Requirements Capture and Specification for Enterprise Applications: a UML based attempt - Report

Christine Choppy
LIPN, Université Paris XIII, France

Gianna Reggio
DISI, Università di Genova, Italy

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 entreprise 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.

Enterprise Applications cover a wide range of applications. Our method, using problem frames, leads to choose precise concepts to handle appropriate variants.
Keywords: Domain Modelling, Requirements Specification, Enterprise Applications, Problem Frames, UML based development method

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."

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 [7], architectural styles re useful to provide an overall structure for the system, while design patterns are more appropriate when structuring the design before coding, thus patterns may differ in granularity.

One of the difficult issues is to start the analysis of a complex problem. M. Jackson proposes "Problem Frames" [7] that can be used by themselves or in combination 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 [7] 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 [9, 10] 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* 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 came in several variants, thus it is not possible to propose for them a simple frame made by few elements as the basic ones in [7]. First of all we had to extend the notation used by Jackson to present the frames, by adding, e.g., provision for composite phenomena and domains, see Sect. 2. We also decided, for the above reasons, in this case not to follow the approach that suggests to decompose the problem at hand in smaller subproblems, till the subproblems match the basic frames; we propose instead a frame where to match directly the problem of the development of the application at hand.

In problem frames presented by M. Jackson [6, 7] 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 are not modified (or removed) when the new system is introduced.

In our understanding, an enterprise applications (shortly EA) may introduce some change in the preexisting context, the business, for instance by replacing some entities (think of the case 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 that is managed by the EA. We then propose an Enterprise frame composed of two parts:

– A *Business Frame* describing the business (in large) which the EA will manage. 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 task of designing the EA, we just give a hint, to shown of the structuring power of the (business and EA) frames will guide also the design step, and will help to trace the requirements.

In Sect. 3 we present the Business Frame and the associated Business Model, and in Sect. 4 the EA Frame, how to derive it starting 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, μEC, where clients buy products chosen in a browsable catalogue and pay using an external payment system; the products are produced by a factory, stocked, and then delivered by a dedicated department.

2 Notation for Frame Presentation

Following Jackson's notation [7], different types of domains are used in the diagrams. As shown in Fig. 1, a machine 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) as a box with a single stripe, and a domain (a problem domain whose properties are given) as a box with no stripe.

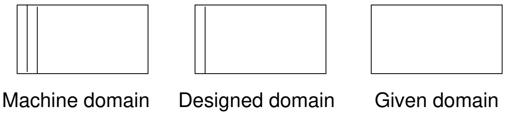


Figure 1: Problem diagram symbols

A solid line connecting two domains is an interface of shared phenomena. In Fig. 2, phenomena `ph1` is controlled by domain `D1` and is shared between domains `D1` and `D2`.

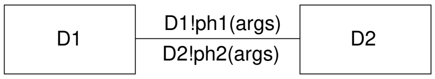


Figure 2: Shared phenomena notation

As shown in Figure 3, requirements are denoted by a dashed oval, and a dashed line connecting a domain and a requirement is a requirement reference, while a dashed arrow is a constraining requirement reference.

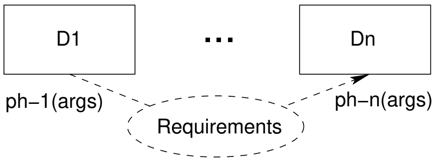


Figure 3: Requirements notation

Problem frames differ in the number of components and the layout, and also in the domain and phenomena types. Using letters in the lower right corner, domains are marked as lexical (data), biddable (people), or causal (see Fig. 4), and the choice for the first letter phenomena name indicates whether it is symbolic, event or causal.

When considering complex systems, we found it useful to provide some extensions to this notation, as shown in Figure 5. When there are several instances of a domain, the usual multiplicity notation (`*` or `n..m`) is a useful graphic abbreviation. When a number of shared phenomena have to be taken into account, they are grouped into a notion of service `S` (a set of phenomena). We also found it useful to have a notion of internal (non sharable) phenomena `i-ph`, and of (external) sharable phenomena `e-ph`.

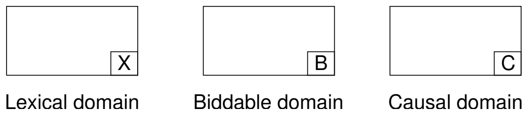


Figure 4: Domains and interface markings

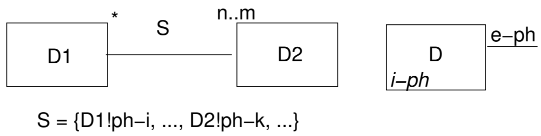


Figure 5: Extensions to the notation

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* (see Fig. 6).

