Requirements Capture and Specification for Enterprise Applications: an MDA compliant attempt (Complete Version)

C. Choppy[1] and G. Reggio[2]

[1] LIPN, Institut Galilée - Université Paris XIII, France
[2] 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 the Michael Jackson's problem frames, and the use of the UML notation, within an MDA approach. 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 development task with the problem frame that we propose for entreprise applications, and this helps to understand the nature of the problem under study. For each problem frame, a diagram is settled, showing the involved domains, the requirements, the design, and their interfaces.
Being within an overall MDA approach, each part of the problem frame will be described by a (UML) model of the appropriate kind (computation independent, platform independent, and platform dependent, depending on the development phase). We then provide guidelines to develop all the models 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 kind of models, and of 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, resulting in two computation independent models.
Keywords: Domain Modelling, Requirements Specification, Enterprise Applications, Computation Independent Models, Problem Frames

1 Introduction

MDA is an approach to system development, which increases the power of models in that work. It is model-driven because it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification. The Model-Driven Architecture prescribes certain kinds of models to be used, how those models may be prepared and the relationships of the different kinds of models.

We address here the issue of producing, understanding and defining all the required models for the development of an enterprise application.

Since patterns are "ready-to-use" structures drawn from experience, it is now quite widespread to use them to help systems development. Among the various kinds of available patterns, problem frames propose an overall problem structure [9].

One of the difficult issues is to start the analysis of a complex problem. M. Jackson proposes "Problem Frames" [9] 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. The domains may be *lexical* (this refers to data or files), *causal* (which means that they can change only as a result of some external event), or *biddable* (this refers to people or other entities capable of autonomous behaviour). Five basic problem frames are provided [9] 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.

Since the UML notation [11, 12] carries valuable concepts experienced in practice, we proposed in [5] associated UML based methods for the various basic frames.

These methods require to produce different identified models (described by their skeletons) and guide the developer from one to the others, thus we think that this approach can be compliant with the MDA proposal [1].

The frames offer a conceptual schema that can be seen as a conceptual road map of the produced models and as a device to help produce the required models, while offering an effective way to develop the required models, and to establish their relationships.

Here we consider the specific topic of developing enterprise applications.

Enterprise applications are complex systems so their development requires appropriate concepts. M. Fowler [7] 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.*"

In the work presented here, we propose a new problem frame, the *Enterprise Application Frame* that we devised for enterprise applications, together with its associated development method based on the UML (cf. Figure 1). In order to keep the notations as consistent as possible, we chose to use the UML as the visual notation for presenting the diagram associated with this and the other frames introduced in the paper.

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 [9]. We thus propose here a composite frame (Business Frame, EA Frame and Machine Frame, cf. Fig. 1) so as to address the different main parts of the enterprise applications.

In problem frames presented by M. Jackson [8, 9] 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 that is managed by the EA. Furthermore, the machine part of the frame is quite complex, and its structure cannot be conveniently described in the frame diagram, thus we give it apart in another frame diagram. We then propose an *Enterprise Application Frame* composed of three parts:

– A *Business Frame* that is a schematic structure of the business that the EA will manage. This Business Frame helps to precisely understand the business considered together with its business rules. Having a separate business frame also promotes its reuse in many different applications.

– A "classical" problem frame, the *EA Frame*, with the EA machine, its domains and require-

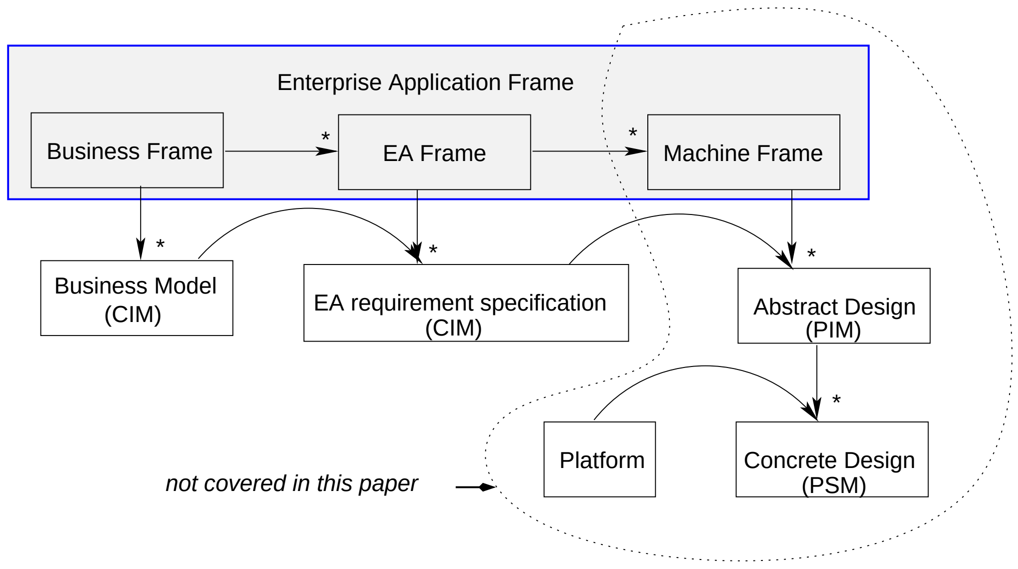


Fig. 1. The method for enterprise applications: artifacts and transformations

ments.
– A *Machine Frame* that is a schematic structure of the EA machine.

The EA Frame will be derived from the associated Business Frame, and similarly the Machine Frame from the corresponding EA Frame.

Then, we present how from the Business Frame we can derive the corresponding UML model, the Business Model, and from the EA Frame plus the Business Model, the Requirement Specification, again a UML model. Following the MDA terminology (see [1]), both models are computational independent CIM models. For lack of room in this paper we consider only this initial steps of the development, which is summarized in Fig. 1, where the various frame diagrams and models to be produced, and the relative transformations are depicted.

In this paper, we do not consider the task of designing the enterprise application, we just give a hint, to show the structuring power of the (Business and EA) frames.

In Sect. 2 we present the Business Frame and how to produce the associated Business Model, in Sect. 3 the EA Frame, how to derive it starting from the Business Frame, and in Sect. 4 how to derive the associated UML Requirement Specification. Due to lack of space, the Machine Frame description is not included here and may be found in [6].

