Architecture Specific Models: Software Design on Abstract Platforms*
(the P2P Case)

E. Astesiano – M. Cerioli – G. Reggio

DISI - Università di Genova (Italy)
email: {reggio,cerioli,astes}@disi.unige.it

Abstract. We address in general the problem of providing a methodological and notational support for the development at the design level of applications based on the use of a middleware. In order to keep the engineering support at the appropriate level of abstraction, we formulate our proposal within the frame of Model Driven Architecture (MDA).
We advocate the introduction of an intermediate abstraction level (between PIM and the PSM), called ASM for Architecture Specific Model, which is particularly suited to abstract away the basic common architectural features of different platforms. In particular, we consider the middlewares supporting a peer-to-peer architecture, because of the growing interest in mobile applications with nomadic users and the presence of many proposals of peer-to-peer middlewares.

Introduction

There are three sources of inspiration and motivation for the work presented in this paper.
Engineering Support for Middleware based Software. The use of a middleware, encapsulating the implementation of low-level details and providing appropriate abstractions for handling them in a transparent way, improves and simplifies the development and maintenance of applications (see, e.g., [3]). But, the direct use of middleware products at the programming level without an appropriate software engineering support for the other development phases is dangerous. Hence, as it is argued in [6], the software engineering support, at the design level at least, has to take middleware explicitly into account, providing middleware-oriented design notations, methods and tools. The main aim of this paper is to show how well-established software engineering design techniques can be tailored to provide support for middleware-oriented design.
Platform Independent versus Platform Specific Modeling. Relying on the features of a particular platform too early during the design phase results in rigid models that cannot be reused for further implementations based on different

* Partially supported by Murst - Programma di Ricerca Scientifica di Rilevante Interesse Nazionale Sahara.

technologies. The so called MDA (for *Model Driven Architecture*), a technique proposed by the OMG, advocates the initial use of a *Platform Independent Model* (PIM) to be refined into one or more *Platform Specific Models* (PSM) that are its specializations taking into account the features of the particular technology adopted. The PSM will be adapted or even completely replaced accordingly to the frequent changes in the technology. However, in our opinion, the gap between PIM and PSM can be too wide. Indeed, there are clearly architectural choices, like the level and paradigm of distribution, that are a step down toward the implementation, having fixed some of the details, but are still quite far away from a specific platform, because they can be realized by several different middlewares. Therefore, we advocate the introduction of an intermediate level, called ASM for *Architecture Specific Model.* Devising an ASM implies to define a basic abstract paradigm for the chosen architecture providing conceptual support for the features of the different middlewares for that kind of architecture. Since our proposal is made in the context of the MDA and due to the relevance and success of development methodologies based on object-orientation in general and in particular supported by the UML notation, technically the ASM level is presented by a UML-*profile*. While many profiles have been proposed, few are middleware-oriented (the best known of them is the one for CORBA) and they are all supporting the PSM level, by providing a notation for the features of one particular middleware. We do not know of any UML-profile supporting the development of software based on P2P middlewares, and this provides further motivation to our work.

Decentralization and Mobility. The use of mobile devices (mobile computing) supporting applications for nomadic users is becoming popular. That kind of application needs an appropriate underlying architecture. As it is easily understood and explicitly advocated by many researchers (see, e.g., [2]), particular advantages seem to be provided by the so-called peer-to-peer (shortly P2P) architecture, namely a network of autonomous computing entities, called *peers*, with equivalent capabilities and responsibilities. Recently many proposals have been put forward for P2P architectures, by researchers and companies, mainly under the form of P2P middlewares (see, e.g., [11, 16, 7, 9, 8, 15]). We have been particularly stimulated by the proposal of PeerWare [4], that has been analyzed and used as a paradigmatic example within the project SALADIN (`http://saladin.dm.univaq.it/`). Here, we present what can be considered an *abstraction of most current proposals, that does not pretend to be the definitive choice, also because the P2P paradigm is very recent and our main aim is methodological in advocating the MDA/ASM approach.*

Our notation supports the design of an application by a set of diagrams describing the system architecture, that is the peers and how they are grouped to share resources. Such peers and groups are typed and each such type is described by a diagram. In the case of a peer type, the diagram describes the activity and the resources used and provided by instances of that type, whereas for a group type it just describes the resources shared in such group. A most notable feature of our approach is that the middleware is naturally integrated into the

object-oriented paradigm, by describing it as an object, one for each peer, whose operations correspond to its primitives. The same pattern can be specialized to provide profiles for the PSM level, by specializing the classes representing the middleware objects and the resources and adding/removing features. As a consequence of the introduction of the ASM level, also the mapping of a PIM to a PSM is factorized in two steps, from PIM to ASM and from ASM to PIM.

