Abstract

We propose a software development method for enterprise applications that combines the use of the structural concepts provided by problem frames, and the use of the UML notation. Problem frames are patterns that provide a precise conceptual model of what is the problem to be solved.

The first step of our method is to match the current task with one of the problem frames that we propose for enterprise applications, and this helps to understand the nature of the problem under study. The problem frames to be considered for enterprise applications are clearly more complex than the basic ones. We then provide guidelines to develop all the artifacts required by the method through a dedicated choice of appropriate UML diagrams together with predefined schemas or skeletons for their contents. Thus, using our method provides a more direct path to the UML models, which saves time (no long questions about which diagrams to use and how) and improves the models quality (relevant issues are addressed, a uniform style is offered). In this paper, we consider the phases of modelling the domain, the requirements capture and specification, and their relationships.

Keywords: Domain Model, Requirements Specification, Enterprise Applications, Problem Frames, UML.

1 Introduction

Enterprise Applications are complex systems so their development requires appropriate concepts. M. Fowler [5] describes them as follows: “Enterprise Applications are about the display, manipulation and storage of large amounts of often complex data and the support or automation of business processes with that data.”

One of the difficult issues is to start the analysis of a complex problem. Since patterns are “ready-to-use” structures drawn from experience, it is now quite widespread to use them to help systems development. Various kinds of patterns are available, problem frames propose an overall problem structure [8], architectural styles [13] are useful to provide an overall structure for the system, while design patterns [6] are more appropriate when structuring the design before coding, thus patterns may differ in granularity. There may also be patterns identified as particularly relevant for some classes of applications.

Here, we consider M. Jackson “Problem Frames” [8] that can be used by themselves or in combination as a good tool to tackle with a first structuring of problems. Problem frames differ by their requirements, domain characteristics, involvements, and frame concern. For each problem frame, a diagram is settled, showing the involved domains, the requirements, the design, and their interfaces. Five basic problem frames are provided [8] together with some variants. Problem frames are also presented with the idea that, once the appropriate problem frame is identified, then the associated development method should be given “for free”.

While the structuring concepts brought by problem frames seem highly valuable to help start the development effort, by showing clearly the items to consider and the general tasks to do, we propose development methods associated with them.

Now, since the UML notation [10, 11] is widespread and also carries valuable concepts experienced in practice, we proposed in [4] for the various basic frames a development method using UML as a notation.

There are some drawbacks with the use of UML. While it provides a nice variety of constructs, it may be difficult to choose which are appropriate. There are no means to fully insure the consistency between the different views used to build a model, and, moreover, the UML semantics is both informal and problematic. However, in [1, 2], it has been shown that UML may be used in a development method in a quite precise, structured and well-founded way, so as to avoid most typical problems (e.g., only a subset of UML is used and its semantics may be formally expressed).

In the work presented here, we propose a new problem frame, the Enterprise frame, composed of two parts (the Business Frame, devoted to the domain description, and the EA Frame), that we devised for Enterprise Applications, together with its associated development method based on the UML.

Enterprise applications are quite complex and large and come in several variants, thus it is not possible to propose
for them a simple frame composed with few elements as the basic ones in [8]. First of all we extend the notation used by Jackson to present the frames, adding provision for composite phenomena and domains, see Sect. 2. In this work, we do not focus on the decomposition issue (it may be necessary to decompose a complex problem into subproblems, for instance to match basic problem frames) which is complementary to our approach. We provide a larger scale frame to match directly the problem of the development of the application at hand.

In problem frames presented by M. Jackson [7, 8] there is a distinction between existing domains and the system to be built as a new part in that world. This implies that the various entities considered in the existing domains are not modified (or removed) when the new system is introduced. In our understanding, an enterprise application (shortly EA) may introduce some change in the preexisting context, the business, for instance by replacing some entities (e.g., when a clerk is replaced by the software system). Thus, in order to propose a problem frame for enterprise applications, we need first to describe the business domain that will be managed by the EA. We then propose an Enterprise frame composed of two parts:

- A *Business (domain) Frame* describing the business (in large) to be managed by the EA. This business framework helps to precisely understand the business considered together with its business rules (that are often quite subtle). Having a separate business frame also promotes its reuse in many different systems.
- A “classical” problem frame, the *EA Frame*, with the EA machine, its domains and requirements.