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 Enterprise Application Frame: the Business Frame

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

2.1 Business Frame

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

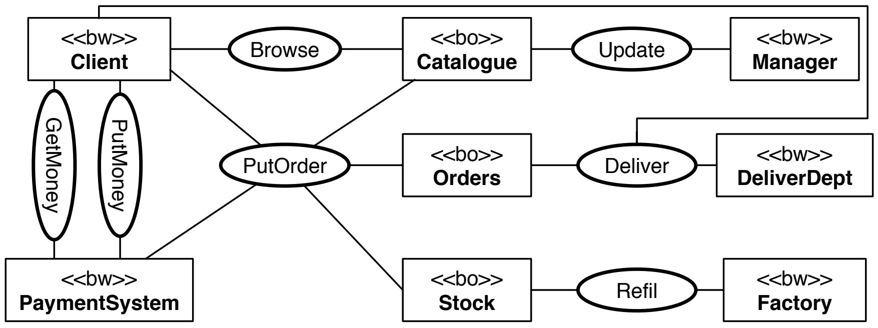


Fig. 2. μEC Business Frame Diagram

We assume that the business consists of various entities interacting among them (called domains in Jackson's terminology) to realize the various activities specific of that business. Technically, the schematic structure of the business will be presented using UML by means of a class diagram (e.g., the one in Fig. 2), where

- all the classes have no attributes nor operations and must be stereotyped by one of the following
  - ≪bo≫ business objects, the entities which are the subject of the business; following Jackson's [9] terminology they must be causal;
  - ≪bw≫ business workers, the entities which are doing something inside the business; we do not distinguish further among them (other approaches, e.g., [10], distinguish between those working for the business and the business actors); again following Jackson's terminology they must be biddable;
  - ≪BCase≫ business cases, they correspond to the relevant activities made in the business. Notice that we do not use the term "business use case" as in other approaches, e.g., in [10], because, we do not model the business from the point of view of some actor interacting with it. Instead, for us a business case is a cooperation among business workers and business objects. 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, thus becoming a more valuable reuasable asset. A class stereotyped by ≪BCase≫ will be visually presented as an oval.
- all the associations are binary, they connect a ≪BCase≫ class (business case) with a ≪bo≫ or ≪bw≫ class, and represent the fact that some entities take part in some activities.

2.2 μEC Case Study: Business Frame

Fig. 2 shows the Business Frame diagram for our e-commerce example μEC. This diagram exhibits business workers (e.g., Manager), business objects (e.g., Orders), and complex business cases among them;e.g., PutOrder is a business case in which Client, Orders, Catalogue and the Stock take part.

2.3 Business Model

After we gave the proper frame diagram for the business of interest, we should describe its component entities and business cases. We chose to do that using UML following the precise method of Astesiano-Reggio [2, 3]. 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 diagram, say BF, the associated Business Model is a UML model consisting of:
– a class diagram having at least a class for each class stereotyped by ≪bo≫ (for "Business Object") or ≪bw≫ (for "Business Worker) appearing in BF, stereotyped again in the same way. We model the autonomous acts of the business workers by self calls of special private operations that we sterotype using ≪A≫. The interaction among the entities part of the business will be modelled by operations of the classes. Obviously, the classes in this diagram may have attributes, other operations and may be defined using other classes (e.g., datatypes).
– defintions of 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 in BF, by means of a UML collaboration summarizing the participants in the case, the 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.

2.4 μEC Case Study: Business Model

Fig. 3 presents the class diagram belonging to the μEC Business Model; it contains a class for each business worker and object in the μEC Business Frame diagram of Fig. 2 appropriately stereotyped. On the top of the diagram there are some classes introducing the data used by the others (e.g., Product and Order).

Fig. 4 presents the behaviour of 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 behaviour of the other passive classes are in Appendix A.1. We do not report the behaviour of the other (active) classes, since we do not have information on the way they behave.

In Fig. 5 we present one business case, precisely PutOrder (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 when it is refused, no reason is given.

3 Enterprise Application Frame: The EA Frame

In this section, we first present the EA Frame, i.e., the core problem frame related to the development of an enterprise application, and how it can be derived by placing EA on the underlying Business Frame.

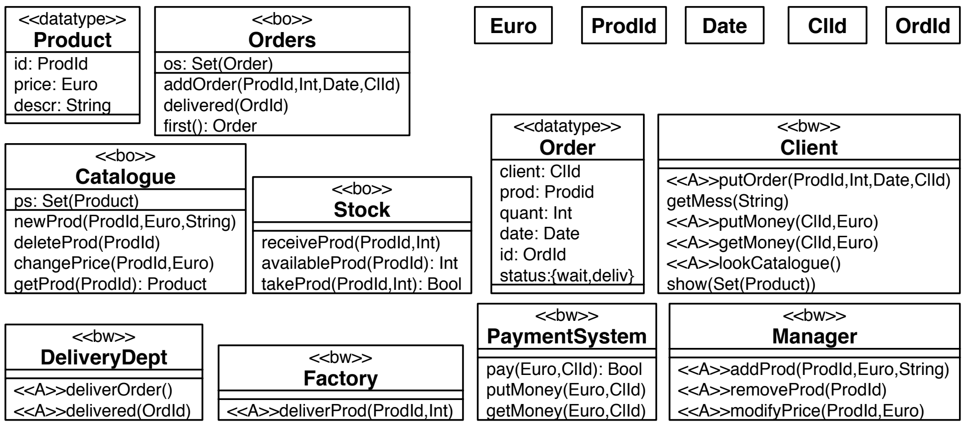


Fig. 3. μEC Business Model: Class Diagram

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

Fig. 4. μEC Business Model: Catalogue Class Behaviour

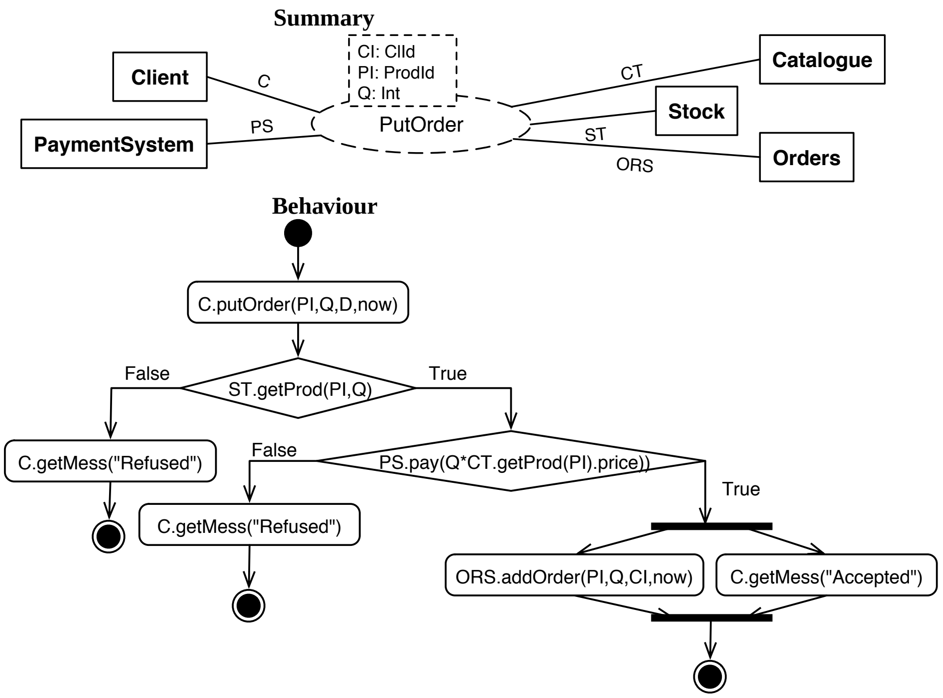


Fig. 5. μEC Business Model: PutOrder Business Case Description

3.1 The EA Frame

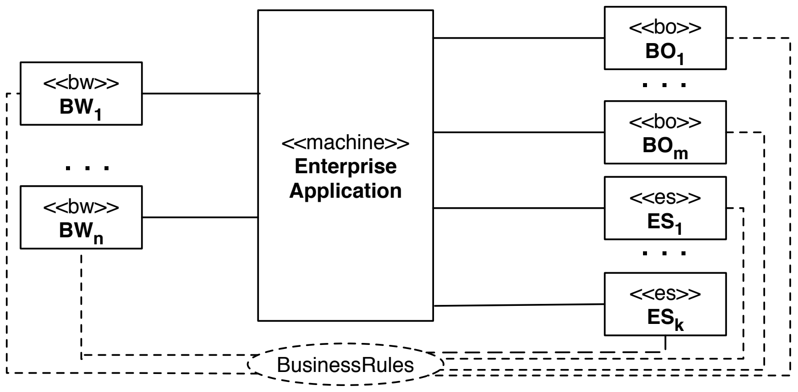


Fig. 6. EA Frame

We report the problem frame diagram for the enterprise applications in Fig. 6 using UML as notation, instead of the original visual notation proposed by M. Jackson in [9]. The EA Frame diagram is a class diagram, where classes are stereotyped by either