In this paper, in Sect. 1, after presenting the basic ideas of MDA, we introduce a running example to illustrate the basic concepts and techniques. This is first done outlining the related PIM. In Sect. 2 we introduce our abstract P2P architecture paradigm showing informally how the example application can be mapped onto a P2P architecture. Then, in Sect. 3 we illustrate our profile by showing its structure and its use in defining an ASM for the example application. Finally, in the last section we draw some conclusions, discuss the relationships to existent work, and give some hints on future directions of research.

1 Introducing and illustrating MDA and PIM

1.1 Model Driven Architecture (MDA)

The Model Driven Architecture (MDA) proposed by OMG, see [10,14], defines an approach to system specification that separates the specification of system functionality from the specification of its implementation on a specific technology platform. Any MDA development project starts with the creation of a *Platform Independent Model* (*PIM*), expressed in UML. The PIM expresses only business functionality and behavior undistorted, as much as possible, by technology. Because of its abstraction, it retains its full value over the years, requiring change only when business conditions mandate. Then, one or more *Platform-Specific Models* (*PSM*) are given, implementing the PIM. Ideally, the PSM are derived from the PIM by a mapping that converts the run-time characteristics and configuration information that we designed in the application model in a general way into the specific forms required by the target platform. Guided by standard mappings, automated tools may perform as much of these conversions as possible. To simplify the definition of the mapping, the PSM should be expressed in UML, possibly enriched by linguistic tools to represent the platform ingredients. This is usually done by a UML profile (extension/variant of the basic notation).

1.2 A Worked PIM Example

In this paper we will use as running example a P2P development of an application for handling the orders of a manufacture company: ORDS. Such company stores the products in a number of warehouses, each one serving some locations denoted by their zips, handles automatically the orders, and, to send the invoices to the clients, uses special mail centers. The orders are collected by salesmen, who may also verify the status of old orders. The case study we present here is a simplified toy version, with a minimal amount of features to illustrate our points.

Our abstract design of the ORDS system (modeled by a PIM, named ORDS-PIM), presented in this section, has been made following a simple method developed by our group. Such method assumes that the designed system is built, besides by ≪datatype≫ representing pure values, by objects of three kinds of classes and provides stereotypes to denote them:

– ≪boundary≫ taking care of the system interaction with the external world,
– ≪executor≫ performing the core activities of the system,
– ≪entity≫ storing persistent data (e.g., a database).

The first two stereotypes specialize active classes, while ≪entity≫ specializes passive classes.

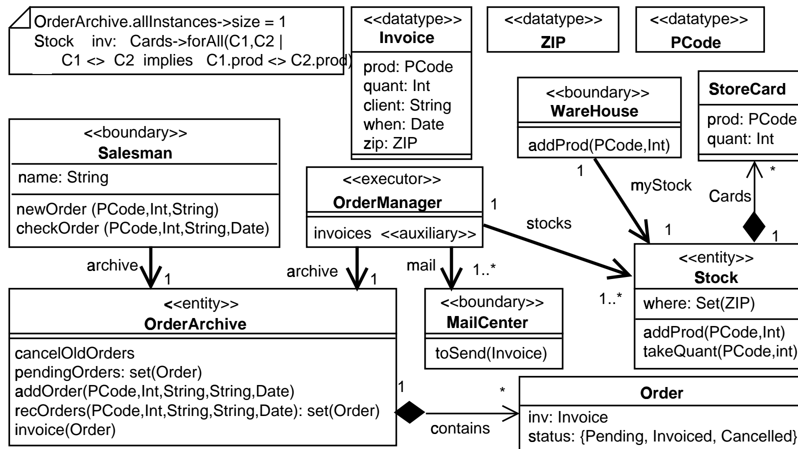


Fig. 1. ORDS-PIM Static View

The static view of the ORDS-PIM, in Fig. 1, shows that the designed system has some interfaces (≪boundary≫ classes) for interacting with the entities of its context: the salesmen, the warehouse workers, which refill the stocked products, and the mail centers, which receive the data to prepare the paper mails. The ≪executor≫ class OrderManager models how the orders, collected by the salesmen, are automatically processed. The persistent data used by the system, as the order archive and the stocks stored in the warehouses, are modeled by the ≪entity≫ classes. The invariants on these classes, reported in the same picture, require that there exists a unique OrderArchive, and each stock contains at most one card for each product.