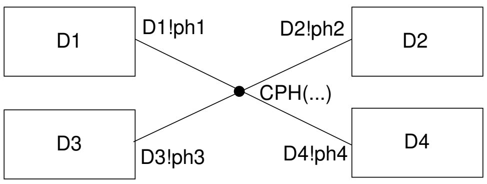


Figure 6: Composite phenomena

3 Enterprise Applications: the Business Frame

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

3.1 Business Frame

Usually, the business subject of an enterprise application is quite complex and needs to be accurately understood and modelled before to start the application development.

We assume that the business consists of various entities interacting among them, to realize the various activities specific of that business. Technically, the business will be schematically structured by means of a Jackson frame, see Sect. 2, having a domain for each entity of the business, and where their mutual interactions are given in terms of composite phenomena.

The domains appearing in a Business Frame must be of kind C or B, those of kind B must have at least one internal phenomena, and those of kind C must have at least one external phenomena and cannot have internal phenomena.

The domains appearing in a Business Frame have to be marked not only with C or B, but also with respect to their role in the business. We distinguish only three categories:
– business objects (marked by O), the entities which are the subject of the business;
– business workers (marked by B), the entities which are doing something inside the business; we do not distinguish further among them (other approaches, e.g., [8], distinguish between those working for the business and the business actors);
– and external systems (marked by E), 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).
Business objects must be of kind C, business workers of kind B, whereas there are no constraints on the kind of the external systems.

The composite phenomena appearing in a Business Frame correspond to the relevant activities made in the business, and will be named *business case*. Notice that we do not use the term "business use case" as in other approaches, e.g., in [8], because, we do not model the business from the point of view of something interacting with it. Instead, for us a business case is seen as a cooperation among business workers, business objects and external systems. We prefer this approach, since it avoids to fix at this points the boundaries of the business under investigation, allowing to get a more abstract description. Moreover, a description of the business produced in this way, can be reused for the development of various different applications, becoming thus a more valuable asset.

3.2 μEC case: Business Frame

Fig. 7 shows the Business Frame for our e-commerce example μEC. This framework exhibits business workers (e.g., Manager), business objects (e.g., Orders), and external systems (e.g., Factory), and complex business cases among them, e.g., Put order is a business case in which Client, Orders, Catalogue and the Stock take part. To avoid to clutter the 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. 8, the Put order business case is described in detail (the others can be found in Appendix A.1).

3.3 (UML) Business Model

After we gave the proper frame for the business of interest, we should describe its component domains and business cases. We chose to do that using UML following the precise method of Astesiano-Reggio [1, 2].

