

Model and Function Driven Development*

Ruth Raventós

Universitat Politècnica de Catalunya
Dept. Llenguatges i Sistemes Informàtics
Jordi Girona 1-3, 08034 Barcelona
raventos@lsi.upc.es

Abstract. This paper proposes a new Information System development approach that may provide a substantial increase in Information Systems development productivity and, at the same time, facilitates changes to accommodate new functional requirements. The new approach is called Model and Function Driven Development. Its main features are the distinction between the model and the function of a Conceptual Schema, its reuse of generic conceptual schemas, and its assumption that the domain layer of the Information System architecture is obtained by an automatic mapping from the conceptual schema in the context of the OMG's Model Driven Architecture (MDA).

1 Introduction

Despite the existence of a seemingly continuous stream of new technologies and methods, according to Nierstrasz [4], software productivity remains universally unimpressive. One of the main reasons is that most methods fail to take into account that the most cost-intensive phase of any successful project is “maintenance”, which involves in practice continuous development and software evolution.

This paper proposes a new development approach focus on reusing conceptual schemas (also called domain models or functional requirements) and on the automatic generation of code from the conceptual schemas. We approach this generation in the context of the OMG's Model Driven Architecture (MDA), where it corresponds to an automatic mapping [3]. We call the new approach Model and Function Driven Development (MFDD), and it can be summarized as follows (see Fig. 1):

- We distinguish two parts in a Conceptual Schema (CS): the model and the function. The model defines the knowledge on the domain that the Information System (IS) needs to know. The function defines the knowledge on the functions the IS must perform.
- The CS of a particular IS, called specific CS, is obtained by refining a generic CS. The specific model is obtained by refining the generic model, and the specific function is obtained by refining the generic function.

* This work has been partially supported by the Ministerio de Ciencia y Tecnología and FEDER under project TIC2002-00744.

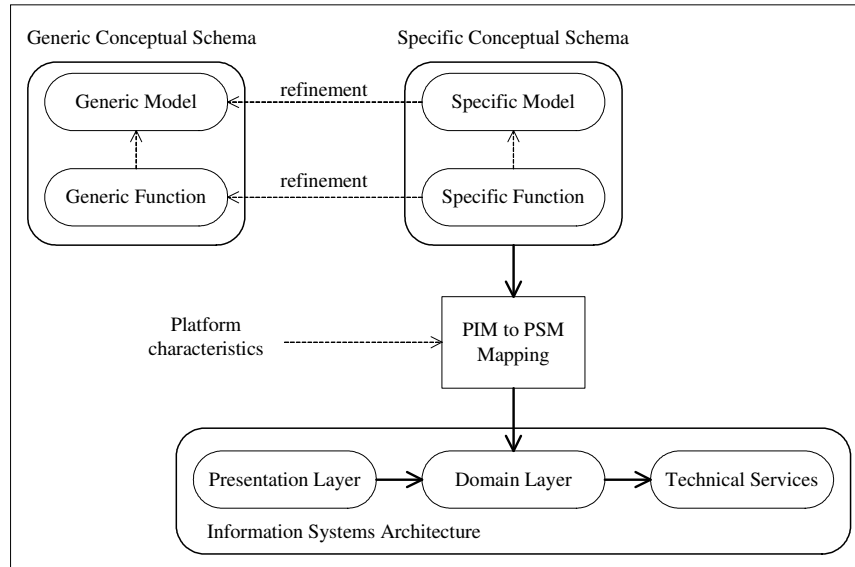


Fig. 1. Model and Function Driven Development.

- The CS thus obtained is input to an automatic PIM-to-PSM mapping, which generates the domain layer of the IS. The domain layer uses the technical services (persistence, transactions, security) that it needs.

Next section reviews the main elements of a CS and makes the distinction between model and function. In Section 3, we deal with the definition of the model part of a CS in UML [7]. We propose a method for the definition of generic model parts of CSs, and we explain how to refine such generic models into specific ones. Section 4 does the same for the function part of a CS. Section 5 contains the (preliminary) conclusions and summarizes the work that remains to be done.