Then, we present how, from the Business Frame, we can derive the corresponding UML model, the Business Model, and from the EA Frame, the Requirement Specification, again a UML model. In this paper, we do not consider the EA design task we just give a hint, to show that the structuring power of the (business and EA) frames yields guidance for the design step, and helps to trace the requirements.

In Sect. 3 we present the Business Frame and the associated Business Model, and in Sect. 4 how to derive the EA Frame from the Business Frame and the associated UML specification of the requirements. In the paper, we use as a running example a small e-commerce site, e-commerce domain (the software to be built) is denoted by a box with a double stripe, a *designed domain* (data structures that may be freely designed and specified) by a box with a single stripe, and a *given domain* (a problem domain whose properties are given) by a box with no stripe.

Letters in the lower right corner reflect a coarse classification of domains - lexical (data), biddable (people, with no predictable internal causality), or causal (predictable causality, controlling and controlled by some phenomena).

A solid line connecting two domains is an interface of shared phenomena. Below, phenomena p1 is controlled by domain D1 and is shared between domains D1 and D2.

Requirements are denoted by a dashed oval, and a dashed line connecting a domain and a requirement is a requirement reference (constraining if with an arrow).

A *frame* (diagram) is a graph the nodes of which are domains, and its arcs are interfaces of shared phenomena. A *problem frame* is a particular frame including at least a machine and a requirement domain.

When considering complex systems, we found it useful to provide some extensions to this notation, as shown below.

When there are several instances of a domain, the usual multiplicity notation (* or n..m) is a useful graphic abbreviation. The notion of service S groups a set of phenomena to be taken into account. We also found it useful to have a notion of internal (non sharable) phenomena i−ph, and of (external) sharable phenomena e−ph.

It is possible to connect several domains with an hyperarc to denote a complex interaction built out of various basic phenomena, that we name *composite phenomena* CPH.
3 Enterprise Applications: the Business (domain) Frame

In this section we introduce the Business Frame and the associated Business Model, illustrating both with an application to the $\mu$EC case study.

3.1 Business Frame

The business domain of an enterprise application is usually quite complex and should be accurately understood and modelled before starting the application development.

We assume that the business consists of various interacting entities that achieve the various business specific tasks. Technically, the business will be schematically structured by means of a frame (cf. Sect. 2), with a domain for each entity of the business, and where interactions (here, business procedures) are described in terms of composite phenomena.

The Business Frame domains are of three categories:
- business objects (marked by BO), the entities which are the business subject, and are causal domains;
- business workers (marked by BW), the entities acting in the business, and are biddable domains; here we do not distinguish further (other approaches, e.g., [9], distinguish between those working for the business and the business actors);
- and external systems (marked by EC or EB), entities external to the business used for outsourcing some activities (e.g., messaging, mail) or for taking advantage of data provided by already available systems (e.g., a system giving information about the credit history of people). EC are causal domains, whereas EB are biddable domains.

The biddable domains should have at least one internal phenomenon, and the causal ones should have at least one external phenomenon and cannot have internal phenomena.

In a Business Frame the domains are connected by composite phenomena that correspond to the business relevant procedures, and are referred to as business cases. The domains connected in a business case are called participants of the case. Notice that we do not use the term “business use case” as in other approaches, e.g., in [9], because, we do not model the business from the viewpoint of outside interactions. We consider a business case as a cooperation between business workers, business objects and external systems. We prefer this approach, since it does not require to fix at this point the boundaries of the business under investigation, thus we can get a more abstract description. Moreover, a description of the business thus produced can be reused for the development of various different applications, becoming thus a more valuable asset.

