Between PIM and PSM: the P2P Case[1)]

Gianna Reggio Maura Cerioli Egidio Astesiano

DISI, Università di Genova - Italy

{reggio,cerioli,astes}@disi.unige.it

1) Partially supported by Murst - Programma di Ricerca Scientifica di Rilevante Interesse Nazionale SAHARA (Software Architecture for Heterogeneous Access Networks infrastructure).

Abstract

We address in general the problem of providing a methodological and notational support for the software development at the design level of applications based on the use of a middleware. In particular, because of the growing interest in mobile applications with nomadic users, we consider the middlewares supporting a peer-to-peer architecture. In order to keep the engineering support at the appropriate level of abstraction we formulate our support within the frame of Model Driven Architecture (MDA), the approach proposed by OMG that considers the design development starting with a Platform Independent Model (PIM) then mapped into one or more Platform Specific Models (PSM) for different platforms, where a model has to be understood in a UML sense.

In this paper our main aim is to advocate the introduction of an intermediate abstraction level, called ASM for Architecture Specific Model, that is particularly suited to abstract away the basic common architectural features of different platforms, further separating concerns and allowing a greater flexibility and reuse when implementing on specific platforms. That solution seems particularly advantageous in presence of many proposals of peer-to-peer middlewares.

Technically, the support for the ASM level is presented in the form of a UML, hence object-oriented, profile which embodies a basic abstract peer-to-peer paradigm architecture. Within that profile the connection with middleware is for each peer naturally represented by an object whose operations correspond to its primitives.

To be concrete all concepts and notations are illustrated by means of a running example.

Introduction

There are three sources of inspiration and motivation for the work presented in this paper:

- the growing demand for applications characterized by decentralization and mobility for which the peer-to-peer paradigm seems to have some clear advantages;
- the recognized need of appropriate engineering support at the design level in the development of middleware-based software;
- the OMG [1] Model Driven Architecture approach, suggesting that Platform Specific Models should be derived from more abstract Platform Independent Models.

The use of mobile devices (mobile computing) supporting applications for users moving from one location to another is increasing and will likely become popular. That kind of applications needs an appropriate underlying architecture. As it is easily understandood and explicitly advocated by many researchers, see, e.g., [Balzarotti *et al.* 2002], particular advantages seem to be provided by the so-called peer-to-peer architecture, namely a network of autonomous computing entities, called *peers*, with equivalent capabilities and responsibilities. Recently many proposals have been put forward for peer-to-peer (shortly P2P) architectures, by researchers and companies, mainly under the form of P2P middlewares (see, e.g., [Murphy *et al.* 2000; Xerox-Parc 1996; Jatelite-System 2002; Mascolo *et al.* 2002; Kortuem *et al.* 2002; Sun-Mycrosystem 2000]). We have been particularly stimulated by the proposal of PeerWare [Cugola and Picco 2001], that has been analyzed and used as a paradigmatic example within the project SALADIN[2].

The role and the importance of the middleware in the development of distributed systems is now well recognized (J. Charles, [Charles 1999]). The fundamental reason is that “middleware enables application engineers to abstract from the implementation of low-level details” (W. Emmerich in [Emmerich 2000]). The use of a middleware, encapsulating the treatment of distribution and mobility and providing appropriate abstractions for handling them in a transparent way, improves and simplifies the development and maintenance of such applications. However there is a danger in the current success of the use of middleware in the industry; namely, the direct use of middleware products at the programming level without an appropriate software engineering support for the other development phases. On the other hand, as it is argued in [Emmerich 2000], the software engineering support, at the design level at least, has to take middleware explicitly into account. Thus, “the software engineering community needs to develop 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;

[1] `http://www.omg.org/`

[2] Software Architecture and Languages to coordinate Distributed Mobile Components`http://saladin.dm.univaq.it/`

that approach is illustrated, as motivated before, in the important case of P2P middleware. However we propose a non-conventional way of approaching that problem at a level of abstraction that we think is more appropriate, perhaps even in principle, but especially in the current situation.

Indeed, at the moment there is an abundance of proposals, but no standard P2P paradigm seems to be established, not even in the practice. That situation, common also to other kinds of middlewares and more generally implementation platforms, poses a basic problem for platform-oriented software development. Relying on the features of a particular platform too early during the design phase may be dangerous, because the resulting model lacks in flexibility and cannot be reused for further implementations based on different platforms. For this reason, the so called MDA (for *Model Driven Architecture*), a technique proposed by the OMG, advocates the initial use of a *Platform Independent Model*, or PIM, designed following the natural structure of the system to be described, to be refined into one or more *Platform Specific Models*, or PSM, that are its specializations taking into account the features of the particular technology adopted and that will be adapted or even completely replaced accordingly to the frequent changes in the technology. However, in our opinion, the gap between an absolutely platform independent and a completely platform specific model can be too large. 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. That can be seen, as it will explained later in more detail, also in the case of the different proposals of middlewares supporting a P2P architecture.

Therefore, we advocate the introduction of an intermediate level, called ASM for *Architecture Specific Model*, where some basic architectural choices are made, fixing the concepts and the paradigm used, but still platform independent. In the case considered here of P2P middlewares, devising an ASM implies to define a basic abstract paradigm for P2P architectures; the one we present in the paper can be considered an *abstraction of most current proposals, but 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.* 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*, providing a visual support for the design based on the concepts and notions of the chosen paradigm. Notice also that the use of the UML is practically mandatory in MDA, because in that approach the various models of a system at different levels of abstraction have to be treated in a homogeneous notation and it is practically assumed that such a notation can currently be provided only by the UML. A UML-profile provides some mechanisms for specializing its reference metamodel in specific domains, e.g., to handle real-time systems. While many profiles have been proposed, few are middleware-oriented; notably the best known is a profile for CORBA, that however, as will be seen, is given following an approach totally different from our. We do not know of any UML-profile supporting the development of software based on P2P middlewares, what provides further motivation to our work.

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, taking into account the sygestion of W. Emmerich [Emmerich 2000] that the engineering support, at the design level at least, has to take the middleware explicitly into account. The same pattern can be specialized to provide profiles for the PSM level, simply specializing the classes representing the middleware objects and the resources and adding/removing features. Notice that 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 on which 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. In Sect. 3.2.7 we give some hints on how our approach may help checking consistency among the various views of the modelled system provided by our notation. Finally, in the last section we draw some conclusions, discuss the relationships to extant 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 [OMG Architecture Board MDA Drafting Team 2001; Siegel and the OMG Staff Strategy Group 2001], defines an approach to system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. To this end, the MDA defines an architecture for models that

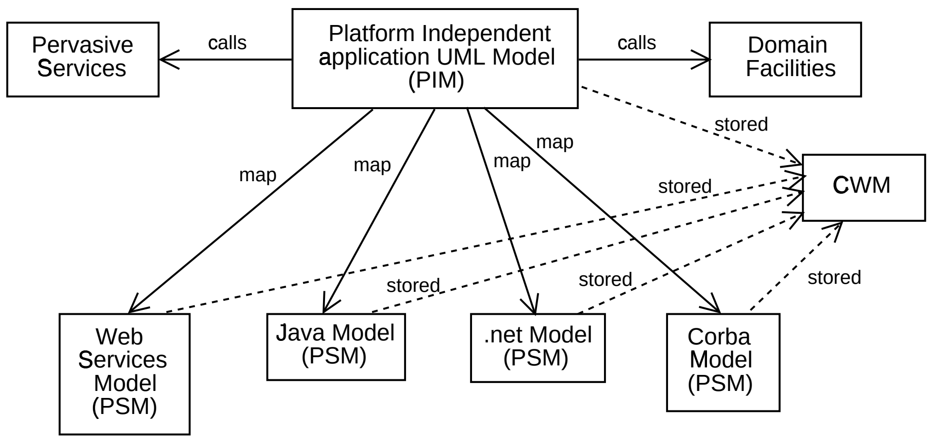


Figure 1: Model Driven Architecture Approach (MDA)

provides a set of guidelines for structuring specifications expressed as models. The MDA approach and the standards that support it allow a model specifying the system functionality to be realized on multiple platforms through mappings to specific platforms, and allows different applications to be integrated by explicitly relating their models, enabling integration and interoperability and supporting system evolution as platform technologies come and go. We simply summarize the MDA by the picture in Fig. 1, which will be commented below.