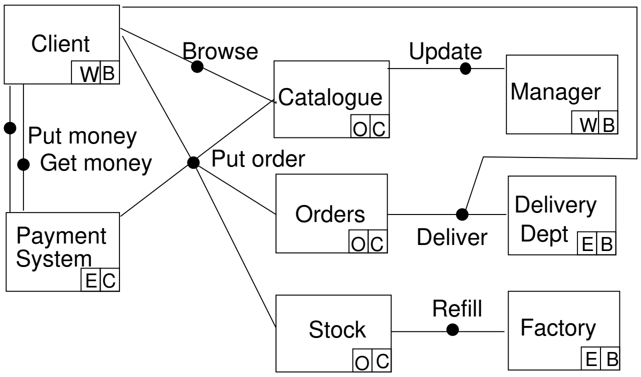


Figure 7: μEC Business Frame

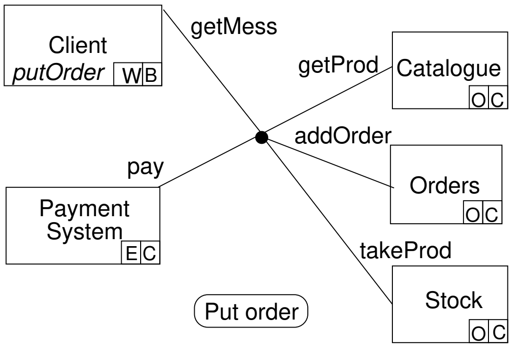


Figure 8: μEC Business Frame: Put order Business Case

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, say BF, the associated Business Model is a UML model consisting of:
– a class diagram with a class for each domain appearing in BF, such class will be active for the B domains and passive for the C ones. We use three stereotypes to indicate for each class which is role w.r.t. the business, precisely ≪bo≫ (for "Business Object"), ≪bw≫ (for "Business Worker) and ≪es≫ (for "External System"), according to the domain markings O, W and E. For each internal phenomenum there will be a private operation in the corresponding class; the happening of the internal phenomena correspond to self-calls of the associated operations. To stress that they correspond to autonomous acts, we use the operation stereotype ≪A≫. The external phenomena, to whom a domain must react, will be modelled by operations of the corresponding class. Obviously, the classes in this diagram may have attributes, 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 for the passive classes, we just model the behaviour of their operations.
– a description of each business case, by means of a UML collaboration summarizing the participants in the case, and possible parameters, and by an activity diagram, whose action-states may only contain calls of the participant operations, and whose conditions must be built by using only the participant operations and attributes.

3.4 μEC Business Model

Fig. 10 presents the class diagram belonging to the μEC Business Model; it contains a class for each domain in the μEC Business Frame of Fig. 7 appropriately stereotyped. On the top of the diagram there are some nonstereotyped classes introducing the data used by the others (e.g., Product and Order).

Catalogue
```
method newProd(PI,E,S)
   {P = create(Product); P.id = PI; P.price = E;
context deleteProd(PI) post:
   not ps.id ->includes(PI)
context changePrice(PI,E) post:
   ps->select(id = PI).price  = E
```

Figure 9: μEC Business Model: Catalogue Behaviour View

Fig. 9 presents the Behaviour View associated with the business object class Catalogue; being a passive class we have just modelled its operations, one by a method and the others by post conditions. The other 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. 11 we present one Business Case, precisely "Put Order" (the others are in Appendix A.1). The activity diagram shows how after the autonomous act of the client of putting 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 describe 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