The paper includes examples from the case study used to test our proposal. The case study deals with electronic auction systems. We plan to develop a generic CS by abstracting from two existing auction systems (www.eBay.com and auctions.yahoo.com) and taking into account the work made by [9]. We also plan to evidence that we may obtain three different auctions systems applying the MFDD approach (www.eBay.com, auctions.yahoo.com and auctions.Amazon.com).

2 Conceptual Schema = Model + Function

The conceptual schema (CS) of an IS includes two kinds of knowledge: about its domain and about the functions it must perform. The former is called the model and the latter the function. Every piece of knowledge in the CS is part of either the model or the function. We then assert that the model and the function are the two components of a CS, and represent this by: ***Conceptual Schema = Model + Function.***

The model defines the general knowledge on the domain, independent from the IS. The model consists of the entity and relationship types, the integrity constraints, the derivation rules and the domain event types with their effects.

The function defines the types of the query events the IS has to respond to, the generating condition of the generated query events and the generating conditions of the generated domain events.

The distinction Model/Function is not a new one. The pioneering work was probably [2] with the fundamental principle: "...the developer must begin by modeling this reality (the real world), and only then go on to consider in full detail the functions which the system is to perform". The originality of our research is that we adhere strictly to the distinction using a modern object-oriented conceptual modeling language (UML), and that we make an additional, orthogonal, distinction between a generic and a specific model and function, as we explain in the next sections.

3 GENERIC AND SPECIFIC MODEL

3.1 Generic Model

Let's assume a domain (such as the auctions) and a set of companies working in that domain (such as eBay, Amazon or Yahoo). Each of these companies has an IS with a corresponding CS. As we have seen, each of these CSs has a model and a function. Then, a *generic model* of a given domain consists of the part of the model that is present (or must be present) in all or many CSs for that domain. Fig. 2 shows an example of a fragment of the structural schema (in UML) for a generic auction model. Marketplace items that are auctioned are classified, depending on different criteria, into several categories and themes which may be decomposed into subcategories and subthemes, respectively. Items are located at some place and may be available for buying at several locations.

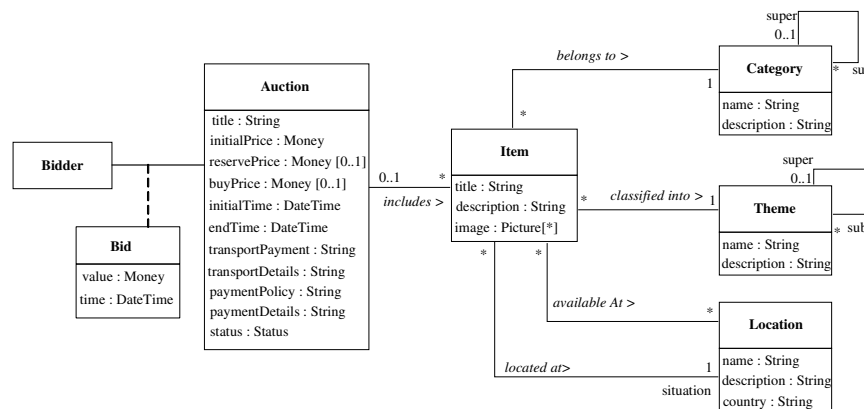


Fig. 2. Fragment of the structural schema for a generic auction model

Domain events may be represented in several ways in an IS. In this paper we represent events as entities like Olivé in [5]. This allows the definition of relationships between events and other entities, integrity constraints, derivations rules, etc. similarly to ordinary entities. In UML, we represent event types as a special kind of entity type. We use the stereotype `<<event>>`. An entity type with this stereotype will define an event type.

The creation of an auction domain event corresponding to the structural schema defined above is shown in Fig. 3.