All MDA development projects start with the creation of a *Platform Independent Model* (*PIM*), expressed in UML and shown at the top of Fig. 1. The base PIM expresses only business functionality and behavior undistorted, as much as possible, by technology. The technological independence of the modeling environment (UML) allows the developers to ascertain, much better than they could if working with a technological model or application, that the functionality embodied in the base PIM is complete and correct. Another benefit: because of its abstraction, the base PIM retains its full value over the years, requiring change only when business conditions mandate.

The *pervasive services* in Fig. 1 are essential services, as Directory services, Event handling, Persistence, Transactions, and Security, that should be available to any application, which MDA proposes to be defined in a standard way (by a UML model) independently from any platform, and thus at the PIM level.

The *domain facilities* are services common and relevant for applications working in particular domains, such as Finance, E-Commerce, Telecommunication, Healthcare, ..., which, as the pervasive services, should be precisely and abstractly defined at the PIM level, again by a UML model.

The *Platform-Specific Models* (*PSM*) should be expressed in UML; however, since UML is independent of middleware technologies, it is not obvious how to harness this power to express a PSM, since we have to decide in which way to use UML to represent the platform ingredients. Such decisions can be defined by a UML profile (extension/variant of the basic notation).

During the mapping step, the run-time characteristics and configuration information that we designed into the application model in a general way are converted to the specific forms required by the target middleware platform. Guided by standard mappings, automated tools may perform as much of these conversions as possible.

All the models developed following the MDA will then be written by using either UML or one of its variants (profile), and all of them will be expressed by using MOF (Meta Object Facility), an elementary object-oriented notation (a basic subset of UML) apt to describe in an OO way the form of the models of a notation. Thus all the model around will be expressed by using MOF, and then have all a common format, which allows, for example, to easily define how to transform one into another. CWM is the Common Warehouse Metamodel the established OMG standard for repository containing models expressed by using MOF.

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: ORDERS. 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 that generate paper mail starting from electronic data. The orders are collected by salesmen, who may also verify the status of old orders. Moreover, salesmen may use the company structures to support the activity of their trade unions (just to exchange documents). There exist three unions, named “A”, “B” and “C” respectively.

The case study we present here is a simplified toy version, with a minimal amount of preserved features to illustrate our points. For example, we do not consider the aspects relative to the payment of the products.

Our abstract design of the ORDERS system (modelled by a PIM, named ORDERS-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 by objects of four kinds:

– *data*, just the values used inside the system,

– *boundary*, that take care of the interaction of the system with the external world,

– *entity*, that store persistent data (e.g., a database),

– and *executor*, that perform the core activities of the system.

The method offers appropriate stereotypes to denote elements of each kind: <<boundary>> and <<executor>> that specialize active classes, and <<entity>> that specializes passive classes. <<datatype>> is instead the standard UML stereotype for classes whose elements are pure values (no identity, no updatable state, operations are pure functions).

The static view of the ORDERS-PIM, see Fig. 2, is a class diagram using the above four stereotypes. For what concerns associations, the method offers the stereotype <<access>> (visually presented by a thick line) modelling the fact that the instances of a class call the operations of the instances of another one. The method requires all such dependences among classes to be made explicit by <<access>> associations. Moreover, since the classes of the four stereotypes above must be fully encapsulated, that is their instances may be accessed only by calling their operations, the <<access>> associations fully depict the interactions among the system constituents. The black diamond represents, as usual, strong aggregation.

Fig. 2 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, the stocks stored in the warehouses and the information about the union activity of the salesmen, are modelled by the <<entity>> classes. The invariants on these classes, reported in the same picture, require that

– there exists a unique OrderArchive,

– there are three unions, named “A”, “B” and “C”, and a salesman must be member of one of them,

– and a stock contains at most one card for each product.

The behaviours of the active classes (<<boundary>> and <<executor>>), purely consisting in reacting to time deadlines or to signals received from outside, are given by the very simple statecharts reported in Fig. 3. The operations of the passive classes (<<entity>>) and the <<auxiliary>> operation[3] invoices of the active class OrderManager are defined in Fig. 4.

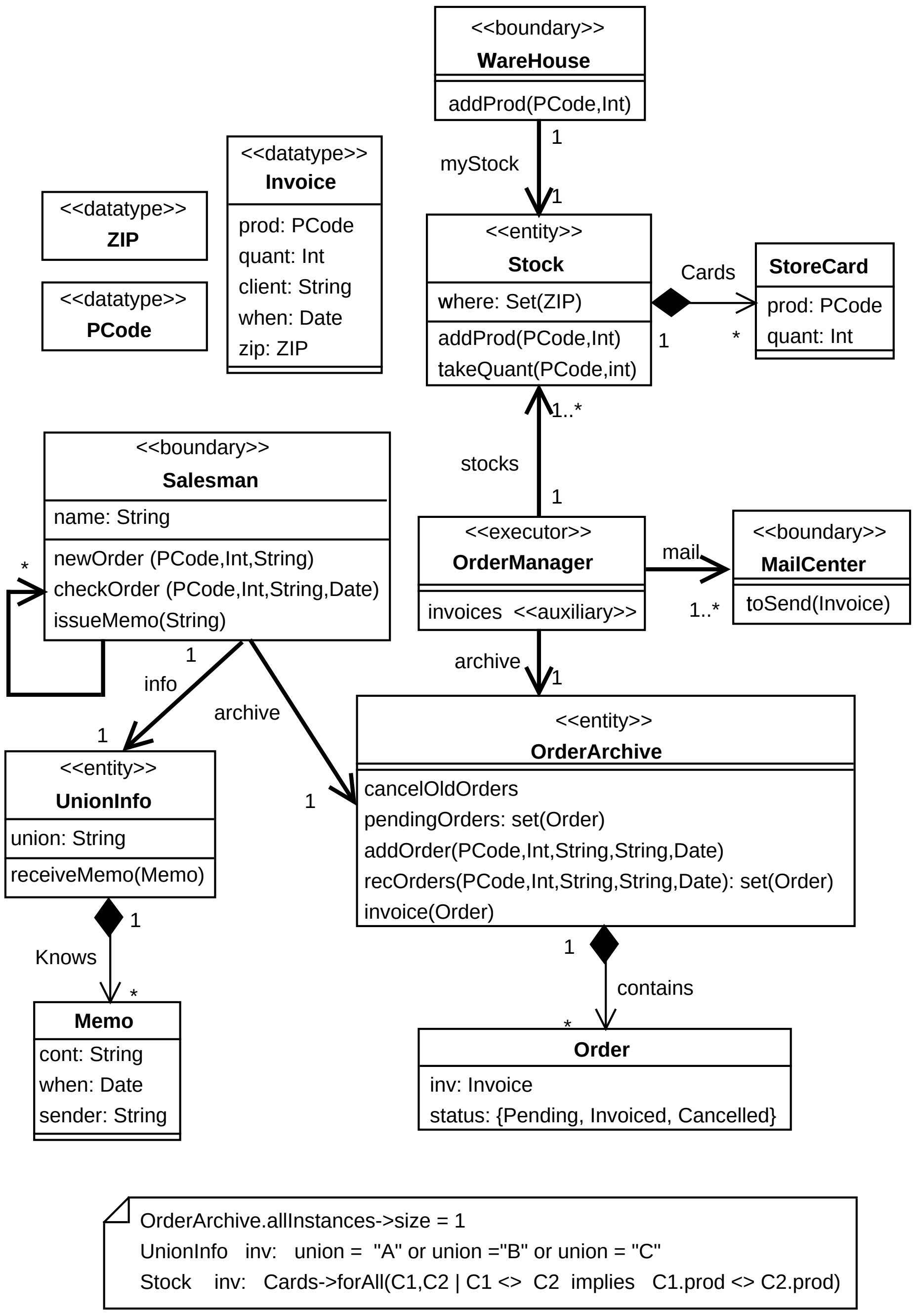
Here and in the following, we use O = create(CLASS,v1,. . . ,vn) as an abbreviation for O= create(CLASS); O.A1=v1, . . . , O.An=vn), where A1, . . . , An are all the attributes of class C in the order in which they appear in the class diagram.