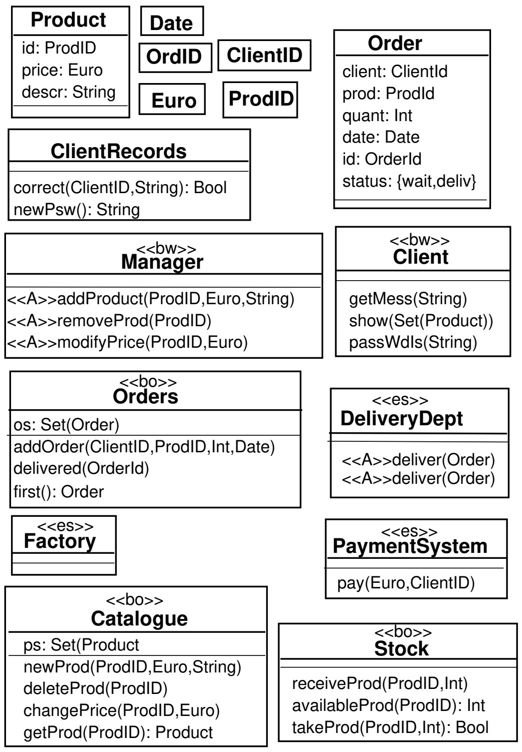


Figure 10: μEC Business Model: Class Diagram

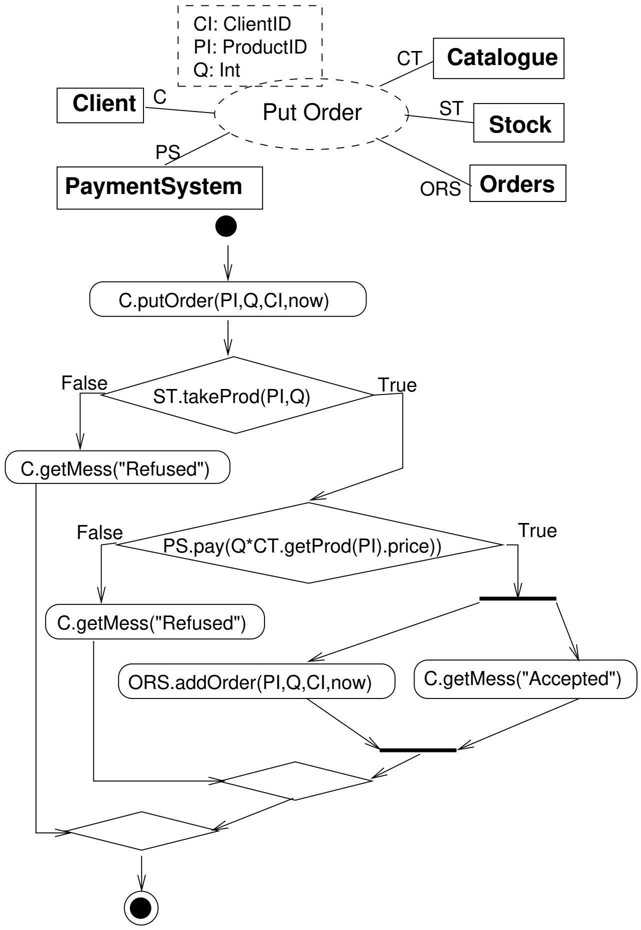


Figure 11: μEC Business Model: Put Order Composite Phenomena/Business Case

build the corresponding Requirement Specification using the UML.

4.1 Problem Frame

We report the problem frame for the Enterprise Application in Fig. 12. In this frame together with the original Jackson's marking of the domains (C and B - to which we add ? when both are possible), we use also O (used for business object BO), W (used for business worker BW) and E (used for external systems ES), as already introduced in Sect. 3.

Notice, that all the domains appearing in Fig. 12 do not have internal phenomena, but only external phenomena shared between them.

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. 13. 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 giving a full model of the business object BO to allow the EA to work with it. An ES-I domain (similarly a BW-I) instead corresponds to some limited information about ES (BW) needed by EA to interact with it (e.g., its name and the way to access it). A

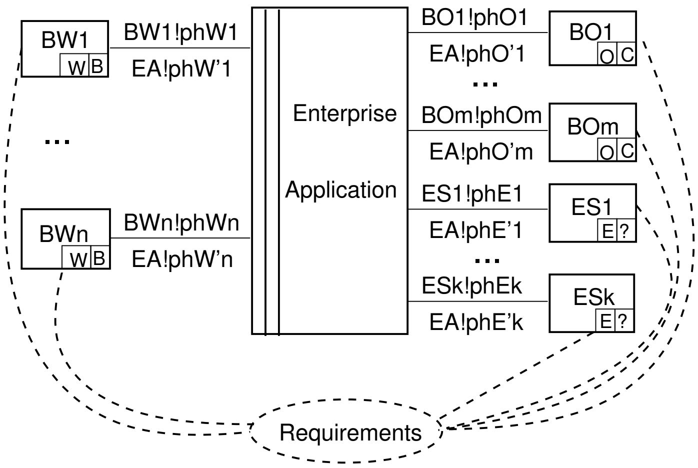


Figure 12: EA Problem Frame

D-I domain may be empty/null; this is usually the case of a domain D with multiplicity 1.

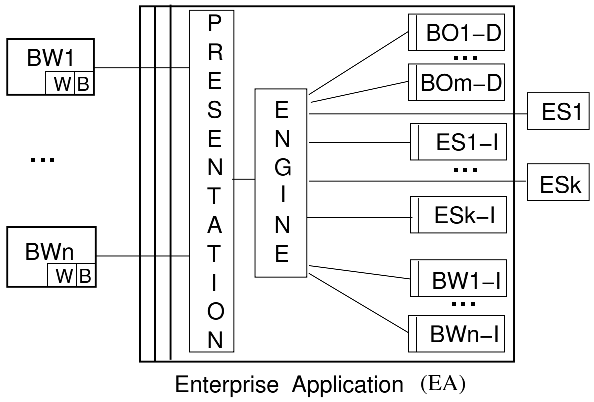
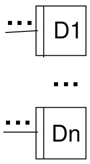
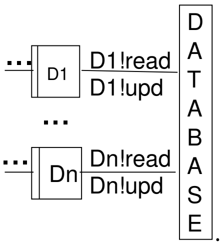


Figure 13: EA Problem Frame: Machine

Fig. 13 shows a preliminary version of the EA frame; 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

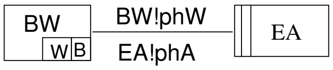


should be replaced by

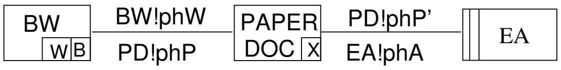


.

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 documents (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 more 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 he EA to run the various business cases, for example, the email to handle the communications with some business workers.