3.2 $\mu$EC case: Business Frame

Fig. 1 shows the Business Frame for our e-commerce example $\mu$EC. This frame exhibits business workers (e.g., Manager), business objects (e.g., Orders), and external systems (e.g., Factory), and complex business cases, e.g., Put order is a business case in which Client, Orders, Catalogue and the Stock take part. To avoid to clutter the

![Figure 1: $\mu$EC Business Frame](image)

Business Frame diagram by listing all the internal and external phenomena of the various domains, we modularly decompose it by giving for each business case a separate diagram showing all the related phenomena. For example, in Fig. 2, the Put order business case is described in detail some others can be found in Appendix A.1).

![Figure 2: $\mu$EC Business Frame: Put order Business Case](image)

3.3 (UML) Business Model

Given a proper frame for the business of interest, we should now describe its component domains and business cases. To this end, we chose to use UML with the precise method of Astesiano-Reggio [1, 2]. This method proposes a precise way to model in general the domain of a software system, which can be specialized to the particular case of enterprise applications as shown in what follows.

Given a Business Frame BF, the associated Business Model is a UML model consisting of:
– A class diagram with a class for each domain appearing in BF, this class is active for the biddable domains and passive for the causal ones. We use three stereotypes to indicate for each class which is its role w.r.t. the business, precisely \textit{\llangle bo \rrangle} (for “Business Object”), \textit{\llangle bw \rrangle} (for “Business Worker”) and \textit{\llangle es \rrangle} (for “External System”), according to the domain markings BO, BW and EB, EC. For each internal phenomenon there will be a private operation in the corresponding class; the internal phenomenon occurrences correspond to self-calls of the associated operations. In order to stress that they correspond to autonomous acts, we use the operation stereotype \textit{\llangle A \rrangle}. The external phenomena, to which a domain must react, will be modelled by operations of the corresponding class. Obviously, the classes in this diagram may have attributes and other operations, and may be defined using other classes (e.g., datatypes).
– Some behaviour views modelling the behaviour of the classes introduced in the class diagram. The behaviour of an active class is given by a statechart, where the passive classes are modelled through their operations behaviour.
– A description of each business case, by means of a UML collaboration summarizing the case participants and possible parameters, and by an activity diagram, where action states may only contain calls of the operations of the participants, and conditions must be built using only the operations and attributes of the participants.

3.4 \textit{\mu}EC Business Model

Fig. 3 presents the \textit{\mu}EC Business Model class diagram. It contains a class for each domain in the \textit{\mu}EC Business Frame (cf. Fig. 1) appropriately stereotyped. On the top of the diagram there are some nonstereotyped classes introducing the data used by the others (e.g., \textit{Product} and \textit{Order}).

\begin{verbatim}
method newProd(PI,E,S) {P = create(Product); P.id = PI; P.price = E; P.descr = S; P = ps->including(P)}
context deleteProd(PI) post: not ps.id ->>including(PI)
context changePrice(PI,E) post: ps->select(id = PI).price = E
\end{verbatim}

Figure 4: \textit{\mu}EC Business Model: Catalogue Behaviour View

Fig. 4 presents the Behaviour View associated with the business object class Catalogue. Since it is a passive class we just modelled its operations, one by a method and the others by post conditions. The other passive class behaviour views are in Appendix A.1. We do not report the behaviour views associated with the other (active) classes, since we do not have information on the way they behave.

In Fig. 5 we present one Business Case, precisely “Put Order” (some others are in Appendix A.1). The collaboration shows which are the participants in the business case, a business worker (the client), some business objects (the catalogue, the orders and the stock) and an external system (the payment system), and that it is parameterized by the client, the bought product and the bought quantity. The activity diagram shows how after the autonomous act of the client to put an order, if the payment system grants the needed money and if the chosen product is in the stock, the order will be accepted. Notice, how this diagram fully describes the business logic; for example, it is clear that an order cannot be cancelled, and that no reason for refusing it is given.