The behaviours of the active classes (≪boundary≫ and ≪executor≫) purely consist in reacting to time deadlines or to signals received from outside; they can be given by trivial statecharts that can be found in [13], together with the definition of the operations.

The method requires all the classes to be fully encapsulated, that is, their instances may be accessed only by calling their operations. Moreover, all the dependencies among classes due to an operation call, have to be made explicit

by an ≪access≫ association, whose stereotype is offered by the method as well. Therefore, the ≪access≫ associations (visually presented by thick lines) fully depict the interactions among the system constituents.

2 A P2P Paradigm

At the moment no standard P2P paradigm seems to be established, not even in the practice. Not only several P2P-oriented middlewares have been proposed (see, e.g., [4, 11, 7, 9, 8], the Jxta Initiative (`http://www.jxta.org/`)), or the Bayou System (`http://www2.parc.com/csl/projects/bayou/`), based on different approaches, but several distributed applications have been developed directly implementing the P2P infrastructure for each of them (see [13] for a short list of some such systems).

In the spirit of our proposal for the introduction of an Architecture Specific level within the MDA approach, we present an abstract P2P paradigm for (mobile) distributed systems, trying to abstract as much as possible from those aspects that have different instantiations in the various proposals. Thus, a model based on the paradigm presented here is flexible enough to be implemented using different P2P middlewares and hence is not platform specific. For a more detailed discussion on the features of the proposed P2P paradigm and its relationship with existing P2P middlewares see [13].

2.1 P2P Paradigm Basic Concepts

For us a *P2P system* is a network of autonomous computing entities, called *peers*, with equivalent capabilities and responsibilities, distributed each on a different host. Each peer may initiate a communication/cooperation and contributes to the activity of the system by offering resources, accessing the resources of the other peers, and performing private activities. *Resource*, here, is used in a broad sense and includes, for instance, (persistent) data, services, ports and sockets. Consequently "accessing a resource" may include making queries and updating data, calling a service, sending a message on a port, or publishing an event.

In our paradigm, we assume that resources may be accessed both in a nominated way, by their (physical or logical) address, or in a property oriented fashion as the result of a query operation for all the resources satisfying a given property. Moreover, to prevent unauthorized access to some resources we assume that the peers are parted in possibly overlapping *groups*, that create a secure domain for exchanging data, disregarding the actual realization of such boundaries. Then the resources of a peer are divided among the groups it belongs to, and the resources relative to a group will be accessible only to the members of that group. In other words, the local space of each peer is partitioned into a *private* and, for each group the peer is member of, a *public* part, that may be accessed by all and only the members of that group. The private part includes all the activity of the peer and is, hence, the only part where the actual access of the (public

and external) resources takes place. The public parts offer the resources needed for the group collaboration.

Since we want to address the mobile case, peers are not required to be permanently available to the groups of which they are members. Thus, while the *membership* of peers to groups is statically decided, their *connection to* a group dynamically changes. Each peer may decide to explicitly join or leave a group changing, as a side effect, the available resources of that group by adding or removing those resident on the peer itself. The capability of (joining) leaving a group is more flexible than simply (connecting to) disconnecting from a network, because it allows to selectively (share) hide the part of the peer resources public on that group. As the presence of peers on a group may dynamically change[1], at any given time a peer connected to some group may access only those resources offered by the peers that are currently connected to such group.

So far, we have discussed the P2P paradigm abstract features. Now, we add a somewhat orthogonal assumption, that the P2P paradigm be supported by some *middleware*, offering a common framework where coordination and cooperation of peers is supported, and the changing network topology is hidden. Then a peer will be a host where, besides the private and the public parts, a software component realizing the middleware services (*middleware component*) resides on. Such middleware component has the absolute control of the resources, that can be accessed only through the middleware primitives. There is just one basic access primitive for this level of middleware, from which other operations can be derived, by specializing some of its parameters. A call of such primitive, named perform, corresponds to selecting a community of resources and performing an action on each of them. Such community of resources is the union of the public communities for some group on a set of peers. Both the group and the set of peers are parameters of the call, as well as the action to be performed.

The middleware is also the unique responsible for group management, in the sense that joining and leaving groups, finding the (currently connected) members of a group are services of the middleware, realized by the "middleware component" resident on the peer and offered to the (private part of the) peer.