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

There are some checks to be done on the performed choice 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 of kind C, then we have another limit case, that is an EA that always report the same information (not related to any request).
- A domain of kind B cannot be inside; indeed being B 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 domain of kind C 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 C and other will be B, thus the C ones may be enclosed, whereas the B ones 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. 14 shows that some business cases are left out; those between the Client and the Payment System

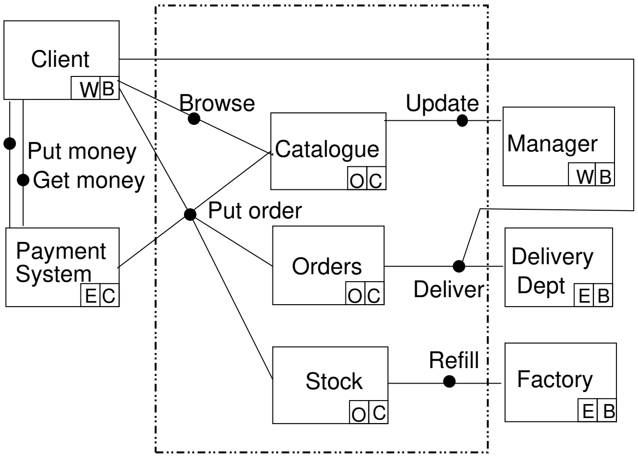


Figure 14: μEC placing

(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 instantiation of the EA problem frame for the case of μEC, reported in Fig. 15. The asterisk attached to the Client domain denotes that there can be any number of clients.

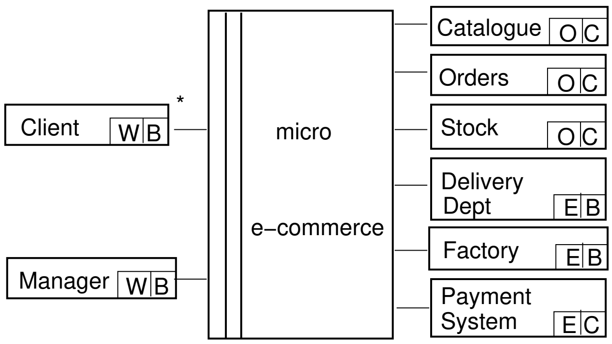


Figure 15: μEC problem frame

4.4 Requirement Specification

The Requirement Specification, corresponding to the Requirement part of the EA problem frame, see Fig. 12, will be given using UML and following the precise method of Astesiano-Reggio [1, 2].

Thus the Requirement Specification will be a UML model containing:

- a class diagram with at least a class for each domain in the frame; those marked by B will be active, whereas those marked by C will be passive classes. For taking into account the O, B and E markings we use again the three corresponding stereotypes introduced in Sect. 3 (≪bo≫, ≪bw≫ and ≪es≫). The shared phenomena will be modelled by means of operations. Then, this class diagram will include a class for the EA, EA, whose operations correspond to its shared external phenomena. The EA class may have some private attributes, which will be used to store information about the business workers and the external systems, but not about the business objects, they are already fully described by the corresponding domains.

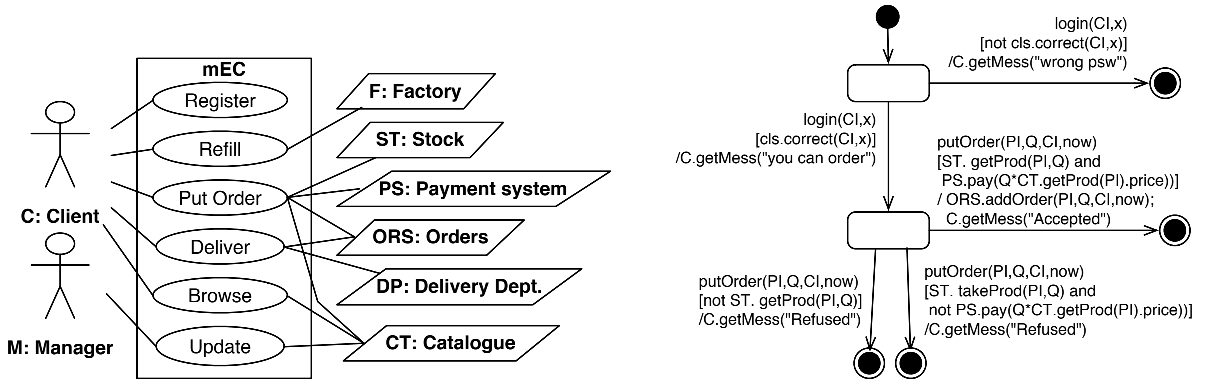
- a complete definition of the behaviour of all the classes stereotyped by ≪bo≫; these are really important since their behaviour is a relevant part of the business logic. Obviously, also the behaviour of the other classes may be modelled, whenever it is not trivial.