2 A P2P PARADIGM

2.1 P2P Paradigm Basic Concepts

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., [Cugola and Picco 2001; Murphy *et al.*

[3] In order to keep the statecharts defining the behaviour simple enough, our design method allows to define <<auxiliary>> operations of active classes, which cannot be used to generate call events and must be defined by an associated method.



```
OrderArchive.allInstances->size = 1
UnionInfo   inv:   union =  "A" or union ="B" or union = "C"
Stock    inv:   Cards->forAll(C1,C2 | C1 <>  C2  implies   C1.prod <> C2.prod)
```

Figure 2: ORDERS-PIM Static View

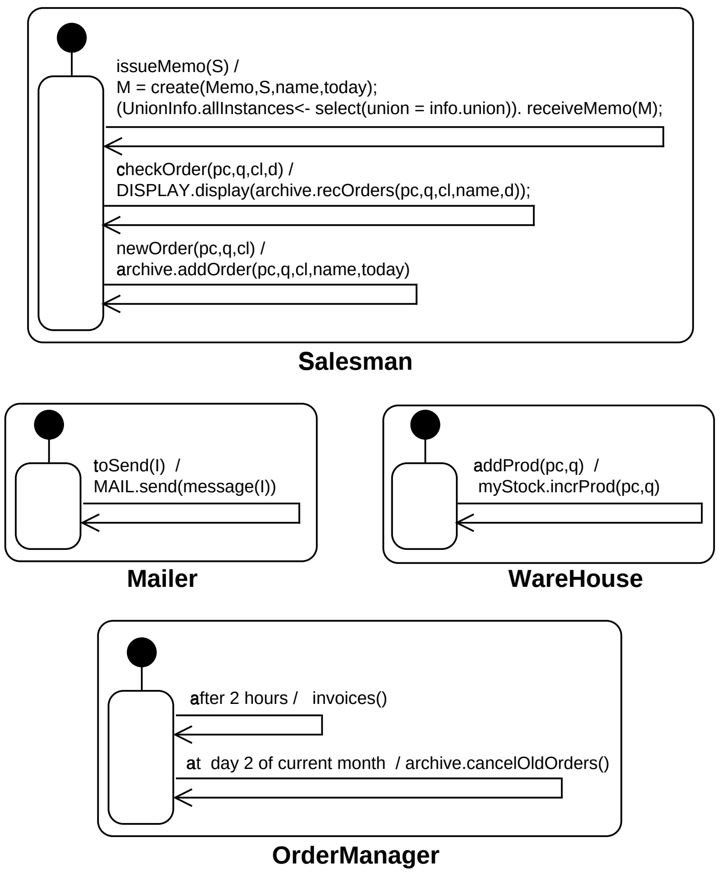

issueMemo(S) /
M = create(Memo,S,name,today);
(UnionInfo.allInstances<- select(union = info.union)). receiveMemo(M);
checkOrder(pc,q,cl,d) /
DISPLAY.display(archive.recOrders(pc,q,cl,name,d));
newOrder(pc,q,cl) /
archive.addOrder(pc,q,cl,name,today)
Salesman
toSend(I) /
MAIL.send(message(I))
Mailer
addProd(pc,q) /
myStock.incrProd(pc,q)
WareHouse
after 2 hours / invoices()
at day 2 of current month / archive.cancelOldOrders()
OrderManager

Figure 3: ORDERS-PIM: Behaviours of the active classes

```
context UnionInfo::receiveMemo(M) post: self.Knows->includes(M)
context OrderArchive::cancelOldOrders() post:
   (Contains->select(O.status = Pending and O.when > (today-3 month))).status = Cancelled
context OrderArchive::pendingOrders() post: result = Contains->select(O.status = Pending)
context OrderArchive::addOrder(pc,q,cl,s,d) post:
   Contains = Contains@pre->union({ create(Order,create(Invoice,pc,q,cl,s,d),Pending) })
context OrderArchive::recOrders(pc,q,cl,s,d) post:  result = Contains->select(O.prod = pc and ...)
context OrderArchive::invoice(O) post: Contains->includes(O) and O.status = Invoiced
context Stock::takeQuant(pc,q) post:
   if Cards->select(SC.prod = pc and SC.quant => q) <> {} then
      (first(scs).quant = first(scs).quant@pre-q and result = True) else result = False
context Stock::incrProd(pc,q) post:
   (Cards->select(SC.prod = pc))->forAll(SC.quant = SC.quant@pre+q)
method invoices():
   { for O in archive.pendingOrders() do
      ss = stocks->select(S.where->includes(O.ZIP)); FoundQuant = False;
      while(ss <> {} and not FoundQuant) do
         { FoundQuant = ss->first.takeQuant(O.prod,O.quant);
         if FoundQuant then { archive.invoice(O); mail.toSend(O.inv) }
         else ss = ss - ss.first } }
```

Figure 4: ORDERS-PIM: Definition of the operations of the Static View

2000; Xerox-Parc 1996; Jatelite-System 2002; Mascolo *et al.* 2002; Kortuem *et al.* 2002; Sun-Mycrosystem 2000]), based on different approaches, but several distributed applications have been developed directly implementing the P2P infrastructure for each of them, mainly in the fields of data sharing (see, e.g., Napster, the several Gnutella based clients, like, e.g., LimeWire or BearShare, Morpheus), cooperative frameworks (see, e.g., Groove[4)] or Magi Technology[5)]) and decentralized computing (see, e.g., [SETI@home 2002; Intel 2002; Porivo Technologies 2002]).

Here 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.

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 whole system by offering resources, by accessing the resources of the other peers, and obviously by performing private activities. *Resource*, for us, is a quite general term, including, e.g., data, persistent data and services, but also, for instance, ports and sockets. Consequently "accessing a resource" may include querying and updating data, calling a service, but also sending a message on a port or publishing an event.

In any case, the paradigm has to fix the modality of access to resources. At a low level of abstraction it seems inescapable to use its (physical or logical) address and, indeed, most existing platforms provides primitives to access a resource given its address (see, e.g., [Xerox-Parc 1996; Jatelite-System 2002; Mascolo *et al.* 2002; Kortuem *et al.* 2002; Traversat *et al.* 2002]). Such primitives may be made directly available to the users and/or exploited to define some higher-level operations to access all the resources satisfying a given property, by encapsulating a search procedure[6)] to get the addresses and the nominated access to the found resources. A few platforms (e.g., [Murphy *et al.* 2000; Cugola and Picco 2001]) hide the lower level resource addressing and let the user see only the property-based access operations. In our paradigm, we assume that both modalities of resource access are available; so resources, but also peers, have identities, that can be used to access them, but it is also possible to perform some operation over all resources satisfying some conditions.

Another important issue in resource access is how to prevent unauthorized accesses to some resources. As for the previous issue, there are several technical solutions, based on the implementation of different

[4)] `http://www.groove.net/`

[5)] `http://www.endeavors.com/`

[6)] There are several techniques to get the addresses of the resources satisfying a given property, mainly based on the idea that the needed associations between resources and addresses are stored in a particular kind of peers, or servers in hybrid systems, or are advertized by the peers offering the resources (e.g., [Sun-Mycrosystem 2000]), or may be looked-up with more or less efficient algorithms visiting the network. Though the choice of such techniques is central to the design of the platforms providing the look-up procedure, they are completely irrelevant, from the user viewpoint and are, hence of no interest here.

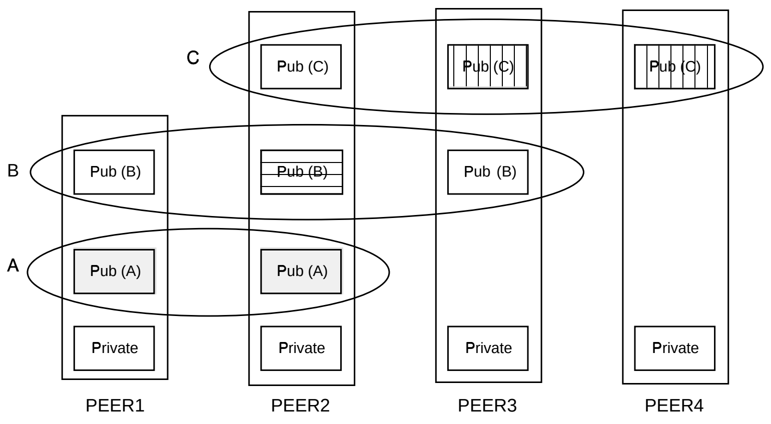


Figure 5: Resource Communities in a System

security protocols. Here, we simply 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 to whom it belongs, 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 only by 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 part(s) offers the resources needed for the group collaboration. A visual representation of the partition of the resource community of a simple system is presented in Fig. 5. The system consists of 4 peers organized in 3 groups (A, B, and C).