– ≪bo≫, ≪bw≫ (already introduced in Sect. 2), they are respectively the business workers that will use the enterprise application and the business objects subject of the enterprise application activities;

– ≪es≫ external systems, i.e., entities external to the enterprise application used for outsourcing some activities (e.g., messages handling, mail) or for taking advantage of data or functionalities provided by already available systems (e.g., a system giving information about the credit history of people or handling credit card payments).

All these entities are called "domains" by Jackson [9], as domains with which the system is working.

– ≪requirement≫ (visually presented by a dashed oval, as in Jackson's frames). In this case, the requirements essentially correspond to the rules governing the business under consideration that the EA should automatize.

– ≪machine≫ represents (using Jackson's terminology) the system to be developed.

The associations allowed on such class diagram are

– binary association (visually presented by a dashed line) between the requirements and entity class.

– or binary associations between the machine and an entity class.

3.2 Placing the Enterprise Application

To develop an enterprise application EA the first step is to place it in the business under consideration, i.e., to decide which part of that business will be automatized by EA. Visually, this can be done by enclosing within a dotted box the part of the Business Frame that will be automatized by the EA (both entities and business cases).

The entities that are not enclosed within the business case and that interact with the EA, will be linked to it by a binary association. Moreover, if they are business objects, then they will be changed into external systems (stereotyped by ≪es≫). Similarly, all the enclosed business objects will be linked to the EA by a binary association, but they will keep their ≪bo≫ stereotype. It is important to distinguish between the external systems and the business objects, since the former will not be under the responsibility of the developer, but instead they exist and should just be used by the EA, whereas the latter are part of the real world, but have to be automatized/digitalized by the developer to be acted upon by the EA. It is also possible to introduce new external systems to cooperate with the EA to run the various business cases, for example, the email system for communications with some business workers.

Thus, now it is possible to draw the EA Frame diagram for the particular case at 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:

• Each enclosed entity must be connected by a chain of business cases having a common participant with an outside entity (otherwise, it is useless and can be dropped).

• There should be at least one entity 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 entities 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 entities 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 entities linked with EA are *causal*, (using the Jackson's terminology), then we have another limit case, that is an EA that always report the same information (not related to any request).

• A *biddable* entity (using the Jackson's terminology) cannot be inside; indeed 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 entity using time related external phenomena (for example to reduce the non causal behaviour to a causal one, which periodically performs some activity), or introducing interaction means in other entities. Sometimes, the entity can be factorized in smaller entities some of them being causal and others being biddable,

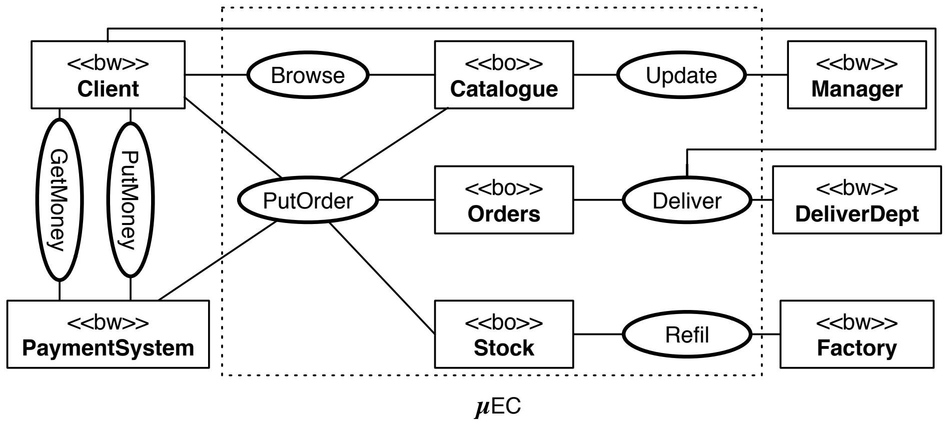


Fig. 7. μEC placing

thus the causal ones may be enclosed, whereas the biddable ones will not.
• . . . more checks may be defined to help developers detect problems quite early.

3.3 μEC Case Study: the EA Frame

Fig. 7 presents the application placing in the μEC case. Notice that it shows that some business cases are left out; those between the Client and the PaymentSystem (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, (s)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 Frame for the case of μEC, reported in Fig. 8. The multiplicity of the association attached to the class Client denotes that there can be any number of clients.

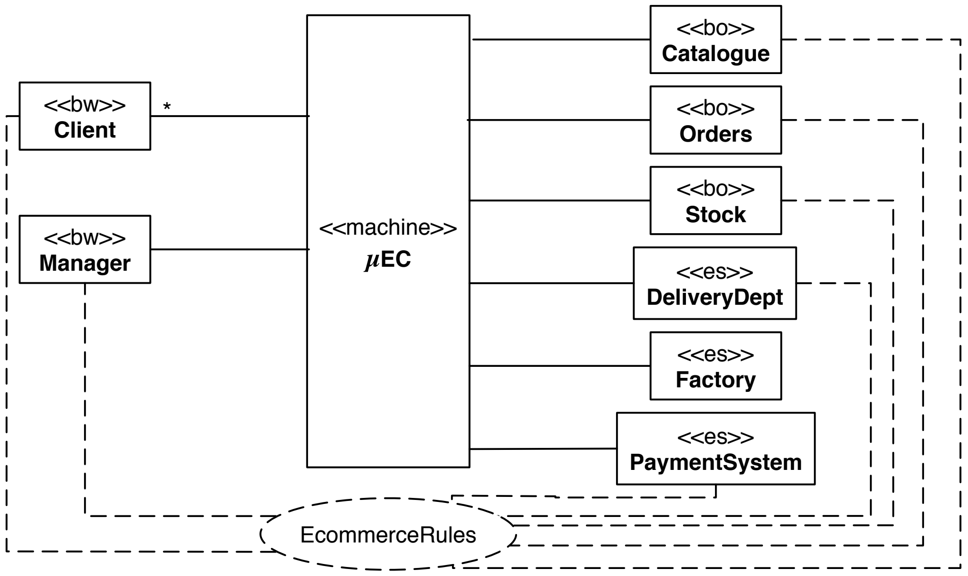


Fig. 8. μEC EA Frame

4 Enterprise Application Requirement Specification

In this section, we present the requirement specification of an enterprise application, and how it can be derived from the EA Frame and the Business Model; then we give the Requirement Specification for the μEC case.