2.2 Mapping ORDS into the P2P Paradigm Architecture

In this section we informally present how to translate the ORDS-PIM, defined in Sect. 1.2, into a distributed P2P system following the paradigm of Sect. 2.1. Then, that system will be designed using the UML profile of Sect. 3, getting the P2P oriented ASM for the ORDS case.

To start, we need a rough description of the distributed structure of the system, that is which computers will be available, their users, and their connections to the network. In this case, we will also introduce some mobility aspects, to show how they are handled in the proposed approach. Let us assume that

[1] Following the approach in [1,9,5], we assume that the middleware masks unannounced disconnections, by mechanisms like caching and some reconciliation policy. That is, we regard the groups as realized on an everywhere available connection net, and trust the middleware to solve the problems due to this idealization.

- The company is structured in different branches.
- Each salesman, branch, warehouse and mail center owns a computer.
- The computers of the salesmen are portable and can be connected to Internet by means of a modem, while all the others are assumed to be workstations stably connected by Internet.

The first step to derive a P2P model from the PIM is to devise the peers composing the system. In this case, obviously, we have a peer for each salesman, each warehouse, each branch, and each mail center.

The next step requires to deploy the ORDS-PIM active objects on such peers. In this case the deployment of the ≪boundary≫ objects (Salesman, MailCenter and WareHouse) is quite obvious, because their external users own a peer each. The OrderManager objects could be in principle be deployed on any host, as they interact equally with all the boundary elements. We decide to deploy an OrderManager on each branch peer, to exploit their computational capabilities.

The groups allow to discipline the cooperation among the peers, which should cooperate only if some active objects deployed on them were already cooperating within the ORDS-PIM, that is if there is an ≪access≫ association, between their classes (see Fig. 1). We have *direct cooperations*, when active objects access each other, like for instance OrderManager accessing MailCenter, or *indirect cooperations* when different active objects access the same passive object, like for instance Salesman and OrderManager sharing OrderArchive. Thus, in a first approximation, we can devise three groups for supporting the cooperations in Fig. 1: Mail (among branches and mail centers), Product (among branches and warehouses), and Company (among branches and salesmen).

The objects of ≪entity≫ classes that are accessed only by objects already deployed on a unique peer, will be deployed on the same, while those that are accessed by objects deployed on different peers need to be shared on some group. In principle, they could be deployed on any peer, but in order to minimize communications and to maximize availability, it may be more convenient to deploy them on one of the peers involved in the sharing. Thus in our example, the unique OrderArchive may be deployed either on the branches (together with the order managers) or on the salesman peers. Since the former are more stably connected and more powerful it seems sensible to use the branch peers to store the OrderArchive. Furthermore, we choose, as reasonable in a P2P setting, a distributed realization of the order archive, splitting it in several subarchives, one for each branch, that will contains the orders handled by the manager deployed on such peer. Analogously, the Stock objects could be deployed on the WareHouse or on the Branch peers. But, since each stock entity represents the status of the corresponding warehouse and may be accessed by several branches we deploy Stock objects on WareHouse peers.

As a final step, all the calls of the operations of the objects that are now resources shared on some group, have to be converted into appropriate calls of the middleware services.

3 A Profile for Peer-to-Peer ASM

In this section we will present, as a UML profile (see [12]), a visual object-oriented notation to model the P2P oriented ASMs following the paradigm illustrated in the previous section.

Since we integrate our P2P paradigm within an object-oriented paradigm, it is most natural to consider, as resources to be *shared*, standard *objects*. Thus resources, being objects, hence naturally typed by their classes, are implicitly provided with a precise interface. Moreover, we may use the OO typing mechanism at a higher level to classify the peers and groups constituting the system.

There are two key points in our profile. First, we represent the middleware component on each peer as an object of a class predefined by the profile, offering as operations the middleware primitives. Thus, the access to its primitives is disciplined by the standard mechanism of operation call (see Sec. 3.4 and 3.5).

Second, a model will consists of several UML (sub)models describing, respectively, the system architecture, the kinds of involved peers and groups. The *P2P Static View* describes the architecture of the system at the static level. It presents the types for the peers and groups building the system. The *Architecture Diagram* describes the actual instances of the peer and group types building the system and the memberships among them. The *Resource Community Model* describes the resources of the system shared on the groups of a type. The *Peer Model* describes the private part resident on the peers of a type, and which resources they offer and expect to find on each group they belong to.