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 *presence in* or *connection to* a group dynamically changes. Each peer may decide to explicitly join or leave a group changing, as a side effect, the group available resources 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, at any given time a peer may access the resources of a group only if it has joined it and, clearly, only those resources offered by the peers that are currently connected to such group.

In the practice a different kind of dynamic reconfiguration is considered (see, e.g., [Sun-Mycrosystem 2000]), where peers and groups may be created and destroyed, and group memberships may vary. In this paper we do not consider these aspects, and thus which are the peers and the groups of a system and the group membership will never change during its life. We think that the dynamic reconfiguration aspects are orthogonal with respect to the kind of a distributed architecture, and thus they can be incorporated later. Resources, instead, may be dynamically created and destroyed.

For what concerns acting on the resources, a peer (or better its private part) that is connected to a group may:

- create a public resource for that group,
- destroy a set of resources,
- perform an operation specific for a kind of resource on a set of resource of such kind.

In the last two cases, the resource set must be part of the shared community of the group and each set element may be directly identified by its identity/address or the set itself may consist of all the resources satisfying some condition, following the two styles of access discussed before.

So far, we have discussed the P2P paradigm abstract features. Now, we add a somewhat orthogonal assumption, that is that the P2P paradigm is supported by some basic software level, the *middleware*, offering a common framework where coordination and cooperation of peers is supported, and the changing network topology is hidden.

A peer will be, then, 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 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 the selection of a community of resources and the performance of 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.

In Fig. 5, we have marked a few possible resource communities on which PEER2 may act. They are the public communities on group A of all the currently connected peers, its own public community on group B, or the public communities on group C of the two peers PEER3 and PEER4. Of course, these are not all the possible communities of resources to be selected for action performance in this particular configuration, but they are a representative collection.

The action to be performed is a further parameter of the perform primitive and corresponds to a piece of code to be potentially sent to other peers and remotely evaluated. It may be understood as a generalization of a method invocation; hence it has two components:

the actual parameters that are values, produced by the local evaluation of some expression. Since these values will be sent to other peers, they cannot be/include references to the private part. This is guaranteed if, for instance, their type is a data type (including the linguistic denotation of peer, group and resource identities).

the body that is a "piece of code" that can be remotely executed. To avoid the well-known problems with logical mobility of code including global references, we do not allow in this part any reference but those to the formal parameters (that will be replaced as usual by the actual parameters before sending the code) and to the target resource, that is by definition collocated with the execution of the body.

The language used to write the action must provide means to represent the resources that it will access, both a nominal style (that is, some linguistic construct to access the resource address/identity) and the property oriented style (that is, some notation to select all the resources satisfying a boolean condition in a given resource community).

The middleware is also the unique responsible for group management, in the sense that joining and leaving groups, finding the members, or 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.

A well-known problem often encountered, in mobile environments, is how to take care of unannounced disconnections, due to the movement of the device hosting the peer out of range. Following the approach in [Arora *et al.* 2002; Mascolo *et al.* 2002; Demers *et al.* 1994], we assume that the middleware masks this kind of disconnections, by mechanisms like caching on each peer a view of the groups and reconciliating the copies after a temporary and involuntary disconnection. That corresponds to abstractly viewing the groups as realized on an everywhere available connection net, trusting the middleware to solve the problems due to this idealization. Therefore, our paradigm is suitable to design distributed applications that *survive* in a mobile environment so far that most of the disconnections are announced or very short in time (so that the divergent evolutions of the copies are not problematic), but it is not a good choice for those applications where mobility in a hard environment, where connections are seldom available, is the central issue. A consequence of our assumption that unannounced disconnections are masked by the middleware is that each group is a connected graph at any given instant.

2.2 Mapping the Example into the P2P Paradigm Architecture

In this section we present the first step to realize the order handling application ORDERS on a P2P distributed architecture, by showing how the ORDERS-PIM, defined in Sect. 1.2, will be transformed into a distributed P2P system following the paradigm of Sect. 2.1. Then, that system will be precisely designed using the UML profile of Sect. 3, getting the P2P oriented ASM for the ORDERS case, named ORDERS-P2P.

To start, we need a rough description of the distributed structure of the system to design, that is which computers will be available, under the control of whom, and how they are connected by the network.

For the use of a P2P distributed paradigm to be appropriate, the hosts involved should be able to cooperate in a paritary way, and it cannot be that there is a unique host. In this case, we will also introduce some mobility aspects, to show how they are handled in the proposed approach.

Let us assume that the following information is given on the distributed architecture to be realized for the ORDERS.

DA1 Each salesman will be equipped with a portable computer.

DA2 The company is structured in different branches.

DA3 Each branch, warehouse and mail center is equipped with a computer.

DA4 Warehouses may be down from time to time, due to internal activity, as inventory, throughout cleaning, or just because out of products.[7)]

DA5 A special application for handling the company mail may be installed on each host corresponding to a mail center.

DA6 The portable computers of the salesmen can be connected to Internet by means of a modem, while all the others are assumed to be stably connected by Internet.

Using that information, we can devise the peers composing the system. In this case, because of DA1 and DA3, we have a peer for each salesman, each warehouse, each branch, and each mail center.

The next step requires to deploy the ORDERS-PIM objects of the <<boundary>> and <<executor>> classes on such peers. In this case the deployment of the <<boundary>> objects (salesmen, mail centers and warehouses) is quite obvious, because their external users own a peer each. We decide to deploy an order manager on each branch peer, in order to take advantage of their computational capabilities, since there is no reason to deploy them together with any particular boundary element, as they interact equally with all of them.

Objects of <<entity>> classes that are accessed only by objects already deployed on a unique peer, will be deployed on the same. In the ORDERS case, each instance of UnionInfo will be deployed on the peer of the corresponding salesman.

Objects of <<entity>> classes 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

[7)] The ORDERS example has been developed having in mind a big dairy company.

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 split the unique archive in several subarchives, one for each branch, that will contains the orders handled by the manager standing 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 obviously deploy Stock objects on WareHouse peers.

The groups allows to discipline the cooperation among the peers of the system. In the P2P system we are going to build, the peers should cooperate only when some active constituents (elements of <<boundary>> or <<executor>> classes) deployed on them were already cooperating within the ORDERS-PIM. The ORDERS-PIM precisely depicts such cooperations by means of the <<access>> associations, drawn as thick lines in the Static View, see Fig. 2. We have *direct cooperations*, when active objects access each other, or *indirect cooperations* when different active objects access the same passive object (element of an <<entity>> class). After having determined the possible interactions, we analyse each of them by examining the other parts of the ORDERS-PIM (definition of the behaviour of active classes, Fig. 3, and of the operations of the passive one, Fig. 4) in order to figure out the groups needed to support these interactions. The results of this analysis in the ORDERS case are as follows.

C1 A branch may cooperate with any mail center (by sending invoices to be mailed);

C2 a branch may cooperate with any warehouse (by accessing its stock information);

C3 a salesman may cooperate with any branch (by accessing its order archive);

C4 a salesman may cooperate with other salesmen by issuing memos that will be received by the others, but only those belonging to its trade union, (by the definition of the issueMemo operation).

Therefore, we can devise six groups for supporting the above cooperations: Mail, Product, Company for cases C1, C2 and C3, and Union1, Union2 and Union3, for the trade unions A, B and C, with a common structure for the case C4. Now we have to analyse any of the cooperations determined above, and to realize them in the new P2P setting by designing the resources that will be shared on the corresponding groups. In general, many different solutions may be possible, additional information, e.g., on the quality aspects on such cooperations or on the timely behaviour of the external users, may help to pick the best choice.

CR1 Since, there is no particular urgency in passing the invoices prepared by the branches to the mail centers, we can realize this cooperation as follows: the branches will make available the prepared invoices on the Mail group, and the mail centers several times a day will collect them.