- a use case diagram and a description of each use case in it, where the actors are all and only the 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 updates of its attributes or call of operations of the use case actors.

Note that here the use cases do not fully specify the requirements, also the description of the behaviour of the various business object 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 have 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 master the complexity of the business logics, and, since it will be 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.

The class diagram part of the Business Model is the starting point of for the class diagram part of the Requirement Specification. The classes corresponding to domain neither included nor connected with the EA will be thrown away, and a class corresponding to the EA has to be introduced. The interfaces of the class connected with EA may need to be modified to allow them to interact with the EA; this mainly concerns the class of stereotype ≪bw≫, where



Use Case Diagram — Use Case Put Order Description

Figure 17: μEC Requirement Specification

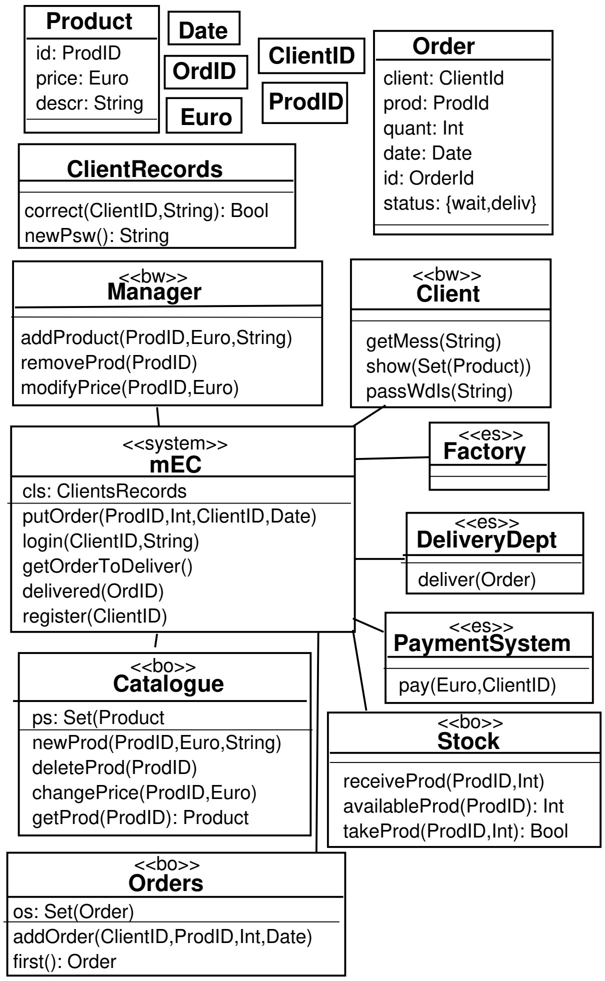


Figure 16: μEC Requirement Specification: Class Diagram

changes in those of stereotype ≪bo≫ or ≪es≫ should be carefully motivated, and their feasibility investigated.

The behaviour of the business objects (classes of stereotype ≪bo≫) should be entirely recovered from the Business Model; obviously, also those of the other classes, if it is not trivial.

For each business case enclosed in the EA, there should be a use case (usually a summary one following the classification of [11], recall enterprise applications are usually quite large and complex). The description of these use cases will be 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 the extra activities due to the interacting with the EA new use cases, at the user or subfunction level may be added.

4.5 μEC case: Requirement Specification

Fig. 16 presents the class diagram part of the Requirement Specification of μEC. It has been defined starting from that of the Business Model in Fig. 10. A new class corresponding to the enterprise application, i.e., μEC, has been added, and its operations model abstractly the communications it can receive by the entities in its context (classes of stereotype ≪bo≫, ≪bw≫ and ≪es≫). The associations appearing in the diagram show the possible flow of the communications. A new data class ClientRecord has been added, these are the information about the clients that μEC needs to interact with them (just the passwords to control their access).

Fig. 17 instead presents the use case diagram; notice that a use case, Register, not corresponding to a business case has been added; this is quite typically, since the introduction of the application in the business may require additional activities, in this case the client must register with the system before to be able to buy.

Finally, Fig. 17 shows a description of a use case, PutOrder (the others are in Appendix A.3); here we can see putting an order from the μEC point of view: it reacts to the requests from the clients accessing and modifying the business objects (stock, orders and catalogue) and by interacting with an external system (Payment System). Differently from the corresponding business case, the client before to buy must login with the system.