4 The EA Problem Frame

In this section, we first present the problem frame related to the development of an Enterprise Application (shortly EA in what follows) and how it can be derived from the underlying Business Frame. Then, we show how to build the corresponding Requirement Specification using the UML.
4.1 Problem Frame

We report the problem frame for the Enterprise Application in Fig. 6. In this frame we use the domain markings BO (for business object), BW (for business worker) and EC and EB (for causal/biddable external systems), as already introduced in Sect. 3; and E? for an external system that may be either biddable or causal. Notice, that all the domains in Fig. 6 do not have internal phenomena, but only external phenomena.

Enterprise applications are quite complex, and so the machine part of the related problem frame is not trivial and will be a composite domain, whose structure is shown in Fig. 7. It needs to be composite to take into account the usual three tier/layer architecture of Enterprise Applications. A BO-D is a designed domain with a full model of the business object BO that EA works with. An ES-I domain (similarly a BW-I) corresponds to some limited information about ES (BW) that EA needs to interact with it (e.g., its name and the way to access it). A D-I domain may be empty/null; this is usually the case of a domain D with multiplicity 1.

Fig. 7 shows a preliminary version of the EA frame machine; further aspects have to be considered. For example, we have to take into account the fact that some of the . . . -D and . . . -I domains must be persistent. Assume that D1, . . . , Dn are the persistent domains, thus

Also the way EA interacts with a business worker or an external system may be further detailed, clarifying whether it will be direct or indirect. For example, such interaction may be realized by exchanging paper docu-
ments (e.g., paper mail), thus
becomes

Furthermore, also the domains Presentation, Engine and Database may be further structured. In this paper we do not consider the design phase, and so we do not discuss any further the frame for the Enterprise Application machine.

4.2 Placing the Enterprise Application (EA)

The developer should decide which part of the business will be taken care by EA. Visually, that can be done by enclosing in a box the part of the Business Frame that will be automatized by the EA (both domains and business cases).

The outside domains participant in an enclosed business case will be then interacting with the EA, and will be linked to EA by some shared interfaces. It is also possible to introduce new domains linked with the EA, for example new external systems to cooperate with the EA to run the various business cases, e.g., the email to handle the communications with some business workers.

Thus, now it is possible to draw the problem frame instance for the particular case at hand.

There are some checks to be done on the performed choices to detect possible problems, which may prevent to build the EA in a sound way; for example:

- Any business object participant in an enclosed business case must be enclosed in EA.
- Each enclosed domain must be connected by a chain of business cases having a common participant with an outside domain (otherwise, it is useless and can be dropped).
- There should be at least one domain outside; otherwise EA will be a completely black box, and thus a useless system, with which no one may interact. In general, this fact may be caused either by an incomplete business frame, or because some business worker was misplaced as a business object (if the EA handles clients, but the clients also interact with EA, then there should be two domains in the Business Frame one for the client as a person and another one for its associated information managed by the EA, e.g., ClientRecord or ClientInfo.
- If there are no domains inside, this is a limit case of an EA that just acts as an interface or a wrapper for a bunch of external systems or to support simply interactions among business workers (e.g., a kind of messaging system); in this case one should wander if this is really a case of enterprise application.
- If all the outside domains linked with EA are causal then we have another limit case, that is an EA that always report the same information (not related to any request).

- A biddable domain cannot be inside; indeed being biddable means that it is not possible to fully automatize his/her/its behaviour. In this case, the developer must first check if it is possible to transform it into a causal domain using also time related external phenomena (for example, in this way it is possible to reduce to non causal behaviour to a causal one, which periodically performs some activity), or by introducing more external phenomena in other domains to allow it to react to them. Sometimes, the domain can be factorized in smaller domains where some of them will be causal and other will be biddable, thus the former may be enclosed, whereas the latter will stay outside.
- ...more checks may be defined to help developers detect problems quite early.