That splitting of the modeling into different views corresponds to a separation of concerns during the design phase, providing means for the specification of each part of the system in isolation, as far as possible, making explicit the assumptions on the outside world. Moreover, the consistency among these views provides a useful tool to check that the intuitions about the resources needed for the performance of some group activity meet the description of the same activity from the viewpoint of the involved partners.

3.1 P2P Static View

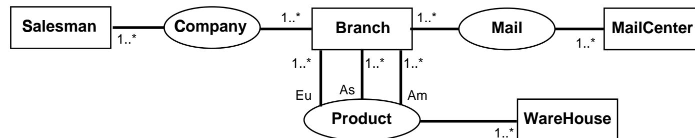
The P2P Static View presents the types of peers and groups used in the P2P system and for each peer type, the possible groups its instances belong to.

In a P2P Static View only classes of the stereotype ≪peer≫ or ≪group≫, that are classes without attributes and operations, and associations of stereotype ≪member≫ may be used.

≪member≫ are oriented associations going from a ≪peer≫ class, represented by a box icon, into a ≪group≫ class, represented by an oval icon, where the multiplicity on the group end is always 1 (and hence it is omitted). In this way the association names allow to uniquely identify the groups a peer of this type belongs to. If there is only one anonymous association from a peer type into a group type, then it is implicitly named as the group type itself.

As discussed in Sect. 2.2, the peers of our P2P realization of ORDS may be classified in four types: Salesman, Branch, MailCenter and WareHouse, while the

groups are of three types: Company, Product and Mail. In order to show a case where several instances of the same group type exist and different associations for the same peer types, let us decide that different groups of type Product represent a continent each. Each warehouse will serve (zip codes within) one continent and is, hence, connected to one Product group, while the branches need to be connected with all the groups of that type, in order to deal with orders for each locations. This classification will restrict the search space for a warehouse capable of handling an order. Such peer and group types are summarized below

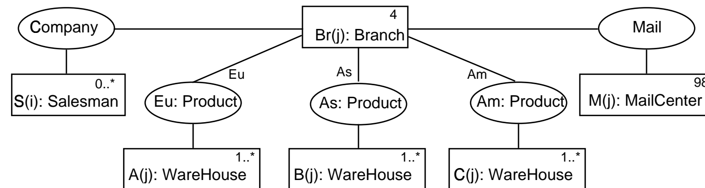


3.2 Architecture Diagram

The Architecture Diagram describes the structure of the modeled P2P system by stating which are the peers and the groups building the system and the memberships among them. It is a collaboration at the instance level satisfying the following constraints.

- The instances, represented as ClassifierRole, must belong to the types presented in the P2P Static View (and are visually depicted using, as previously, the box and the oval icons).
- The links, all belonging to the ≪member≫ associations present in the P2P Static View, are labeled with the corresponding association name.

If the number of the peers and groups composing the system is not determined a priori, or they are too much to be shown on a diagram, we can attach to the object icons multiplicity indicators expressing the number of instances.



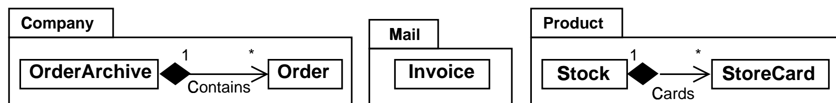
The Architecture Diagram of the ORDS-P2P, above, shows that there is one group of type Company (Mail) connecting all peers of appropriate types, and three groups of type Product, and moreover that each warehouse is member of one of them while each branch is member of each such group. In this case, we have 4 branches, 98 mail centers, and any number of salesmen and warehouses.

3.3 Resource Community Model

A Resource Community Model describes the types of the resources shared by the members of any group of a given type. It simply consists of a UML package

named as the group type itself containing at least a class diagram, where the new special OCL basic types PeerId, GroupId and ResourceId may be used. These types corresponds to peer, group and resource identities to be used as arguments and results of the middleware primitives. They are used as a bridge among the different views composing a UML-P2P model. Since intuitively the resources are manipulated by the peer internal activity through the middleware, no calls of the middleware primitives are allowed within a resource community model.

The models of the resource communities of the ORDS-P2P are presented below, and each of them consists of the shared data needed to realize indirect cooperations as discussed in the previous section. The details of the definition of each class are omitted, because they are as in the ORDS-PIM.