CR2 The warehouse peers share the stock info on the Product group and the branches access them each time they have to decide whether an order may be processed.

Since many different warehouses may be used to fulfill an order (all those handling the location of the client and having enough product), a branch to avoid looking up all the warehouses when handling each order, keeps the identity of the last warehouse used to serve an order for a location (denoted by its zip). It looks for a new one, whenever the selected one either is not working or has not enough product.

CR3 The branches will make their order archives available on the Company group to the salesmen. The salesmen connected to Company will add the new orders on the archive of one of the available branches.

Salesmen may have also to browse all the order archives to check the status of old orders, and that may be done without problems when they are connected to the same group Company.

CR4 Salesmen are frequently disconnected and so when a memo is issued it cannot be directly delivered to all the members of the union of the issuer. On the other side, there is no particular urgency and relevance on delivering them, thus there is not the need to use a (stable) peer to host them; instead we will us the typical P2P technique of replicating the data on several peers to cope with unavailability. The issued memo will be made available by the sender on the group of his union. A salesman may, from time to time, look for unread memos and copy them into its private part, and also into the public part of his union group. In this way, the chance of the other salesmen to read the memos are increased. To detect the unread memos, a salesman keep the date of the most recent memo received by any other salesman.

As a final step, all the calls of the operations of the objects that are now resources shared on some group (deployed on the same on an a different peer), 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 a visual object-oriented notation to model the P2P oriented ASMs following the paradigm illustrated in the previous section. Technically, we will present such notation as a UML profile.

3.1 Generalities

A *UML profile* [OMG 1999] is a standardized choice of a subset of the UML constructs, together with extensions, constraints and semantic specializations over them, that tailor the UML to a particular platform, or application environment.

We recall that he UML offers several mechanisms to extend/specialize itself; precisely:

stereotype To define a new construct starting from an existing one, by imposing additional constraints on its use and/or by specializing its semantics. Usually, a stereotype is depicted by adding the string <<stereotype name>> over the original icon, but it is also possible to define a new icon for it (e.g., datatypes are special classes whose objects cannot change their states and where two objects with the same state are the same).

tagged value To associate to an existing construct additional information, as a pair tag (a keyword) and value of some type (e.g., to add to a statechart the author, as the pair "author" = string);

constraint To restrict the possible use of some constructs (e.g., statecharts may be associated only with active classes).

specialization of the semantics To specialize the semantics (clearly informal) of some parts of UML, named semantic variation points, which is is not fixed, but for which several possibilities are given (e.g., to assume that the event queue is handled in a pure FIFO way).

Since we integrate our P2P paradigm within an object-oriented paradigm, it is most natural to consider, as resources to be *shared*, standard *objects*. This choice brings into the picture a new powerful tool supporting the design of quality software, that is *typing*. Indeed, as shared resources are objects, they are naturally typed by their classes, implicitly providing a precise interface for the resources and hence a contract between providers and users. Moreover, we may use the same mechanism to classify the different peers and groups constituting the system architecture.

There are two key points in our profile. The first is that a model will consists of several (more or less standard) UML (sub)models describing, respectively, the system architecture, each kind of involved peer, and each kind of resource community shared on groups. The second point is that we represent the middleware component on each peer as an object of a class predefined by the profile, offering as operations the middleware primitives. In this way the access to the middleware primitives is uniformly disciplined by the standard mechanism of operation call.

3.2 Illustrating the UML-P2P Profile

In this section, we introduce the UML-P2P profile by showing the structure of its models, with the help of ORDERS-P2P, the ASM for the ORDERS case study.

3.2.1 Overall Structure

We use the UML to model the architecture of the system as a whole, the resources shared on the groups of each type, and the part of the system resident on the peers of each type.

The architecture of the system is described

- at the static level by the *P2P Static View* that presents the used types for the peers and groups and the capabilities for peers of a given type of being member of groups of a given type;
- at the instance level by the *Architecture Diagram* that states the actual instances of the peer and group types building the system and the memberships among them.

The resources of the system shared on the groups of a type are described by a *Resource Community Model.* 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 are described by a *Peer Model.*

Moreover, the consistency among these models 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.

3.2.2 P2P Static View

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

In a P2P Static View only classes that are of the stereotype <<peer>> or <<group>> and associations that are of stereotype <<member>> may be used.

<<peer>> and <<group>> are classes without attributes and operations, visually represented respectively by the box and the oval icons.

<<member>> are oriented associations going from a <<peer>> class into a <<group>> class, where the multiplicity on the group end is always 1 (and thus it is omitted). In this way the association names allow to uniquely identify the groups a peer 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. <<member>> associations are visually depicted by thick lines.

In Sect. 2.2 we determined the peers and the groups of our P2P realization of ORDERS. The peers may be classified in four types: Salesman, Branch, MailCenter and WareHouse, each one with several instances, while the groups are of three types: Company, Product and Mail, with a unique instance, and Union with three instances. Such peer and group types are summarized in Fig. 6.

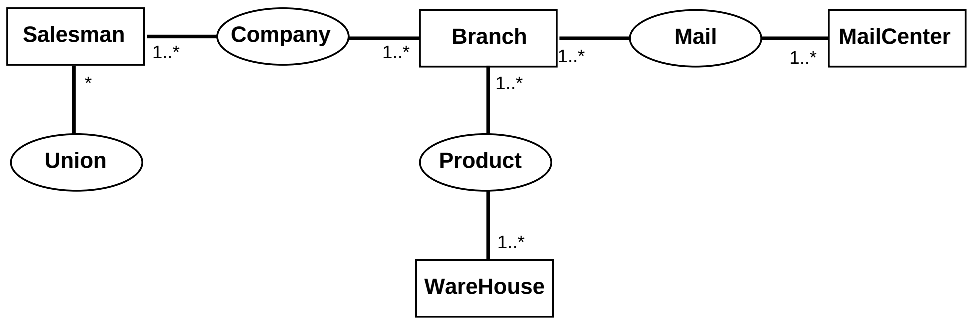


Figure 6: ORDERS-P2P: P2P Static View

3.2.3 Architecture Diagram

The Architecture Diagram describes the structure of the modelled P2P system by saying 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 peer and group 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, depict the groups to whom the various peers belong. Each of them is named with the corresponding association name (no ambiguity, due to the restrictions on the multiplicity of the <<member>> associations).
- We further assume that all peers and groups composing the modelled system must appear in the Architecture Diagram. If their number 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. Thus, we can express precisely the architecture of the system by means of this diagram.

The Architecture Diagram of the ORDERS-P2P (see Fig. 7) shows that there is a unique group of type Company, Product and Mail that all peers of appropriate types belong to, and three groups of type Union, and moreover that each salesman belongs exactly to one of them (that corresponding to his trade union). In this case, we have exactly 98 mail centers, and any the number of the members of the three groups of Union type may be whatever.

3.2.4 Resource Community Model

A Resource Community Model describes the types of the resources shared by the members of any group of a given type. Hence, in a UML-P2P model there will be a Resource Community Model for each group type present in the P2P Static View.

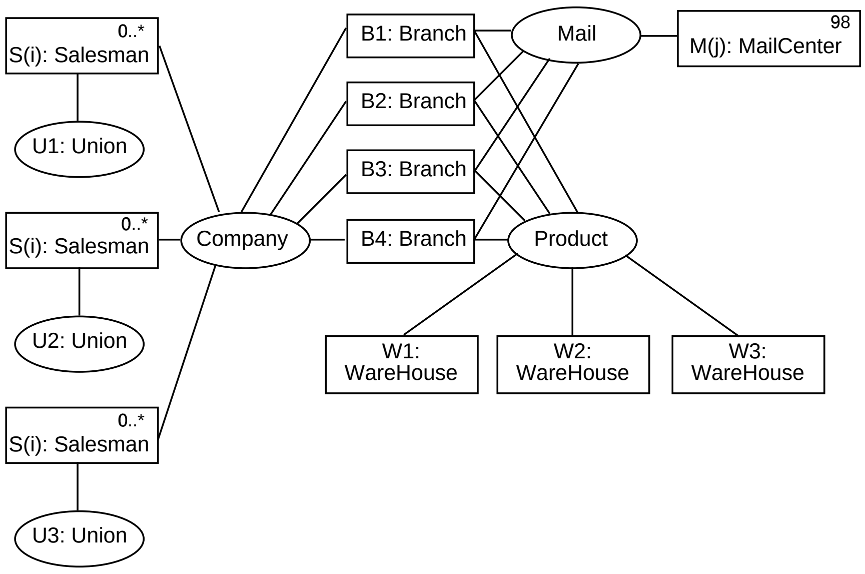


Figure 7: ORDERS-P2P: Architecture Diagram

A Resource Community Model for a group type simply consists of a standard UML package named as the group type itself containing at least a class diagram; however the new special OCL basic types PeerId, GroupId and ResourceId may be used. These types are a syntactic representative of peer, group and resources identities to be used as arguments and results of the middleware primitives. They are used as a bridge among the different UML models composing a UML-P2P model. Following the intuition that the resources are manipulated by the peer internal activity through the middleware, no calls of the middleware primitives are allowed within a group model.