4.3 µEC: Placing the Enterprise Application

Notice that Fig. 8 shows that some business cases are left out; those between the Client and the Payment System (they do not concern µEC). Note also, that the Manager cannot be put inside µEC (i.e., automatized) since her/his activity cannot be reduced to a purely reactive one, she/he has to decide when and which products to add and to remove from the catalogue, whereas the price change perhaps may be automatized, for example by linking it to the change of some rate, which can be given by some external system. Then, we can get the the EA problem frame instance for the µEC case (Fig. 9). The asterisk attached to the Client domain denotes that there can be any number of clients.

4.4 Requirement Specification

The Requirement Specification, corresponding to the Requirement part of the EA problem frame, see Fig. 6, is expressed in UML and following the precise method of Astesiano-Reggio [1, 2].
Thus the Requirement Specification is a UML model containing:

- A class diagram with at least a class for each domain in the frame; those biddable are active, whereas causal ones are passive classes. To take into account the BO, BW and E markings we use again the three corresponding stereotypes introduced in Sect. 3 (<<bo>>, <<bw>> and <<es>>). The shared phenomena are modelled by means of operations. Then, this class diagram includes a class for the EA (EA) with operations corresponding to its shared external phenomena. The EA class may have some private attributes, which are used to store information about the business workers and the external systems (but not the business objects), they are already fully described by the corresponding domains.

- A complete definition of the behaviour of all the classes stereotyped by <<bo>>: these are really important since their behaviour is a relevant part of the business logic. Obviously, the other classes behaviour may be modelled, whenever it is not trivial.

- A use case diagram and a description for each of its use cases, where the actors are all and the only domains connected with EA. The description of the behaviour of a use case consists of a statechart associated with the class EA, such that
  - the events are either timed events or call events,
  - the conditions concern only its attributes,
  - and the actions are either its attribute updates or calls of the use case actions operations.

Note that, here, the use cases do not fully specify the requirements, and the description of the various business objects behaviour is a fundamental part. Using the standard terminology, we can say that the business logic is partly included in the definition of the business objects, and partly in the use cases. Thus, we factorized it in two parts: the rules/what to do (in the use cases), and the subjects/who is acted on (the business objects); we think that this should help to master the business logic's complexity, and, since it is reflected in the architecture of the machine to develop (see Sect. 4.1), to help the design procedure.

To give the requirement specification we start from the (UML) Business Model and from the placement diagram.

![Figure 9: µEC problem frame](image)

![Figure 10: µEC Requirement Specification: Class Diagram](image)
cases is derived from those of the corresponding Business Cases. No other use case at the summary level should be introduced, the core functionalities of the enterprise application should be derived from the business; however, to accommodate extra activities due to the entities introduced during the placement phase, new use cases (at the user or subfunction level) may be added.

### 4.5 \( \mu \)EC case: Requirement Specification

Fig. 10 presents the class diagram part of the \( \mu \)EC Requirement Specification. It is defined starting from that of the Business Model in Fig. 3. A new class corresponding to the enterprise application, i.e., \( \mu \)EC, is added, and its operations model abstractly the communications it can receive from the entities in its context (classes of stereotype \( \texttt{bo} \), \( \texttt{bw} \) and \( \texttt{es} \)). The associations in the diagram show the possible communications flow. A new data class ClientRecord is added, with the information about the clients that \( \mu \)EC needs to interact with them (just the passwords to control their access).

Fig. 11 presents the use case diagram; notice that a use case, Register is added; this is quite common, since the introduction of the application in the business may require additional activities, in this case the client must register with the system before buying.

Finally, Fig. 11 shows a description of the PutOrder use case (the others are in Appendix A.3) described from the \( \mu \)EC viewpoint: it reacts to the clients requests to access and modify the business objects (stock, orders and catalogue) and to interact with an external system (Payment System). Contrary to the corresponding business case, the client must login with the system before buying.