5 Conclusion and future work

In this paper we have presented a software development approach for Enterprise Applications. The first step is to match the Business Frame and the EA Frame that we propose, then the descriptions 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 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 used easily (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 [8], for doing it using UML. This profile and the associated method are quite different from what we have proposed in this paper. First of all, there is a difference in the aims, the profile of [8] is intended to modell businesses for business analysts and designers, and thus, e.g., they consider, e.g., business goals and stake holders. 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 point of view, 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). Only when we will have put the application to be developed in the business context, we will make this distinction. In this way, one of our business frame/model may be reused for many different application supporting many different aspects of that business.

References

[1] E. Astesiano and G. Reggio. Towards a Well-Founded UML-based Development Method. In A. Cerone and P. Lindsay, editors, *Proc. of SEFM '03*, pages 102–117. IEEE Computer Society, Los Alamitos, CA, 2003.

[2] E. Astesiano and G. Reggio. Tight Structuring for Precise UML-based Requirement Specifications. In M. Wirsing, A. Knapp, and S. Balsamo, editors, *Radical Innovations of Software and Systems Engineering in the Future, Proc. 9th Monterey Software Engineering Workshop, Venice, Italy, Sep. 2002.*, number 2941 in Lecture Notes in Computer Science. Springer Verlag, Berlin, 2004.

[3] C. Choppy and G. Reggio. Using CASL to Specify the Requirements and the Design: A Problem Specific Approach. In D. Bert and C. Choppy, editors, *Recent Trends in Algebraic Development Techniques, Selected Papers of the 14th International Workshop WADT'99*, number 1827 in Lecture Notes in Computer Science, pages 106–125. Springer Verlag, Berlin, 2000.

[4] Christine Choppy and Gianna Reggio. A UML-Based Approach for Problem Frame Oriented Software Development. *Information and Software Technology*, 2005. accepted for publication.

[5] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison Wesley, 2003.

[6] M. Jackson. *Software Requirements & Specifications: a Lexicon of Practice, Principles and Prejudices*. Addison-Wesley, 1995.

[7] M. Jackson. *Problem Frames: Analyzing and Structuring Software Development Problems*. Addison-Wesley, 2001.

[8] S. Johnston. Rational UML Profile for business modeling. Available at `www-128.ibm.com/developerworks/rational/library/5167.html`, 2004.

[9] OMG. *UML Specification 1.3*, 2000. Available at `http://www.omg.org/docs/formal/00-03-01.pdf`.

[10] OMG. UML 2.0 Superstructure, 2003. `http://www.omg.org/cgi-bin/doc?ptc/2003-08-02`.

[11] S. Sendall and A.Strohmeier. Requirements Analysis with Use Cases. `http://lglwww.epfl.ch/research/use_cases/RE-A2-theory.pdf`, 2001.

A Appendix

A.1 Business Frame

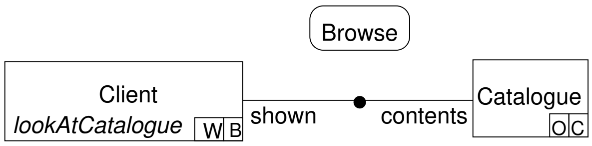


Figure 18: μEC Business Frame: Browse Business Case

A.2 μEC Business Model

The composite phenomena Refill is shared between Stock and Factory, and it involves a product identity PI, and a quantity Q as in Figure 21. The associated business case exhibits the activities involved.

The composite phenomena Deliver is shared between Client, Orders, and DeliverDepartment as in Figure 22.