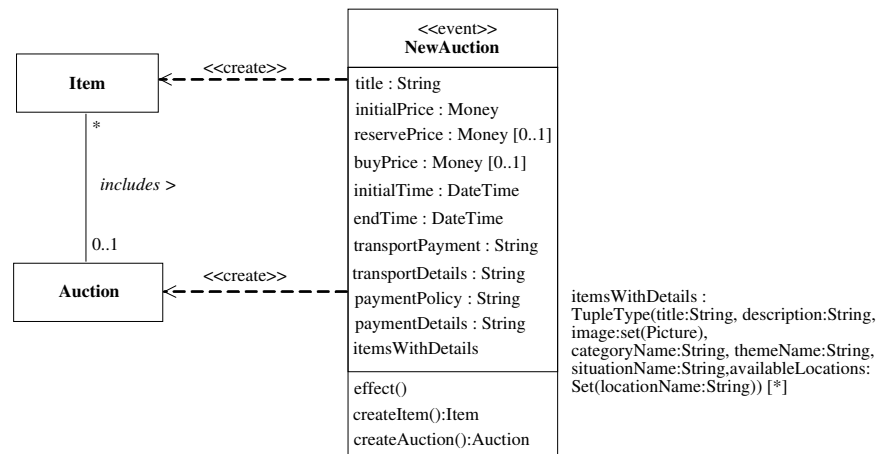


Fig. 3. New Auction domain event in the generic model

The effect of the *NewAuction* is that a set of new items are created and put for selling in an auction. *NewAuction* includes the operation *effect()*, that defines the behavior of the domain event calling two operations: *createItem()*, which creates a new item, and *createAuction()*, which creates the auction for selling the items. In the MFDD approach, these operations must be formally specified in some language. We use for this purpose the OCL [6]. For space reasons, we include here only the specification of *createAuction*:

```

context NewAuction::createAuction():Auction
post : a.ocIsNew() and a.ocIsTypeOf(Auction) and a.title = title and
a.initialPrice=initialPrice and a.reservePrice = resevePrice and
a.buyPrice = buyPrice and a.initialTime=initialTime and
a.endTime=endTime and a.transportPayment=trasportPayment and
a.transportDetails=transportDetails and a.paymentPolicy = paymentPolicy
and a.paymentDetails=paymentDetails and a.status = Status::defined
    
```

3.2 Specific Model Refined from the Generic Model

In the MFDD approach, the specific model for a particular IS is obtained by refinement of the generic model, after using a predefined set of domain-independent schema transformation operations.

In UML, the refinement may include the addition and removal of entity types, attributes, associations, derivation rules, integrity constraints and domain event types.

Fig. 4 shows the fragment of the Amazon structural schema obtained by refinement of the generic one shown in Fig. 2. The refinement has consisted of the following operations: (i) redefining cardinality of the attribute *image* of *Item*, from multiple to one, (ii) removal of entity type *Theme*, (iii) removal of *title*, *reservePrice*, *buyPrice* and *paymentPolicy* attributes of *Auction*, (iv) removal of *name* and *description* attributes of *Location*, and (v) adding the new attribute *USzipCode* to *Location*.

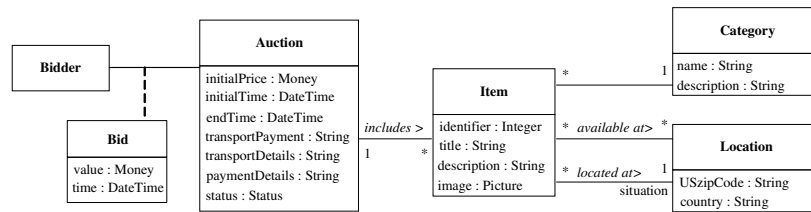


Fig. 4. Fragment of the structural schema of the Amazon model

The refined domain event type *AmazonNewAuction*, as a result of the transformation operations applied to the structural schema, is shown in Fig. 5