3.4 The Middleware

We introduce the middleware in our profile by defining a UML interface, P2PMW, whose operations correspond to the services that it offers. Notice that the primitives provided by the (abstraction of the) middleware presented here are not intended to match directly any existing middleware. Indeed, we are aiming at a profile for the support of an intermediate level of design where the platform has yet to be decided, and only the architecture paradigm of the system has already been fixed. We may complement the P2PMW interface with a UML class realizing it, say P2PMWclass, which will be then part of the profile definition. The UML description of the operations of P2PMWclass will give an abstract semantic description of the middleware primitives; whereas the attributes of P2PMWclass will give an abstract view of the information on the network managed by the middleware and on the current activities of the middleware itself.

The middleware primitives use the special types PeerId, GroupId and ResourceId, already introduced, and also RemoteAction, defining what can be done on a selected community of resources. RemoteAction is a specialization of UML action (Action is the corresponding meta-class), adding two new actions:

- returnCopy that will make a deep copy of its argument in the private community of the calling peer and return its identity,
- returnRef that will give back the identifier (element of the special type ResourceId) of the argument. Such resource identifier, whenever used in a remote action in the correct resource community will identify the original resource.

A remote action, as any UML action, allows to model both querying and imperative updating over a community of objects. To describe a query or an update on some particular resources in a community, we can use directly the identifiers of such resources, corresponding to a direct reference to the resources in a named style of middleware. But, it is as well possible to find them indirectly, for example by using an OCL expression of the form C.allInstances->select(...) for

selecting all resources of class C satisfying some condition, achieving in this way the anonymous style of resource lookup favoured by some middleware.

A RemoteAction must be statically correct in the context of the communities on which it will operate; in particular no references to the caller environment may appear in it. That constraint on one side allows remote evaluation, and on the other bans interlinks between private and public communities (of different groups) and among the communities local to different peers, as the users cannot exploit (private) local object identities when assigning values to the attributes of (possibly remote) public objects through code execution.

The primitives of P2PMW are:

- mySelf():PeerId returns the identifier of the peer, where the middleware component is resident.
- join(GroupId) connects to the given group.
- leave(GroupId) disconnects from the given group.
- isConnected(GroupId):Boolean checks if there is an active connection to the given group.
- whoIsConnected(GroupId):Set(PeerId)] returns the set of the identifiers of the peers currently connected to the given group.
- members(GroupId,PeerType):Set(PeerId) given g and PT, returns the set of the identifiers of the peers of type PT that are members of g[2].
- perform(GroupId,Set(PeerId),RemoteAction):Set(OclAny) given g, ps, and ra, executes ra in all the public communities for group g of those peers in ps that are currently connected. Then, it collects the results of any returnCopy and returnRef actions, producing a, possibly empty, object collection and yields it as result.

3.5 Peer Model

A Peer Model describes the software required by the modeled P2P system on the peers of a given type. Hence, in a UML-P2P model there will be a Peer Model for each peer type present in the P2P Static View.

A Peer Model for a peer type PT whose instances have the capability of being member of group of the types G1, ..., Gk consists of the following parts:

- :P2PMW, a denotation of the used middleware component, that is a *ClassifierRole* for the interface defined in Sect. 3.4, to recall that it is possible to use its operations in the private part.
- for each group type GT in {G1, ..., Gk}:
  * a package GT-public defining the static structure of the public resource community made available to any group of type GT the peers of type PT belong to.
  * a package GT-external defining how the peers of type PT view the external resource communities provided by the other members of any group of type GT they belong to.

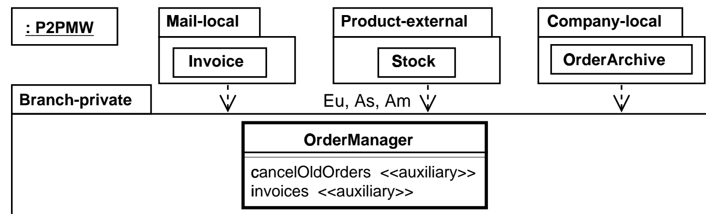
[2] Since memberships are statically fixed, the result of this operation is a constant, needed only to give the peers a linguistic means to access the group members.

Each of the two packages for GT defines a resource community by a class diagram that is a subset of the one of the GT Resource Community Model. If any of these two packages is empty (because the peers do no offer resources or do not require resources on such group), then it will be omitted in the diagram.

– a package, named PT-private, that describes the part of the system resident on the peers of that type, parameterized by the references to the groups it is member of (determined by the ≪member≫ associations in the P2P Static View). Such parameters are depicted over the lines connecting the packages corresponding to their types.