4.1 Requirement Specification

The Requirement Specification, corresponds to the Requirement part of the EA Frame, see Fig. 6, and is given using UML and following the precise method of Astesiano-Reggio [2, 3]. Thus the Requirement Specification is a UML model containing:

– a class diagram with at least a class for each domain in the frame; such classes are stereotyped using ≪bo≫, ≪bw≫ and ≪es≫, already introduced in Sect. 2. Then, this class diagram will include a class for the EA (called EA, and stereotyped by ≪machine≫). 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 classes.
– a complete definition of the behaviour of all the classes stereotyped by ≪bo≫; these are really important since they are 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 all the actors are the only the entities connected with EA. The description of a use case behaviour consists of a statechart associated with the class EA, such that
 (i) the events are either timed events or call events,
 (ii) the conditions concern only its attributes,
 and (iii) 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, so the description of the behaviour of the various business objects 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 [6]), to help the design procedure.

To give the Requirement Specification for an enterprise application, say EA, we start from its Business Model and its EA Frame diagram.

The class diagram part of the Business Model is the starting point for defining the class diagram part of the Requirement Specification. The classes connected with the enterprise ap-

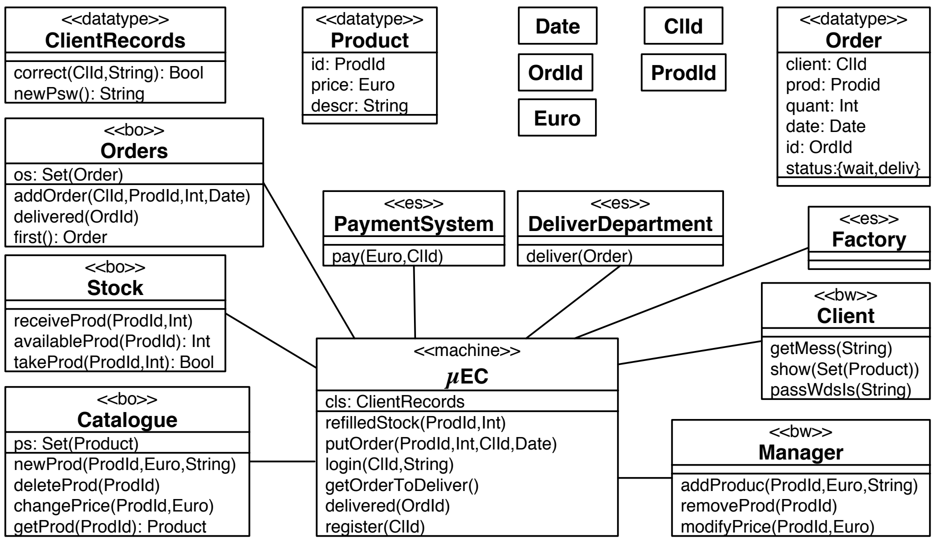


Fig. 9. μEC Requirement Specification: Class Diagram

plication in the EA Frame diagram will be kept, and their stereotypes changed accordingly to what presented by the EA Frame; new classes stereotyped by ≪es≫ will be added for all the external system not already considered in the Business Model (Business Frame), and a class stereotyped by ≪machine≫ corresponding to the EA will 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 classes of stereotype ≪bw≫, where changes in those of stereotype ≪bo≫ or ≪es≫ should be carefully motivated, and their feasibility investigated.

The behaviour of the business objects (classes stereotyped by ≪bo≫) should be entirely recovered from the Business Model; obviously, this applies as well to those of the other classes (when not trivial).

For each business case enclosed in the EA, there should be a use case (usually a summary one following the classification of [13], since 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.

4.2 μEC Case Study: Requirement Specification

Fig. 9 presents the class diagram part of the Requirement Specification of μEC. It has been defined starting from that of the Business Model in Fig. 3. 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. 10 presents the use case diagram; notice that a use case, Register, that is not associated with a business case, is added; this is quite usual 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 a use case, PutOrder (the others are in Appendix A.2); 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)

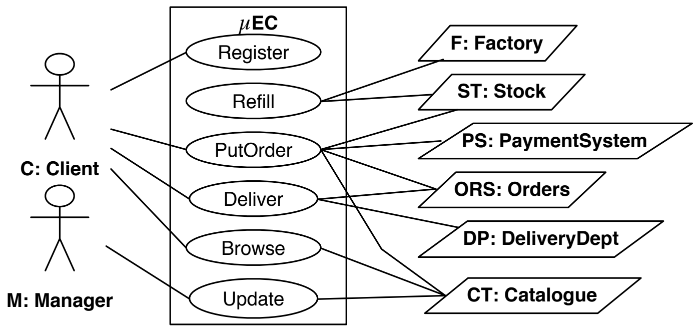


Fig. 10. μEC Requirement Specification: Use Case Diagram

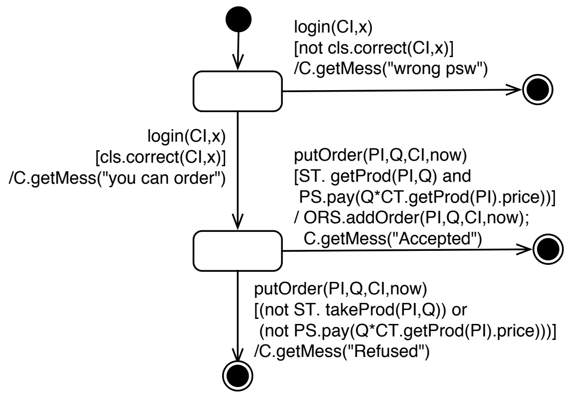


Fig. 11. μEC Requirement Specification: description of the use case PutOrder

and by interacting with an external system (Payment System). Contrary to the corresponding business case, the client must login with the system before buying.

5 Abstract Design Phase

5.1 The Machine Frame

Enterprise applications are quite complex, so the machine part of the related problem frame is not trivial and we have decided to present it by a special frame, the Machine Frame. Moreover, s ince it is not possible to structure the EA in a unique way, this frame will help to determine and present its structure/architecture.

Technically, the Machine Frame is a UML class diagram, where the classes are stereotyped by

– ≪des≫ representing a designed domain (using the Jackson's terminology),
– ≪machine≫ representing submachines to be designed,
– ≪bo≫, ≪bw≫ and ≪es≫ (introduced before) representing the context of the EA.

In Fig. 12 we show a simple version of the Machine Frame following the usual three tier/ layer architecture of the enterprise applications. The submachine Presentation handles the interactions with the business workers, whereas Engine is in charge of executing the various business procedures. The designed domains BO_i-D (i=1,…, m) give the EA a full model of the business object BO_i to work with.

Notice that the frame shows that any BO_i-D is related with the corresponding given domain BO_i (in Jackson's terminology, i.e., part of the real world). An ES-I domain (resp. a BW-I) corresponds to some limited information about ES (resp. BW) necessary for EA interaction (e.g., its name and the way to access it). A D-I domain may be empty/null.

We could have alternatively pursued an approach suggested by Jackson to decompose the EA Frame into smaller subframes (e.g., Presentation and Engine may be considered as the machine parts of other smaller problem frames). We prefer to propose guidelines and patterns to assist the developer to produce the proper structuring of the EA, taking advantage of the peculiarity of this kind of application, and of the existing best practices, as presented by [7].