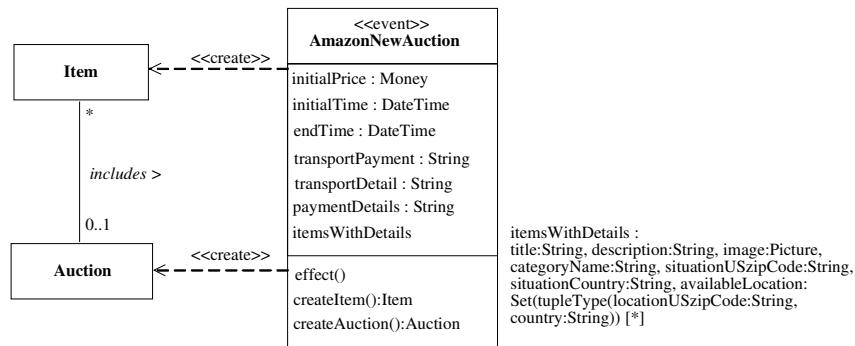


Fig. 5. NewAuction domain event for the Amazon model

The refinement has consisted of: (i) redefining cardinality of the *image* attribute of the *itemsWithDetails* attribute, from set to one, (ii) removal of *title*, *reservePrice*, *buyPrice* and *paymentPolicy* attributes of *AmazonNewAuction*, (iii) removal of *name* and *description* attributes corresponding to the situation of the *itemsWithDetails* attribute, (iv) adding the new *situationUSzipCode* attribute to the *itemsWithDetails* attribute, (v) removal of the name and description attributes corresponding to the available locations of *itemsWithDetails*, (vi) adding the new *locationUSzipCode* attribute to the available locations of *itemsWithDetails* attribute, and (vii) refinement of the specification of *createItem* and *createAuction* operations.

4 GENERIC AND SPECIFIC FUNCTION IN UML

4.1 Generic Function

Similarly to the generic model, a *generic function* of a given domain consists in the part of the function that is present (or must be present) in all or many CSs for that domain.

We may find, similarly to frameworks, that various aspects of an application cannot be anticipated and therefore they may be kept sufficiently flexible to be adapted to any particular needs. This problem was defined in object-oriented design by [8] as *hot-spots*, that is, those aspects of an application domain that have to be kept flexible. UML does not offer any special feature to indicate whether any classifier or operation is a hot-spot. However, UML provides extension mechanisms, such as stereotypes, tags and constraints that allow us to define (virtual) new metaclasses by adding some additional semantics. Therefore, by using this extension mechanism we may adapt the concept of hot-spot, template methods and hook methods of object-oriented design defined in the Template Pattern [1, 8].

In the generic function we define hot-spot query event types that include *template* and *hook* operations. A template operation is an operation that provides the skeleton of the behavior of the event and calls hook operations that must be adapted in the refined event type by adding, replacing or overriding the postconditions of their specification. Template and hook operations are marked in the UML, with the `<<temp>>` and `<<hook>>` stereotypes respectively.

In the auctions domain, users may search an item by browsing categories or by browsing themes, and list items within the selected category or theme as shown in Fig. 6. The effect of the *BrowseCategories* event is to show a list of all the subcategories of the category selected and to list all the auctioned items within such category. *BrowseCategories* includes the *template operation*, *effect()*, that provides the skeleton of the behavior of the query event calling two *hook operations*, *showCategories()* and *ShowItems::showItem()*. The former list all the subcategories of the category selected and the latter shows some details of a given item in such a category. These two hook operations have to be redefined to be adapted to each application needs and allow the designer to specify in a different manner such operations without invalidating the CS for generic IS.

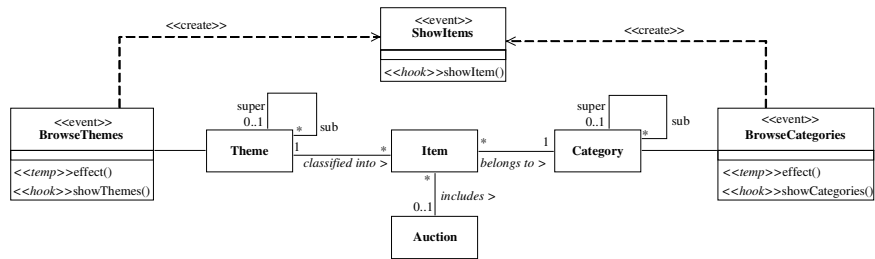


Fig. 6. Fragment of the generic function for Auctions

The specification of the effect operation of the *BrowseCategories* event is as follows:

```

context BrowseCategories::effect ()
post : self^showCategories() post : category.allDescendants.item->
forAll(i|ShowItems.allInstances-> any() ^showItem(i))

```

4.2 Specific Function Refined from the Generic Function

The representation of the specific function in UML is similar to the generic function (redefining, of course, hot-spots). The refinement of the specific function requires two steps, (i) to adapt the function schema to the refined model (we call the resulting schema refactored function schema) and (ii) to apply the transforming operations needed to reach the desired requirements. The refactoring of the generic function consists in the removal of the *BrowseTheme* event.

The example of the fragment of the refined function schema of Amazon is shown in Fig. 7. The refinement, in this case, is the specification of the *answer* given by both hook operations.