  A private part is any UML model, where the calls of the middleware object operations may appear, and that implicitly imports all the packages describing the resource communities that the peer may access. There are some obvious static correctness requirements: a peer may only join/leave a group of which it is member; the calls to the primitive perform on some resource community has to be correct w.r.t. the "type" of the community itself. The type of its community and of the communities of different peers on a group of type GT is given by the two packages public and external for GT.

Example: Branch Peer Model Here we illustrate the use of the Peer Models on the example of the Branch peer type, shown below. The other Peer Models of (a slightly different version of) the ORDS-P2P example can be found in [13], as well as the full details of the Branch Peer Model, that we omit here for brevity.



In Sect. 2.2 we decided to deploy one object of the ≪executor≫ class OrderManager on each branch peer; thus we put the corresponding class in the private part of the Branch peer model. Moreover, we decided to realize the unique database of orders in a distributed way, deploying the orders managed by each branch on its peer. Thus, since the OrderArchive has to be shared with salesman, we will have its class in the Company-public package. Analogously, as the branches need to access the warehouse catalogues, we will have the corresponding class Stock in the Product-external package. Notice that in the Branch Peer Model there is no information about the actual deployment of the Stock objects; it is simply assumed that they are provided by peers on the group Product. Thus, a change in the design of the actual location of those objects would not affect the design of this kind of peers.

The OrderManager has to be slightly modified in order to accommodate the P2P realization and in particular the accesses to the shared data have to be mapped onto calls to the middleware primitives. Thus, for instance the PIM

version of the invoices method has to be modified by using the middleware to access the order archive, the stock and the mail. We leave as comments the PIM version of the shared resource access, to better show the difference.

```
method invoices()
  { os = perform(Company,mySelf,
      returnRef OrderArchive.allInstances.pendingOrders());
    \\ PIM version: os = archive.pendingOrders()
    for O in os do {
      z = O.inv.zip;
      g = z.zip2area();
      ss = perform(g,any,
        if Stock.allInstances->select(S.where->includes(z))<>{} then{returnRef mySelf};);
      \\ PIM version: ss = astocks->select(S.where->includes(O.ZIP));
      done = False;
      while(ss <> {} and not done) do
        { done = perform(g,ss->first,
          Stock.allInstances.takeQuant(O.prod,O.quant););
        \\ PIM version: done = ss->first.takeQuant(O.prod,O.quant);
        if done then { perform(Company,mySelf,OrderArchive.allInstances.invoice(O));}
        else ss = ss - ss.first } }
        \\ PIM version: archive.invoice(O);
        \\ Moreover, in the PIM version it performed mail.toSend(O.inv); while
        \\ here we leave that responsibility to the mail centers; see comment below
```

Notice that, since the warehouses are now classified by their area, we need to know how the ZIP codes are mapped onto the the group identifiers in order to perform a search on the correct group. This is realized by zip2area of the ZIP class with result type GroupId, that is, hence, added to the class. Notice that the OCL operation .allInstances refers to those instances in the community determined by the enclosing call of perform, so for instance the first occurrence in the previous method refers to the public community for the group Company of the peer itself (if it is currently connected, otherwise it is empty).

A further difference in the definition of invoices in the distributed case is that we move the responsibility for calling the toSend operation is moved from the OrderManager to the MailCenter, that will periodically perform a query for invoiced orders, send the corresponding mail and update their status.

3.6 Static correctness of the overall model

Besides the standard UML constraints on the static correctness, we impose some further conditions, following the principle that a strict typing helps the design of systems, by allowing an early, if rough, check of consistency.

As the different parts of a model correspond to different views of the same system, besides the static correctness of each view, already stated before, we have to check for consistency among them. This has already been partially done. For instance, the form of each peer model depends on the P2P Static View and the classes admissible in each public or external package of some group type

GT have to appear in the Resource Community Model for GT. Now, we check the GT-public and GT-external packages for GT from all the Peer Models of the possible members one against the other in order to be sure that if a peer expects some resources from the community, some (other) peer is offering it on the *same* group. Hence, we require that for each group in the Architecture Diagram and each peer member of it, the GT-external package of the type of that peer is included in the union of the GT-public packages of the types of that group members. This condition does not guarantee that all the expected cooperations will really take place, as it is possible that the peers providing some resource are not connected at the same time as the peers needing such resource. But, this kind of failure is due to the dynamic configurations of the groups (to the actual presence of the members) and cannot be checked statically.