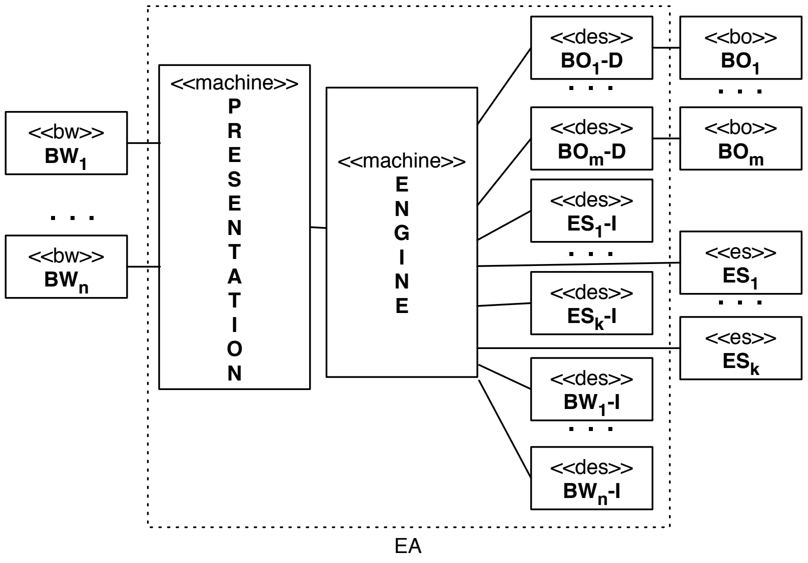
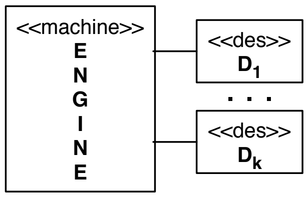


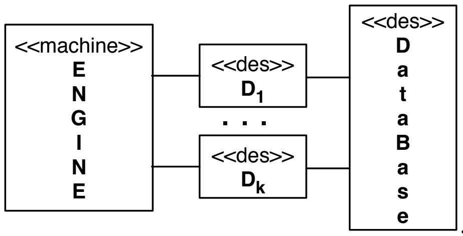
Fig. 12. Enterprise Application Frame: Machine Frame

Fig. 12 shows a preliminary version of the EA frame; it can be refined and restructured considering various aspects. Here we list some patterns that may help to obtain the appropriate machine frame in the various cases.

Introducing Databases Assume that some of the designed domains $\mathsf{D}_1, \ldots, \mathsf{D}_k$ have to be persistent and that they should be stored in a database, say DataBase. Thus the frame of Fig. 12 should be refined by replacing



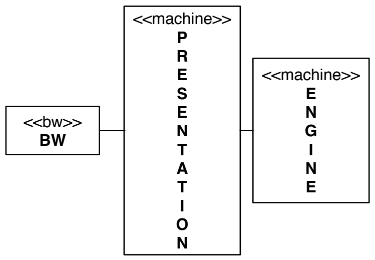
with



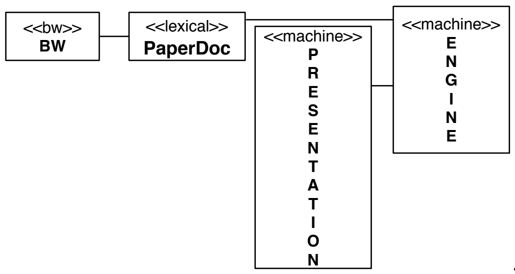
.

Mediated Interaction with EA The way EA interacts with a business worker or an external system may be further detailed, clarifying whether it will be direct or mediated either by some device or by an auxiliary entity.

For example, such interaction may be realized by exchanging paper documents (e.g., paper mail), thus



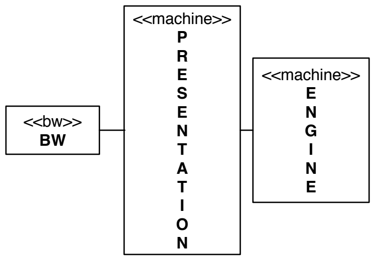
becomes



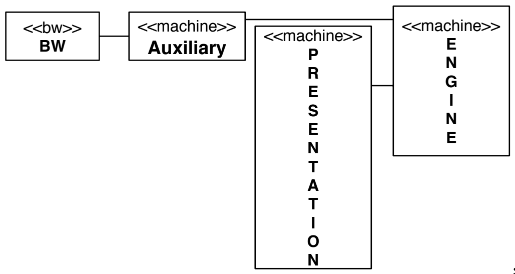
.

The (class) stereotype ≪lexical≫ denotes, using Jackson's terminology, a lexical domain; and, in this case, it describes the documents that will be produced by the EA, to be then given to the business worker.

If, instead, the interaction is realized by some device, e.g., a teller machine, then

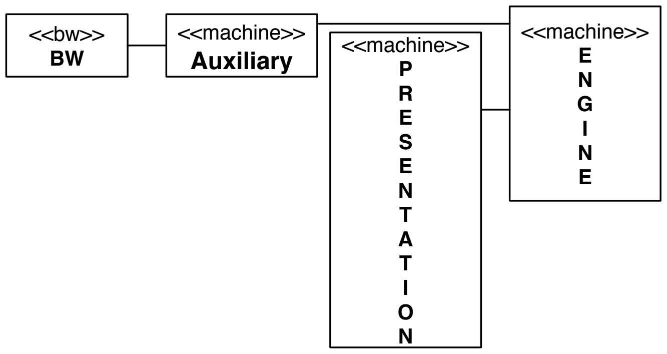


becomes



,

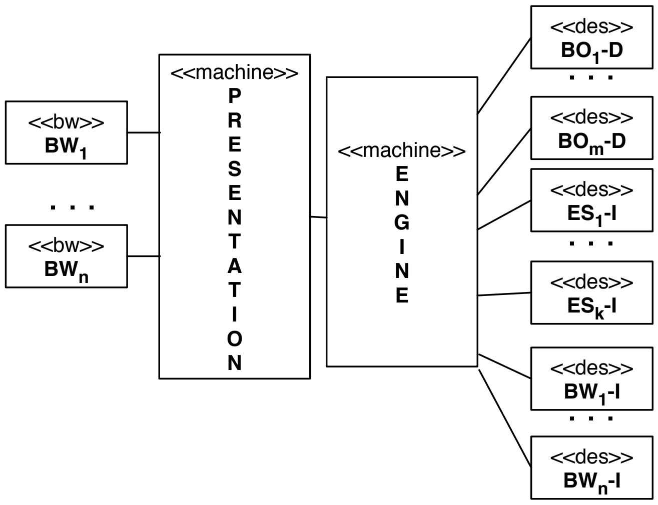
if the mediator device has to be developed, or



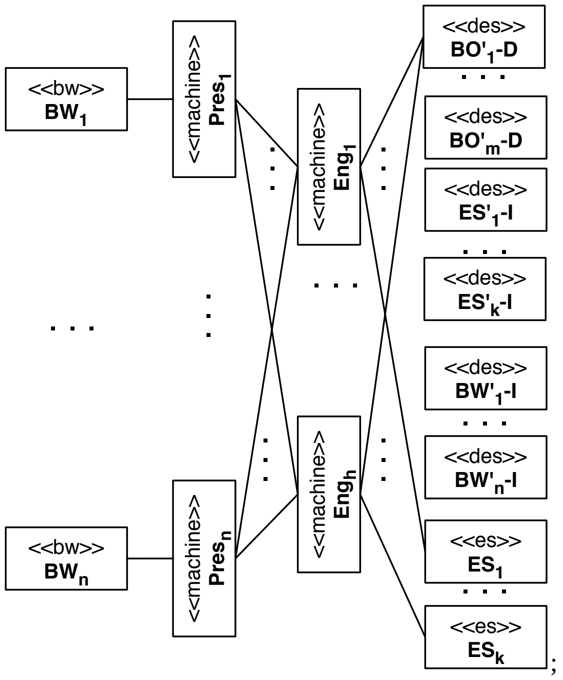
if the mediator device is already existing and will be used, as an external system, by EA.