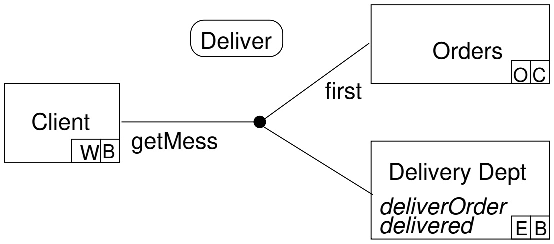


Figure 19: μEC Business Frame: Deliver Business Case

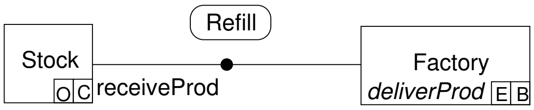


Figure 20: μEC Business Frame: Refill Business Case

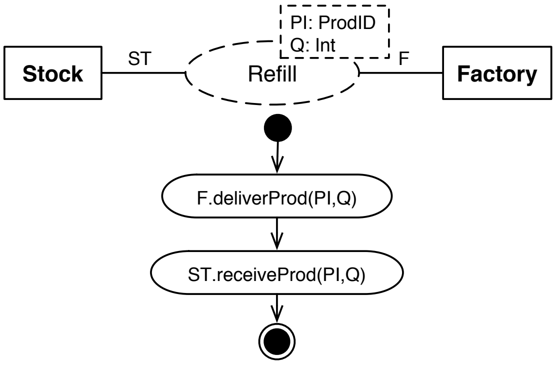


Figure 21: μEC Business Model: Refill Business Case

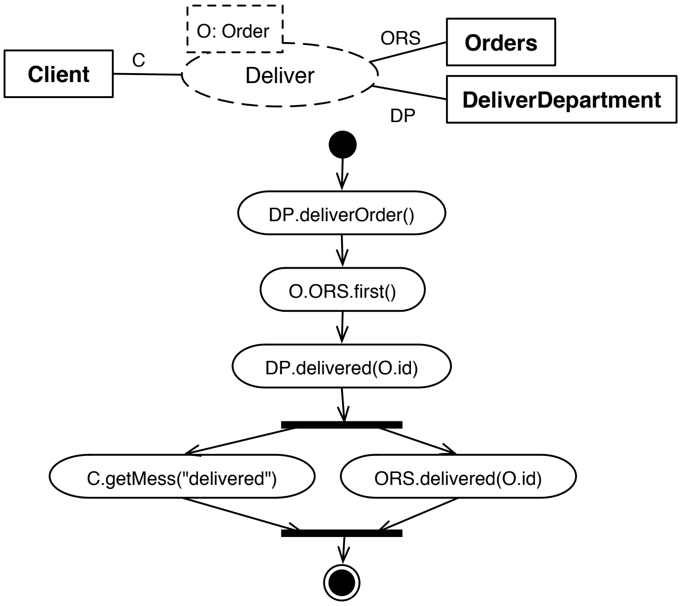


Figure 22: μEC Business Model: Deliver Business Case

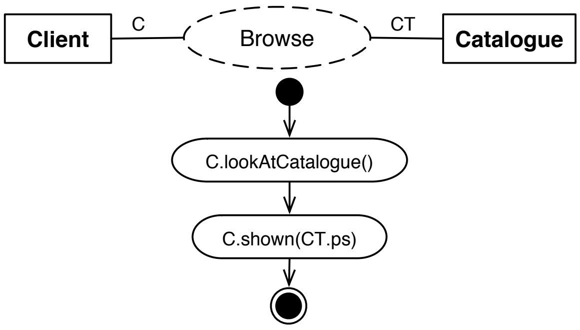


Figure 23: μEC Business Model: Browse Business Case

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
      result  = False
```

Orders

```
method addOrder(CI,PI,Q,D)
   {O = create(Order); O.client = CI; O.prod = PI;
   O.quant = Q; O.date  = D;
   O.id  = X s.t. os.id->excludes(X); os  = os U {O}}
context delivered(OI) post:
   os->select(id = OI).status  = delivered
context first() post:
   os->select(status = waiting)->forall(O.date <= result.date)
```

Figure 24: μEC Business Model: Behaviour Views

A.3 μEC Requirement Specification: Use Case Descriptions

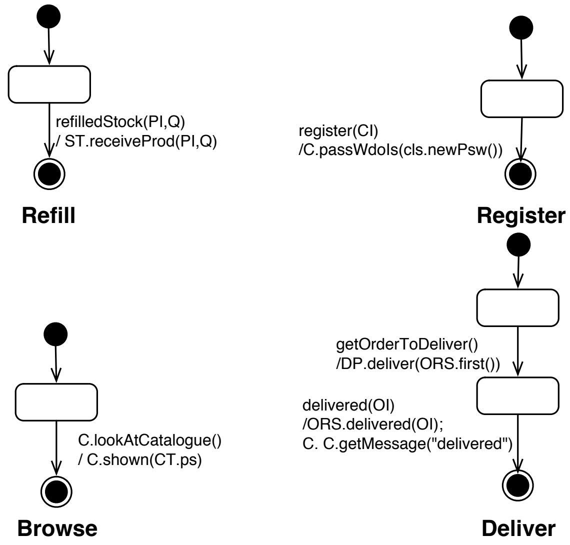


Figure 25: μEC Requirement Specification: Use Case Descriptions