The models of the resource communities of the ORDERS-P2P are reported in Fig. 8, and all of them are quite simple, just showing the shared data, defined as in the ORDERS-PIM.

3.2.5 The Middleware

We introduce the middleware in our profile by defining a UML interface, P2PMW, whose operations bijectively 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, but 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 (pre-post conditions, activity diagrams, ...) 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

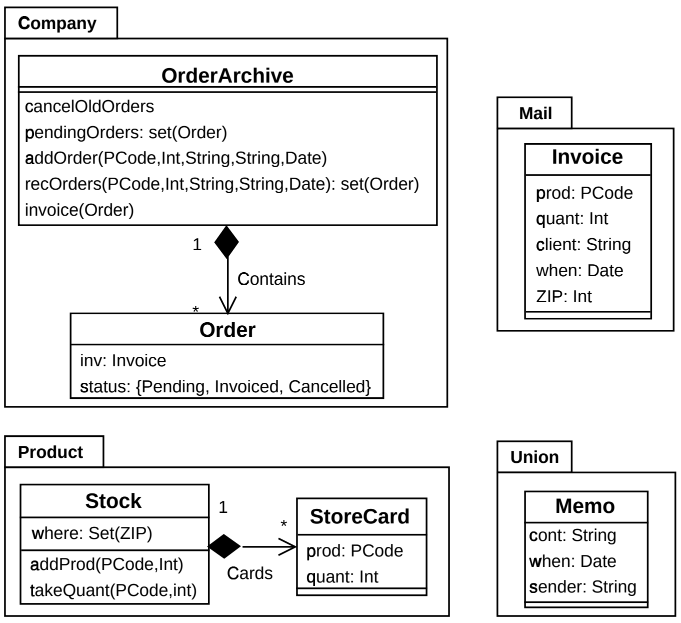


Figure 8: ORDERS-P2P Resource Community Models

on the current activities of the middleware itself.

In Fig. 9[8] we present the interface P2PMW.

The middleware primitives use the special types PeerId, GroupId and ResourceId, already introduced, representing identification of peers, groups and resources respectively, 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), consisting in adding two new actions:

[8] The UML types OclType and OclAny correspond respectively to the set of all types of a UML model and to the type of generic objects.

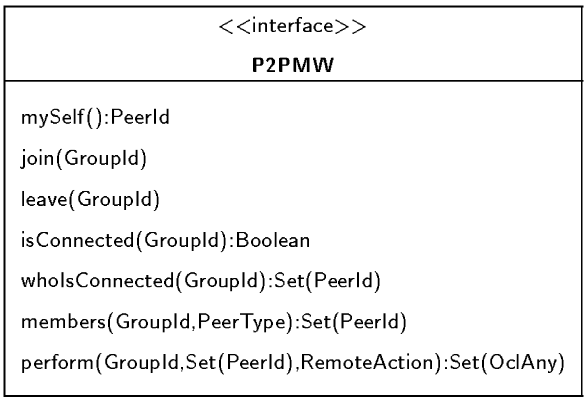

<<interface>>
P2PMW
mySelf():PeerId
join(GroupId)
leave(GroupId)
isConnected(GroupId):Boolean
whoIsConnected(GroupId):Set(PeerId)
members(GroupId,PeerType):Set(PeerId)
perform(GroupId,Set(PeerId),RemoteAction):Set(OclAny)

Figure 9: P2PMW: The Middleware Interface

returnCopy that will make in the private community of the calling peer a deep copy of the objects taken as arguments and return their identities,

returnRef that will give back the identifiers (element of the special type ResourceId) of the arguments. Such resource identifiers, 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; indeed, an action may be used to define the body of a method. To describe a query or updating on some particular resources in a selected 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. Notice that the creation of new resources may be obtained by using the create action, without acting on some existing resources.

Note that RemoteAction elements must be statically correct in the context of the communities on which then will operate, in particular no references to the caller environment may appear in them. This 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 returns the identifier of the peer, where the middleware component is resident.

join connects to the given group.

leave disconnects from the given group.

isConnected checks if there is an active connection to the given group.

whoIsConnected returns the set of the identifiers of the peers currently connected to the given group.

members given a group identifier g and peer type PT, returns the set of the identifiers of the peers of type PT that are members of the given group. 9)

9) 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 other peer referencing.

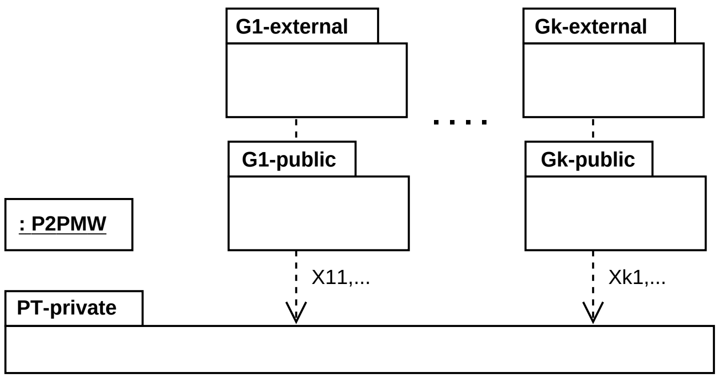


Figure 10: Structure of a Peer Model

perform given a group identifier g, a set of peer identifiers, ps, and a remote action ra, executes ra in all the public communities for group g of the peers in ps. Then, it collects the results of any returnCopy and returnRef actions, producing a, possibly empty, object collection and yields it as result.

Two particular choices for instantiating the peer set parameter that are used quite often are "all the members currently connected", corresponding to an activity on all the group resources, and "myself", corresponding to the access to the caller own public part.

3.2.6 Peer Model

A Peer Model describes the software required by the modelled 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[10] consists of the following parts, see Fig. 10:

- :P2PMW, a denotation of the used middleware component, that is a *ClassifierRole* for the interface defined in Sect. 3.2.5, 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 public resource community, which will be made available to any group of type GT to whom the peers of type PT belongs.
  - 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 to whom they belongs.

[10] That is, the peer type PT is associated to group types G1, ..., Gk by <<member>> associations in the P2P Static View diagram.

Each of the two packages for group type GT defines a resource community by a class diagram that is a subset of the one of the GT Resource Community Model, stating the types of the resources belonging to the community, and possibly by additional constraints on the resources actually building the community, e.g., stating that there are at least two resources of some type, or that there is at most one resource of some type having some property.

If any of these two packages is empty (because the peers either 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 of whom it is member (determined by the names of the <<member>> associations in the P2P Static View). Such parameters are depicted over the lines connecting the packages corresponding to their types.

  The private part may be 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. The calls to the middleware primitives must satisfy the following static correctness requirements. A peer may only join (and leave) a group of which it is member, determined by the P2P Static View diagram. 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: SalesmanPeer Model

Here we illustrate the use of the Peer Models on the example of the Salesman peer type, see Fig. 12, emphasizing the methodological aspects. The other Peer Models of the ORDERS-P2P example can be found in Appendix A.

In Sect. 2.2 we decided to deploy one object of the <<boundary>> class Salesman and the associated object of <<entity>> class UnionInfo on each salesman peer; thus we put the corresponding classes in the private part of the Salesman peer model (see Fig. 12). Notice that such classes have been slightly modified in order to accommodate the P2P realization of their interactions (see CR3 and CR4 of Sect. 2.2). Below, we detail such changes.

The behaviour of the new Salesman' is in Fig. 13.