Refining the thiers/layers The domains Presentation, Engine and Database should be further structured, but not a standard unique way can be given; instead, it is possible to propose many different architectural patterns together with guidelines to choose the most adequate ones and to apply in the various particular cases. In this paper we just briefly present some of the most basic ones.

A standard way to decompose the Engine is to have a dedicated engine for each use case, and any business worker kind has its own specific presentation. In this case



becomes



where the lines connecting the Pres_i and the Eng_j with the other domains are determined the actors of the various use cases (the engine of a use case is connected to the presentation of all its business worker and business object actors).

5.2 μEC Case Study: Machine Frame

In Fig. 13 we present a possible Machine Frame for the μEC case study. We have decided to have two different presentation machines, for the two kinds of business workers (clients and managers), and a different engine for handling each use case, except that we put together the two use cases of a registered client, to browse the catalogue and to put an order. All business objects are persistent data which will be recorded in a unique database, which has to be designed by the developer, indeed it is a designed domain. We have also introduced a new designed domain, Clients-I, for keeping the information about the clients, which will be made persistent by using the same database.

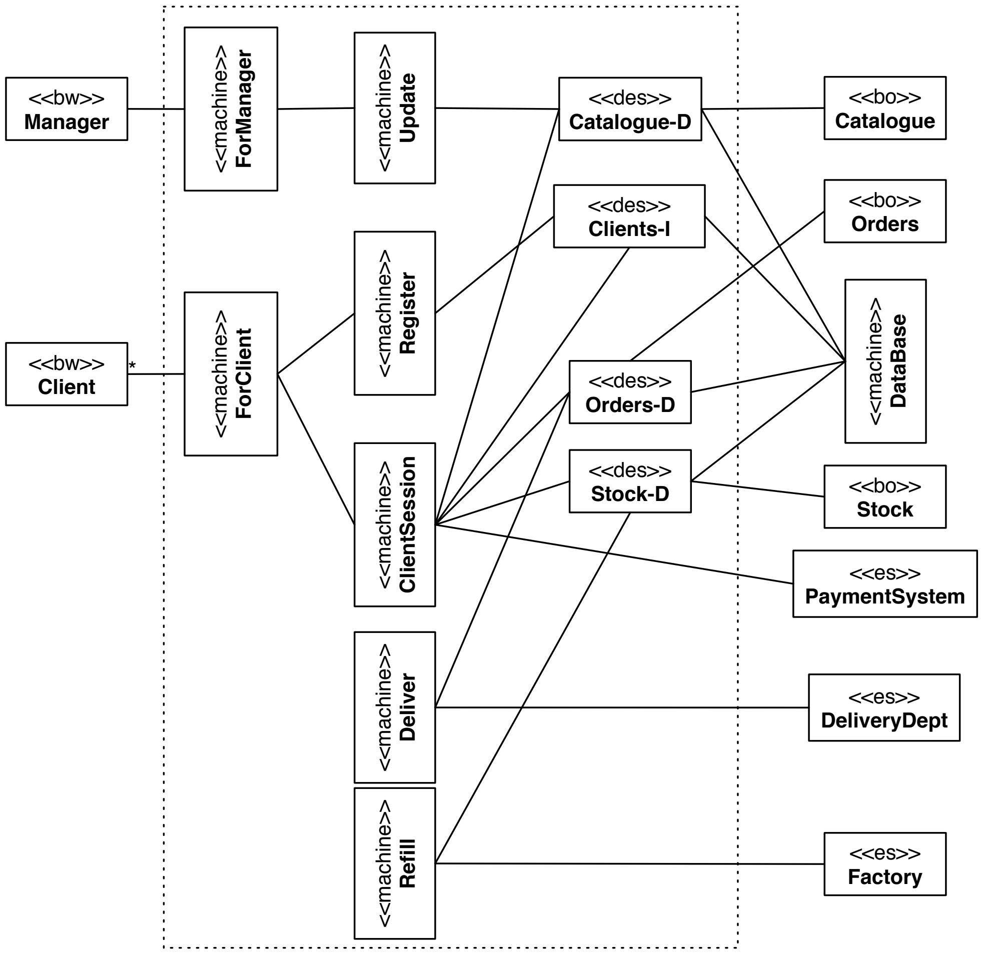


Fig. 13. μEC Requirement Specification: Class Diagram

6 Conclusion and future work

In this paper we 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 models. We thus combine the use of the UML notation, the use of the structuring concepts provided by the problem frames, in an overall MDA approach.

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 produce the corresponding UML models.

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 UML notation to express the frames. As a matter of fact, 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. Thus, our Business and EA Frames could also be expressed using a problem frame notation. We chose to adopt here a uniform UML notation, both for the frames and for the associated UML models.

Another option is to move to formal specification languages, as we did in [4] 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.

Business modelling in RUP (The Rational Unified Process) is an important task to be done before the requirement specification, and it offers a specific UML profile, see [10], 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 [10] is intended to modell businesses for business analysts and designers, and thus, e.g., they consider 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). We will make this distinction only when we would put the application to be developed in the business context. 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. MDA Guide Version 1.0.1. Available at `http://cgi.omg.org/docs/ormsc/03-06-01.pdf`, 2003.
2. 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. Available at `ftp://ftp.disi.unige.it/ person/ReggioG/AstesianoReggio03g.pdf` and `ftp://ftp.disi.unige.it/ person/ReggioG/AstesianoReggio03g.ps`.
3. 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. Available at `ftp://ftp.disi.unige.it/person/ ReggioG/AstesianoReggio03f.pdf`.
4. 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*

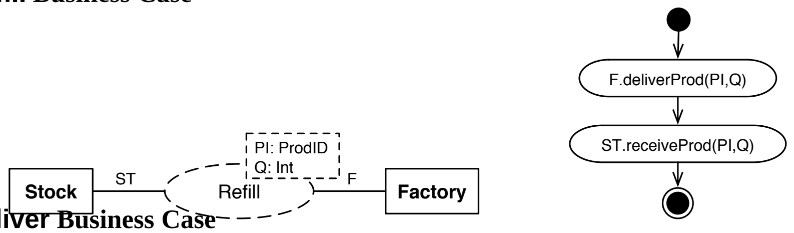
Techniques, Selected Papers of the 14th International Workshop WADT'99, number 1827 in Lecture Notes in Computer Science, pages 106–125. Springer Verlag, Berlin, 2000. A complete version is available at `ftp://ftp.disi.unige.it/ person/ReggioG/ChoppyReggio99a.ps`.