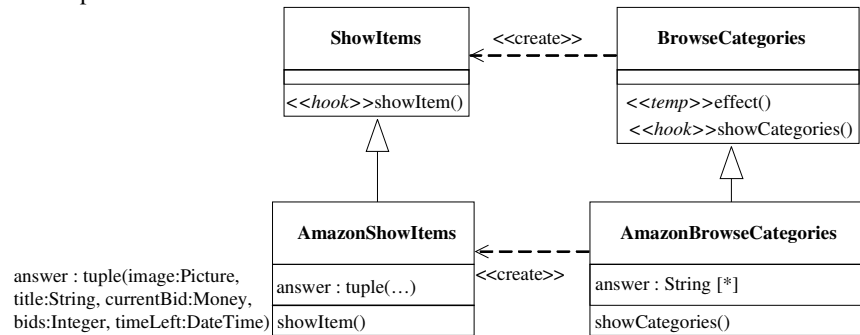


Fig. 7. Fragment of the refined function schema for Amazon

5 CONCLUSIONS

We have presented the Model and Function Design Development (MFDD) approach to IS development. The originality of the approach lies on its distinction between the model and the function of a CS, its reuse of generic CSs, and its assumption that the domain layer of the IS architecture is obtained by an automatic mapping from the CS. Our working hypothesis is that the MFDD approach may provide a substantial increase in IS development productivity and, at the same time, facilitates changes to accommodate new functional requirements. Most of the changes concern the functions the IS performs. Our distinction between model and function should ease the changes.

The MFDD approach has several advantages over other approaches: (i) it does not constrain or limit the customization of the final product, (ii) it provides designers with

a generic schema within the same domain as starting point and facilitates the traineeship in such a domain, (iii) the set of predefined transformation operations helps designers on the refinement, (iv) design and code may be automated, (v) separation of model and function eases the location of the parts of the CS to change when adapting to new requirements, (vi) most of the set of predefined operations used when refining may also be used for any future change of the CS and (vii) design and code of the changes referred in (v) and (vi) may also be automated.

The objective of our research in progress is to show that a method based on the MFDD approach is both feasible and convenient. We have presented in this paper our preliminary thoughts on such a method. We have used UML as conceptual modeling language, but our method should be adaptable to other languages.

Naturally, further investigation remains to be done. The method must be developed in full detail, including all (or most of) the refinement operations. These operations can be used not only in the development of new ISs, but also in later changes. We realise that the most cost-intensive phase of any successful software project is “maintenance” [4], and that therefore much may be gained by providing adequate support to it.

To test our ideas, we plan to develop a complete generic model and function for the auction domain, and to show that the whole specific model and function for (at least three) particular ISs can be obtained with the application of those refinement operations.

References

1. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. Design-Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
2. Jackson, M. System Development. Prentice/Hall International, Englewood Cliffs, N.J., 1983.
3. Kepple, A., Warner, J. and Bast, W. MDA Explained The Model Driven Architecture: Practice and Promise. Addison-Wesley, 2003.
4. Nierstrasz, O., Software Evolution as the Key to Productivity. in Radical Innovations of Software and Systems Engineering in the Future, 9th International Workshop, RISSEF 2002, Venice, Italy, October 7-11, 2002, Revised Papers, (2004), Springer-Verlag Berlin Heidelberg 2004, 274-282.
5. Olivé, A., Definition of Events and their Effects in Object-Oriented Conceptual Modeling Languages. in 23rd International Conference on Conceptual Modeling (ER 2004), (Shanghai, China, 2004).
6. OMG. UML 2.0 OCL Specification. Group, O.M. ed. Doc. (ptc/03-10-14), 2003.
7. OMG. UML 2.0 Superstructure Adopted Specification. Group, O.M. ed. OMG Adopted Specification (ptc/03-08-02), 2002.
8. Pree, W. (ed.), Design patterns for object-oriented software development. ACM Press/Addison-Wesley Publishing Co., 1995.
9. Ré, R., Braga, R.T.V. and Masiero, P.C., A Pattern Language for Online Auctions Management. in Proceedings of the 8th Pattern Languages of Programs Conference (PLOP 2001), (Monticello-IL, USA, 2001).