First, because the portable computer of the salesmen may be put off and on we have added two operations (start and close) to start and to end its activity, which take care to join and to leave the groups the peer belongs to (see Fig. 13).

method addMemos()

```
{ ms = perform(Union,any,returnCopy Memo.allInstances->select(M.when>[[self.lastMemos]].lastDate(M.sender);
  perform(Union,mySelf,makeCopy(ms));
  self.Knows = self.Knows ->union(ms);
  self.lastMemos = update(self.lastMemos,ms); }
```

Figure 11: Definition of the operation addMemos

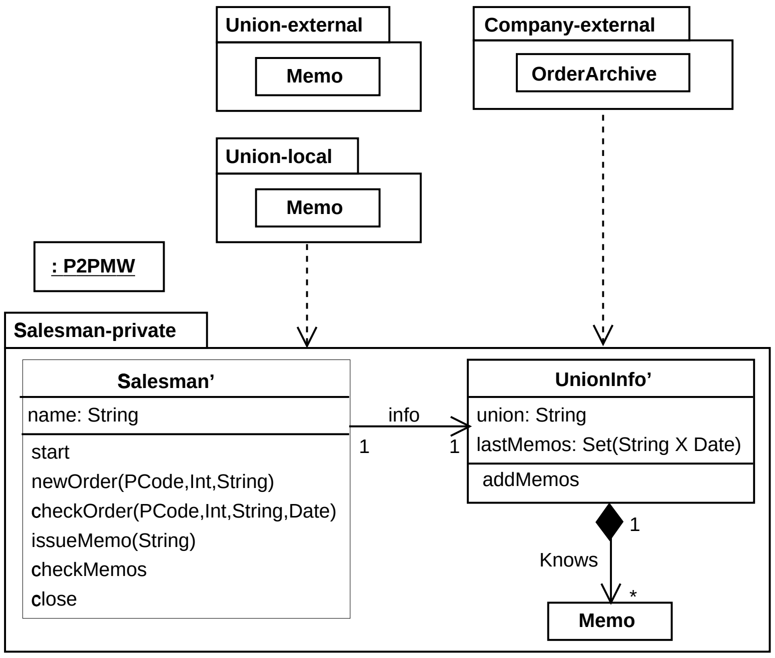


Figure 12: Salesman Peer Model

To realize the exchange of the union memos (CR4) Salesman has the new operation checkMemos, which activates the recovering of the memos from the peers of members of his union. The effect of receiving a checkMemoscall (see Fig. 13) is to call the new operation of UnionInfo addMemos (defined in Fig. 11). addMemos after having recovered the unread memos, which are determined by using the new attribute lastMemos (the date of the last memo received by each union member) copies them in the private part of the peer. Then, it makes public copies of the found memos on the Union group, to help other salesmen to get them, whenever the sender is not connected, and update the value of lastMemos. As a consequence, the packages Union-external and Union-public contain the class Memo because the salesman peers access memos on the community of the other peers members of the group Union, and make public the found ones.

To handle new orders and to recover the status of the old ones (CR3), the salesman peers access the order archives on the group Company, thus the package Company-external contains the class OrderArchive.

Notice that in Fig. 11 and Fig. 13 some expressions appearing as arguments of perform are enclosed by

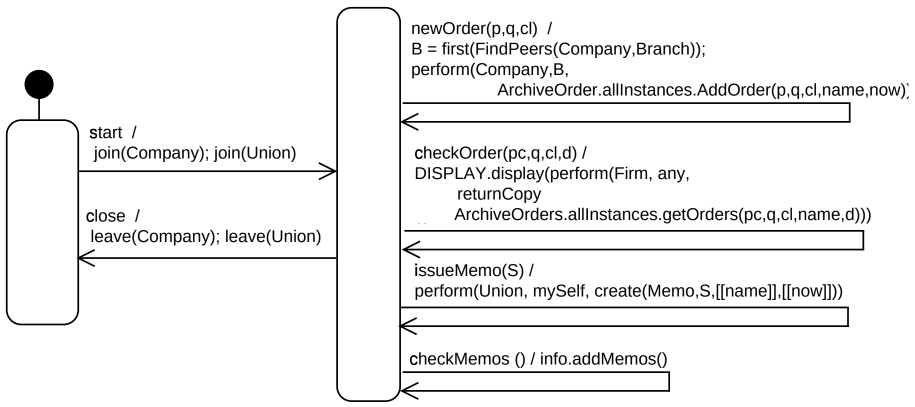

newOrder(p,q,cl) /
B = first(FindPeers(Company,Branch));
perform(Company,B,
ArchiveOrder.allInstances.AddOrder(p,q,cl,name,now))
start /
join(Company); join(Union)
close /
leave(Company); leave(Union)
checkOrder(pc,q,cl,d) /
DISPLAY.display(perform(Firm, any,
returnCopy
ArchiveOrders.allInstances.getOrders(pc,q,cl,name,d)))
issueMemo(S) /
perform(Union, mySelf, create(Memo,S,[[name]],[[now]]))
checkMemos () / info.addMemos()

Figure 13: Salesman Peer Model: Behaviour of the class Salesman'

double square brackets. This is just a notational facility provided by the UML-P2P profile for expressing the remote action actual parameters, while the formal ones are implicit, (see Sect. 2.1). The enclosed expression will be evaluated by the caller, that is an object of the private community, instead than being remotely evaluated on some resource object belonging to some public community. That notational facility is provided by the UML-P2P profile to allow in the remote action part some subexpressions that have to be evaluated in the caller environment before moving the action to the possibly remote object for execution. The only restrictions are that such subexpressions are enclosed by [[double square brackets]] and that their type must be a datatype. Technically, that just corresponds to replacing in each call of the middleware primitives the local subexpression with another expression with the same value, say V, but without any reference to the caller state. If the type of the expression is a predefined type this is trivial; instead if it is a user defined datatype, then it is sufficient to create a copy of V.

Notice in the two pictures Fig. 11 and Fig. 13 the OCL operation .allInstances refers to those instances in the community determined by the enclosing call of perform, precisely all the public communities of peers of type Salesman connected to the group union.

3.2.7 Static correctness of the overall model

Together 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. First of all the GT-public and GT-external packages for a group type GT from all the Peer Models of the possible members should be checked 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. To verify this condition we need not only the GT-public and GT-external packages for any given group type GT from all the Peer Models of the peer types whose instances may be members of the instances of G, but also the Architecture Diagram. Indeed, if some resource has to be shared on some group between two peers, first of all the resource class should appear in the GT-public package of the peer type of the provider and in the GT-external package of the peer type of the consumer. But, for the sharing to take place, the two peers have to be connected on the *same* group, while the GT-public and GT-external are at the group type level and do not concern the instance level. 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, RELATED AND FUTURE WORK

The use of a middleware, encapsulating the treatment of distribution and mobility and providing appropriate abstractions for handling them in a transparent way, improves and simplifies the development and maintenance of such applications. From the experience of some projects dealing with significant case studies, we have been impressed by the gap existing between the proposed rigorous software development techniques and the use of the various kinds of middleware in the practice. The gap has been noted also in the literature ([Emmerich 2000]), where the use of middleware-oriented methods in software engineering has been advocated.

We have worked out along that suggestion a methodological proposal for P2P middlewares in the context of the MDA (for *Model Driven Architecture*), a technique proposed by the OMG, that advocates the initial use of a *Platform Independent Model*, or PIM, designed following the natural structure of the system to be described, to be refined into one or more *Platform Specific Models*, or PSM, that will be adapted or even completely replaced accordingly to the frequent changes in the technology. To take into account the increasing number of proposals of P2P middlewares, we have proposed the introduction of an intermediate level, called ASM for *Architecture Specific Model.* Since our proposal is made in the context of the MDA and due to the relevance and success of the UML, technically the ASM level has been presented by a UML-*profile*.

More specifically, this paper has presented the first steps to support an MDA oriented process to develop P2P distributed systems. Indeed, first we have presented a UML profile, UML-P2P, to express the intermediate P2P architecture oriented models (ASM), and shown on an example how to transform a system described by a PIM into a P2P oriented ASM described by a UML-P2P model.

The middleware is naturally integrated into the OO paradigm of UML, describing its software components present on 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 modelling applications using some middleware. Another possible benefit of this approach is that we can give a full UML description of the class of such object, that will play for the profile user the role of a reference manual, in the very language the user is familiar with.