### 5 Conclusion, Related and Future Work

In this paper we present a software development approach for Enterprise Applications. The first step is to match the Business Frame and the EA Frame that we propose with the problem at hand, then the modelling of the various parts of the frames are achieved following the proposed UML diagrams. We thus combine the use of the UML notation, the use of the structuring concepts provided by the problem frames, together with our methodological approach for well-founded methods. While the Business Frame and the EA frame provide a first overall structure for the application, our method shows, for each development phase, how to use appropriate UML constructs.

As mentioned in the introduction, we think problem frames are very good at providing a first requirement structure that is invaluable to start the analysis of a problem and understand its nature. Problem frames provide a means to reuse experience that is helpful to start a complex problem analysis with some structuring concepts in mind.

In our opinion, the approach proposed here will help the development of enterprise applications, at least in the first phases, since

- it will reduce the time spent to decide which UML constructs to use and how to model the domain and the requirements;
- by requiring that the modelling of the domain and of the problem is accompanied by their "framing" , it will help the developer to manage such complex things, and after will offer a support to navigate the complex produced UML models.

Here, we have used a simple toy example to present our approach; clearly, now it needs to be validated by more relevant case studies, and this will be our next step.

In this work, we chose to use the notation provided by Jackson for problem frames, and then to pursue the development using the UML notation. We consider that these are useful graphic notations that may be easily used (problem
frame notation is quite simple, and UML is widespread). However, we think that the essence of our approach is in the use and combination of the relevant underlying concepts, and that they could be expressed using different notations as preferred by the user of our approach. For instance, our Business and EA Frames could also be expressed using UML diagrams (we did not chose this option for several reasons, one is to use both the problem frame concepts and notation, the other is to use a different graphical language for a different level of abstraction). Another option is to move to formal specification languages, as we did in [3] to provide CASL and CASL-LTL specifications for some of the basic problem frames, which can be done both for the problem frame level and for the specification of the various parts of a frame.

Concerning the business modelling, RUP, the Rational Unified Process, considers it as an important task to be done before the requirement specification, and offers a specific UML profile, see [9], for doing it using UML. This profile and the associated method are quite different from what we propose in this paper. First of all, there is a difference in the aims, the profile of [9] is intended to model businesses for business analysts and designers, and thus, e.g., they consider business goals and stakeholders. We have more limited aims, that are to retrieve enough information on the business to capture and specify the requirements for a supporting application. Furthermore, our proposal is more minimalist, introducing just a few stereotypes and a few guidelines on how to use the UML constructs. From a more technical viewpoint, in our business frame and associated UML business model we do not precisely fix the boundary of the business, and as consequence we do not have business use cases (modelling the interaction between business actors and the business) nor the distinction between business workers (inside the business) and business actors (outside of the business). It is only when we will have put the application to be developed in the business context, that we will make this distinction. In this way, one of our business frame/model may be reused for many different applications supporting many different aspects of that business.

References


A Appendix

A.1 µEC Business Frame

Browse business case

Deliver business case

Refill business case

A.2 µEC Business Model

Fig. 12 shows the participants of the business case Refill and the fact that it involves a product identity PI, and a quantity Q. The associated activity diagram exhibits its behaviour.

Figure 12: µEC Business Model: Refill Business Case

The participants of the business case Deliver are Client, Orders, and DeliveryDept as in Fig. 13.

The Behaviour Views of the µEC business model passive classes are given below.

Stock
context receiveProd(PI,Q)
pre: Q > 0
post: available(PI) = available@pre(PI) + Q
context takeProd(PI,Q)
pre: Q >= 0
post: if Q <= available(PI) then
(result = True and available(PI) = available@pre(PI)-Q)
else (result = False)

Orders
method addOrder(CI,PI,Q,D)
(O = create(Order); O.client = CI; O.prod = PI;
O.quant = Q; O.date = D;

A.3 µEC Requirement Specification: Use Case Descriptions (Fig. 15)

Figure 15: µEC Requirement Specification: Use Case Descriptions