5. Christine Choppy and Gianna Reggio. A UML-Based Approach for Problem Frame Oriented Software Development. *Information and Software Technology*, 2005. accepted for publication.
6. Christine Choppy and Gianna Reggio. Requirements Capture and Specification for Enterprise Applications: an MDA compliant attempt (Complete Version). Technical report, LIPN and DISI, 2005.
7. Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison Wesley, 2003.
8. M. Jackson. *Software Requirements & Specifications: a Lexicon of Practice, Principles and Prejudices*. Addison-Wesley, 1995.
9. M. Jackson. *Problem Frames: Analyzing and Structuring Software Development Problems*. Addison-Wesley, 2001.
10. S. Johnston. Rational UML Profile for business modeling. Available at `www-128.ibm.com/developerworks/rational/ library/5167.html`, 2004.
11. OMG. *UML Specification 1.3*, 2000. Available at `http://www.omg.org/docs/formal/00-03-01.pdf`.
12. OMG. UML 2.0 Superstructure, 2003. `http://www.omg.org/cgi-bin/ doc?ptc/2003-08-02`.
13. S. Sendall and A.Strohmeier. Requirements Analysis with Use Cases. `http://lglwww.epfl.ch/research/use_cases/RE-A2-theory.pdf`, 2001.

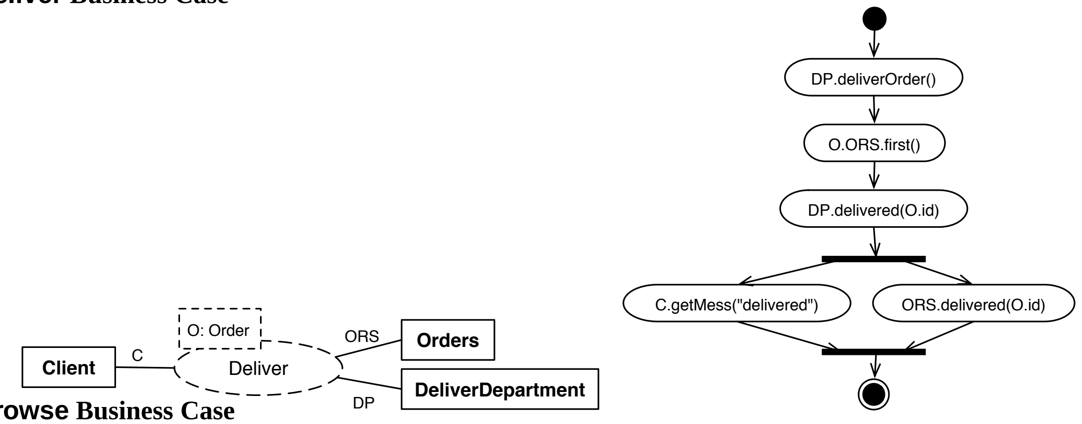
A μEC Case Study

A.1 μEC Business Model

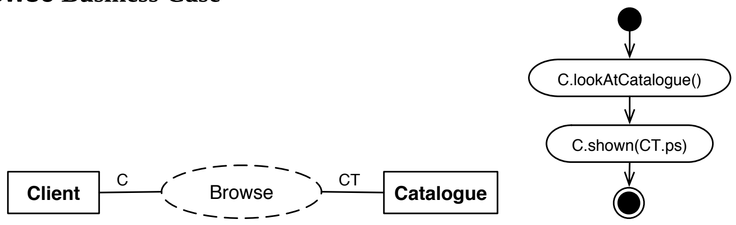
Refill Business Case



Deliver Business Case



Browse Business Case



Stock Class Behaviour

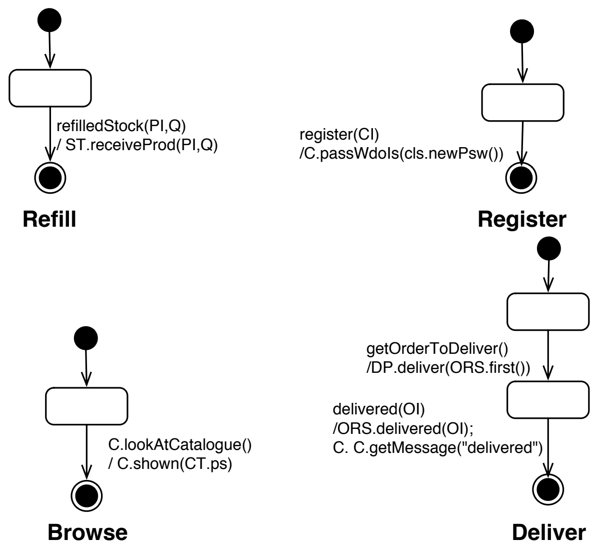
```
context receiveProd(PI,Q)
  pre:  Q > 0
  post:  available(PI) = available@pre(PI) + Q
context takeProd(PI,Q)
  pre:  Q >= 0
  post:  if Q <= available(PI) then
    result = True and available(PI) = available@pre(PI)-Q
    result  = False
```

Orders Class Behaviour

```
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)
```

A.2 μEC Requirement Specification: Use Case Descriptions



B Reference Cards for Artifacts and Transformations Required by the Method

Business Frame It is a UML class diagram, where

– all the classes have no attributes nor operations and must be stereotyped by either
  - ≪bo≫ (business object) or
  - ≪bw≫ (business worker) or
  - ≪BCase≫ (business case, visually presented as an oval).

– all the associations are binary, they connect a ≪BCase≫ class with a ≪bo≫ or ≪bw≫ class.

Business Model It is a UML model consisting of

– a class diagram whose classes may be stereotyped by ≪bo≫ (business object) or ≪bw≫ (business worker). Private operations of active classes may be stereotyped by ≪A≫, to denote that they represent autonomous acts.

– definitions of the behaviour of the classes introduced in the class diagram. The behaviour of an active class is given by a statechart, and the passive classes behaviour only consists in their operations behaviour described either by pre-post conditions or by associated methods.

– descriptions of business cases, consisting of a UML collaboration summarizing the participants in the case and the possible parameters, and an activity diagram, whose action-states may only contain calls of the participant operations, and whose conditions must be built using only the participant operations and attributes.

EA Frame Diagram It is a UML class diagram, where

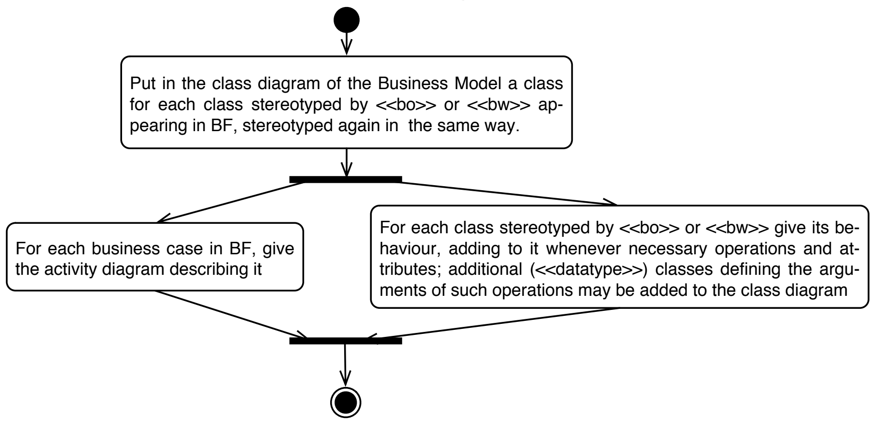
- classes are stereotyped by either
  - ≪bo≫, ≪bw≫, ≪es≫ (respectively, business workers, business objects and external systems) or
  - ≪requirement≫ (visually presented by a dashed oval) or
  - ≪machine≫ (the application to be developed);

  there should be exactly one class stereotyped by ≪requirement≫ and exactly one by ≪machine≫;
- the allowed associations are
  - binary association (visually presented by a dashed line) between the requirements and entity class.
  - or binary associations between the machine and an entity class.

