of the sales management systems. The outline of this application is to record contracts of leased floors and to print out the invoices for tenants every month. Size of application is 109 programs, converting into COBOL about 114KLOC(kilo line of codes). The rate of productivity comparing traditional method is 1/40.9. This rate is much more better than SEA/I[MIZ86] for a program synthesizer. Only three programs couldn't make with kusanagi.

Before construct this system, we attempted a domain analysis for sales management system to reuse the result. We got about 0.7 KLOC of framework and objects descriptions, and we reused about 0.5 KLOC. New description is about 1.0 KLOC. This means 1/3 of whole system is reused.

6. Conclusions

The specification using domain knowledge simplify their description. We proposed a domain-oriented specification language and its support tools using knowledge of business application domain. This language had applied for real software developments, and had good results for productivity and reusability. This project is in progress to verify in various systems of business application domain.

References

[COD70] Codd,E.F.:A relational model for large shared data banks, CACM 13, pp.337-387(1970)

[EMB92] Embley,D.W., Kurtz,B.D., Woodfield,S.N.:Object-oriented systems analysis -A model-driven approach,Prentice-Hall 1992

[MAT91] Matsumoto,M.:Specifications reuse process modeling and case study-based evaluations, IEEE COMPSAC 91, pp.499-506(1991)

[MIZ86] Mizuno,Y., Matsumoto,M.:Software development technology innovations",Proc. of international conference on software engineering environment, Aug.1986

[RUM91] Rumbaugh,J., Blaha,M., Premerlani, W., Eddy,F., Lorensen, W.:Object-oriented modeling and design, Prentice-Hall 1991