4 Conclusions

From the experience of some projects dealing with significant case studies, we have been impressed by the gap existing between the proposed rigorous techniques for software development from scratch and the use of the various kinds of middleware in the practice without any methodological support. In the context of the MDA (*Model Driven Architecture*), we have proposed the introduction of an intermediate level, between PIM and PSM, called ASM (*Architecture Specific Model*) and we have illustrated our proposal in the case of a P2P architecture where distribution and mobility are fully encapsulated by some middleware. The middleware is naturally integrated into the OO paradigm of UML, describing its software components present on each host as objects whose operations correspond to its primitives. The idea of representing the middleware components as objects is general and powerful enough to be reused whenever designing a UML profile to model applications using some middleware.

In order to complete the MDA picture, we need to define in the general case how to transform a PIM into an ASM presented by using UML-P2P. In this paper, we have just sketched how to handle the case of the ORDS application. The mapping will be given by a set of systematic guidelines.

Moreover, to fill the MDA landscape towards the PSM for the special case of P2P architecture, we further need to define the notations to express the PSM, that are UML profiles for specific P2P middlewares, such as PeerWare [4], Jxta [15], Xmiddle [9] et cetera, and guidelines to translate an ASM into the corresponding PIM. We can define such profiles following the way we have defined UML-P2P, just replacing the abstract generic ingredients with the more specific ones supported by that particular middleware.

Acknowledgments We acknowledge the benefits of many discussions with the colleagues of the Sahara project, and especially the designers of PeerWare, G.P. Cugola and G.P. Picco.

References

1. A. Arora, C. Haywood, and K.S. Pabla. JXTA for J2ME^{TM}Extending the Reach of Wireless With JXTA Technology. Technical report, Sun Microsystems, Inc., 2002. Available at `http://www.jxta.org/project/www/docs/JXTA4J2ME.pdf`.
2. D. Balzarotti, C. Ghezzi, and M. Monga. Supporting configuration management for virtual workgroups in a peer-to-peer setting. In *Proc. SEKE 2002*. ACM Press, 2002.
3. J. Charles. Middleware Moves to the Forefront. *Computer*, 32(5):17–19, 1999.
4. G. Cugola and Gian P. Picco. PeerWare: Core Middleware Support for Peer-to-Peer and Mobile Systems. Manuscript, submitted for publication, 2001.
5. A. Demers, K. Peterson, M. Spreitzer, D. Terry, M. Theimer, and B. Welch. The Bayou Architecture: Support for Data Sharing amoung Mobile Users. Technical report, Xerox Parc, Santa Cruz, CA, US, 1994.
6. W. Emmerich. Software Engineering and Middleware: A Roadmap. In A. Finkelstein, editor, *The Future of Software Engineering*, pages 117–129. ACM Press, 2000.
7. Jatelite-System. Jatelite White Paper. Available at `http://www.jatelite.com/pdf/jatelite_en_whitepaper.pdf`, 2002.
8. G. Kortuem, J. Schneider, D. Preuitt, T.G.C. Thompson, and Z. Segall S. Fickas. When Peer-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobile Ad hoc Networks . In *Proceedings of 1st International Conference on Peer-to-Peer Computing (P2P 2001)*. IEEE Computer Society, 2002.
9. C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich. XMIDDLE: A Data-Sharing Middleware for Mobile Computing. *Wireless Personal Communications*, 21:77–103, 2002.
10. OMG Architecture Board MDA Drafting Team. Model Driven Architecture (MDA). Available at `http://cgi.omg.org/docs/ormsc/01-07-01.pdf`, 2001.
11. A. Murphy, G. Picco, and G-C. Roman. Developing Mobile Computing Applications with Lime. In M. Jazayeri and A. Wolf, editors, *Proceedings of the 22th International Conference on Software Engineering (ICSE 2000), Limerick (Ireland)*, pages 766–769. ACM Press, 2000.
12. OMG. White paper on the Profile Mechanism – Version 1.0. Available at `http://uml.shl.com/u2wg/default.htm`, 1999.
13. G. Reggio, M. Cerioli, and E. Astesiano. Between PIM and PSM: the P2P Case. Available at `http://www.disi.unige.it/person/ReggioG/`, 2002.
14. J. Siegel and the OMG Staff Strategy Group. Developing in OMG's Model-Driven Architecture (MDA). Available at `ftp://ftp.omg.org/pub/docs/omg/01-12-01.pdf`, 2001.
15. Sun-Mycrosystem. Jxta Initiative. WEB site `http://www.jxta.org/`, 2000.
16. Xerox-Parc. The Bayou Project. WEB site `http://www2.parc.com/csl/projects/bayou/`, 1996.