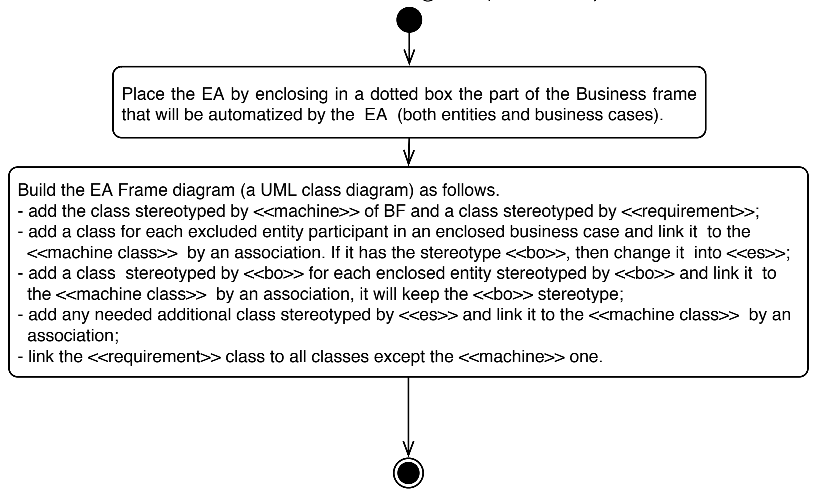
EA Requirement Specification It is a UML model consisting of:

- a class diagram, where classes may be stereotyped by ≪bo≫, ≪bw≫, ≪es≫ and ≪machine≫ (exactly one with this ≪machine≫ stereotype, called EA); and all the associations are binary and connect the class EA with those stereotyped by ≪bo≫, ≪bw≫, ≪es≫;
- descriptions of the behaviour of the classes not stereotyped by ≪machine≫, using statecharts (if active) and pre-post constraints or methods to define their operations (if passive); at least the behaviour of all the classes stereotyped by ≪bo≫ must be defined;
- a Use Case Diagram, where the actors are typed useing the classes with stereotypes ≪bo≫, ≪bw≫, ≪es≫ (any of them must type at least one actor);
- use case descriptions (one for each use case appearing in the use case diagram), are statecharts 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 calls of operations of the use case actors.

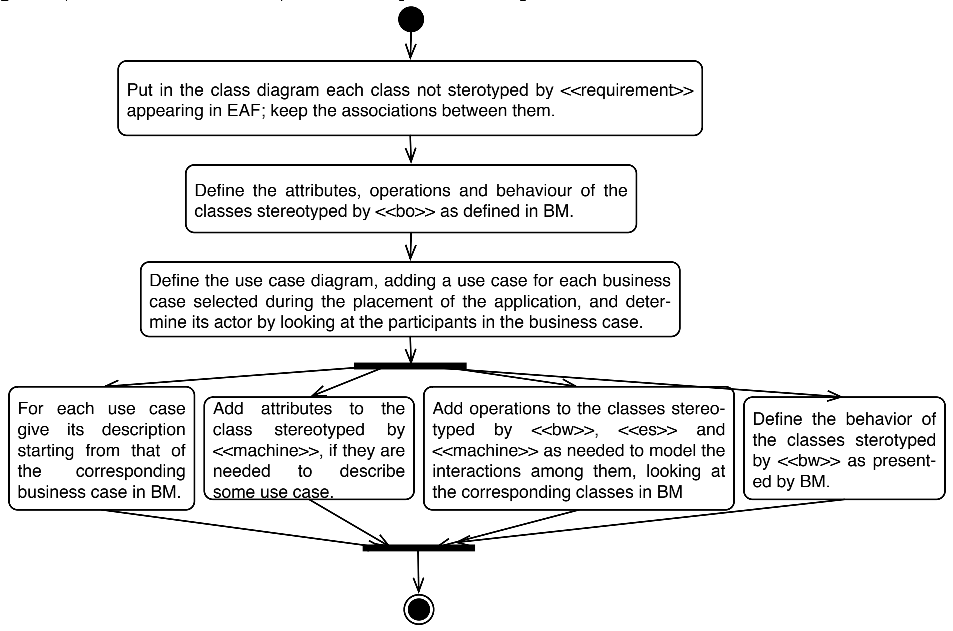
Business Frame ⇒ Business Model The UML activity diagram below shows the guidelines for the transformation of a Business Frame diagram (called BF) into a Business Model.


Put in the class diagram of the Business Model a class for each class stereotyped by <<bo>> or <<bw>> appearing in BF, stereotyped again in the same way.
For each business case in BF, give the activity diagram describing it
For each class stereotyped by <<bo>> or <<bw>> give its behaviour, adding to it whenever necessary operations and attributes; additional (<<datatype>>) classes defining the arguments of such operations may be added to the class diagram

Business Frame ⇒ EA Frame The UML activity diagram below shows the guidelines for the transformation of a Business Frame diagram (called BF) into an EA Frame.


Place the EA by enclosing in a dotted box the part of the Business frame that will be automatized by the EA (both entities and business cases).
Build the EA Frame diagram (a UML class diagram) as follows.
- add the class stereotyped by <<machine>> of BF and a class stereotyped by <<requirement>>;
- add a class for each excluded entity participant in an enclosed business case and link it to the <<machine class>> by an association. If it has the stereotype <<bo>>, then change it into <<es>>;
- add a class stereotyped by <<bo>> for each enclosed entity stereotyped by <<bo>> and link it to the <<machine class>> by an association, it will keep the <<bo>> stereotype;
- add any needed additional class stereotyped by <<es>> and link it to the <<machine class>> by an association;
- link the <<requirement>> class to all classes except the <<machine>> one.

EA Frame and Business Model ⇒ EA Requirement Specification The UML activity diagram below shows the guidelines for the transformation of a Business Model and an EA Frame diagram (called BM and EAF) into a Requirement Specification.


Put in the class diagram each class not sterotyped by <<requirement>> appearing in EAF; keep the associations between them.
Define the attributes, operations and behaviour of the classes stereotyped by <<bo>> as defined in BM.
Define the use case diagram, adding a use case for each business case selected during the placement of the application, and determine its actor by looking at the participants in the business case.
For each use case give its description starting from that of the corresponding business case in BM.
Add attributes to the class stereotyped by <<machine>>, if they are needed to describe some use case.
Add operations to the classes stereotyped by <<bw>>, <<es>> and <<machine>> as needed to model the interactions among them, looking at the corresponding classes in BM
Define the behavior of the classes sterotyped by <<bw>> as presented by BM.