Admittedly our work is an experimental first attempt, in the very recent spirit of the MDA. There are a number of related directions to investigate, that we outline to put our work in a wider perspective.

An important orthogonal addition to our work would be to extend the UML-P2P to cover the case of P2P architecture that can dynamically reconfigure itself, i.e., where peers and groups may be dynamically created and destroyed, and where also the set of the group members may change (i.e., a peer may become member of other groups or may leave a group).

Then we need to define in the general case how to map a system designed by a PIM onto a P2P distributed architecture by transforming the PIM into an ASM presented by using UML-P2P. In this paper, we have just shown how to handle the case of the ORDERS application. The mapping will be given by a set of systematic guidelines.

In another direction, 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 [Cugola and Picco 2001], Jxta [Sun-Mycrosystem 2000], xmiddle [Mascolo *et al.* 2002] etcetera. 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 the particular middleware: group structure, kind of resources and their operations, the way it support the access to the resources (by name or anonymously), the offered services.

In the literature there are already proposals of UML profiles for supporting the development of distributed systems, see e.g., [Astesiano and Reggio 2001] for the case of using tuple spaces as coordination mechanism. Other attempts, in the literature, at developing UML notations for supporting the use of a specific middleware concern CORBA; currently an official proposal is under discussion at the OMG, see [OMG 2000]. The main difference of [OMG 2000] with our approach is that it gives a UML way to define the interfaces written using CORBA IDL, whereas the other aspects of the system should be modelled as usual when using the UML. We think that, if we try to build a UML profile for CORBA in the same way we have built UML-P2P, likely we would get a more abstract view of the "CORBA components" and a notation taking care of the design of the whole system. Indeed, we would have to find out

- the primitives provided by the middleware;

- a precise but quite abstract description of their arguments;
- a precise but quite abstract description of the network as perceived by the middleware users (perhaps a world of objects/components realizing the Corba IDL interfaces).

For what concerns the methods for supporting the development of P2P distributed systems a recent interesting approach has been proposed [Kirda *et al.* 2002; Dustdar and Gall 2002] in the more specific field of the applications for supporting the cooperative work. They suggest to introduce an intermediate layer between the middleware and the application, providing the most common services needed by the cooperative applications, as authentication, messaging, and user management. Such services, defined on the basis of those provided by the underlying middleware, will be used to realize the application. To see the relationships to our work we have to go back to the MDA proposal, see Sect. 1.1, we can see that it includes the so called "pervasive services" that are essential services, which should be available to any application. At the moment MDA has just very briefly introduced the idea, and proposes that pervasive services should be defined in a standard way independently from any platform, and thus at the PIM level. In our more refined setting, including the intermediate architecture-oriented models, we think that we should surely define such pervasive services at the architecture level, thus at the ASM level, because any kind of architecture needs/relies on particular services. We have to further investigate whether those architecture-oriented pervasive services can completely replace the platform independent ones, or can just help standardize and support the reuse at the architecture-oriented level. We plan to add to our P2P-oriented layer the definition of a set of appropriate relevant services, such as those of [Kirda *et al.* 2002; Dustdar and Gall 2002], which seem quite general and covering the most common aspects of P2P mobile applications (for example, user management, event publish and subscribe). Technically, such services would be introduced in the UML-P2P profile as predefined packages offering definitions of standard resources (e.g., events), of group types (e.g., a community of users) and specializations of the interface of the objects representing the middleware components (extension with new operations corresponding to access the standard resources and/or using the standard groups). However, such services should be derived, in the sense that they can be defined by using the basic notation of UML-P2P, that is by using our abstract P2P middleware, similarly to [Kirda *et al.* 2002; Dustdar and Gall 2002] that shows how to realize their services for cooperative work by using the underlying chosen middleware. That approach would also guarantee that a mapping from an UML-P2P model into a model in a profile for a specific middleware can be applied also to a model that uses those pervasive services.

Acknowledgments

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

REFERENCES

Arora, A., C. Haywood, and K. Pabla (2002), "JXTA for J2ME^{TM}Extending the Reach of Wireless With JXTA Technology," Technical report, Sun Microsystems, Inc., Available at `http://www.jxta.org/project/www/docs/JXTA4J2ME.pdf`.

Astesiano, E. and G. Reggio (2001), "UML-SPACES: A UML Profile for Distributed Systems Coordinated Via Tuple Spaces," In *Proc. ISADS 2001*, IEEE Computer Society Press, Available at `ftp://ftp.disi.unige.it/person/ReggioG/ AstesianoReggio00a.pdf`.

Balzarotti, D., C. Ghezzi, and M. Monga (2002), "Supporting configuration management for virtual workgroups in a peer-to-peer setting," In *Proc. SEKE 2002*, ACM Press.

Charles, J. (1999), "Middleware Moves to the Forefront," *Computer 32*, 5, 17–19.

Cugola, G. and G. P. Picco (2001), "PeerWare: Core Middleware Support for Peer-to-Peer and Mobile Systems," Manuscript, submitted for publication.

Demers, A., K. Peterson, M. Spreitzer, D. Terry, M. Theimer, and B. Welch (1994), "The Bayou Architecture: Support for Data Sharing amoung Mobile Users," Technical report, Xerox Parc, Santa Cruz, CA, US.

Dustdar, S. and H. Gall (2002), "Architectural Concerns in Distributed and Mobile Collaborative Systems," In *Proc. SEKE 2002*, ACM Press.

Emmerich, W. (2000), "Software Engineering and Middleware: A Roadmap," In *The Future of Software Engineering*, A. Finkelstein, Ed., ACM Press, pp. 117–129.

Intel (2002), "Intel Philantropic Peer-to-Peer Program," WWW site `http://intel.com/cure/`.

Jatelite-System (2002), "Jatelite White Paper," Available at `http://www.jatelite.com/pdf/jatelite_en_whitepaper.pdf`.

Kirda, E., P. Fenkan, G. Reif, and H. Gall (2002), "A Service Architecturr for Mobile Teamwork," In *Proc. SEKE 2002*, ACM Press.

Kortuem, G., J. Schneider, D. Preuitt, T. Thompson, and Z. S. S. Fickas (2002), "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.

Mascolo, C., L. Capra, S. Zachariadis, and W. Emmerich (2002), "XMIDDLE: A Data-Sharing Middleware for Mobile Computing," *Wireless Personal Communications 21*, 77–103.

OMG Architecture Board MDA Drafting Team (2001), "Model Driven Architecture (MDA)," Available at `http://cgi.omg.org/docs/ormsc/01-07-01.pdf`.

Murphy, A., G. Picco, and G.-C. Roman (2000), "Developing Mobile Computing Applications with Lime," In *Proceedings of the 22th International Conference on Software Engineering (ICSE 2000), Limerick (Ireland)*, M. Jazayeri and A. Wolf, Eds., ACM Press, pp. 766–769.

OMG (1999), "White paper on the Profile Mechanism – Version 1.0," Available at http://uml.shl.com/u2wg/default.htm.

OMG (2000), "UML Profile for CORBA, Version 1.0," Available at http://www.omg.org/cgi-bin/doc?ad/00-02-02.pdf.

SETI@home (2002), "The Search for Extraterrestrial Intelligence," WWW site http://setiathome.ssl.berkeley.edu/.

Siegel, J. and the OMG Staff Strategy Group (2001), "Developing in OMG's Model-Driven Architecture (MDA)," Available at ftp://ftp.omg.org/pub/docs/omg/01-12-01.pdf.

Sun-Mycrosystem (2000), "Jxta Initiative," WWW site http://www.jxta.org/.

Technologies, P. (2002), "PeerReview," WWW site http://www.porivo.com/peerReview/.

Traversat, B., M. Abdelaziz, M. Duigou, J.-C. Hugly, E. Pouyoul, and B. Yeager (2002), "Project JXTA Virtual Network," Technical report, Sun Microsystems, Inc., Avalaibe at http://www.jxta.org/project/www/docs/JXTAprotocols.pdf.

Xerox-Parc (1996), "The Bayou Project," WWW site http://www2.parc.com/csl/projects/bayou/.

A Peer ModelS OF THE ORDERS-P2P EXAMPLE

MailCenter

The peers of type MailCenter are quite simple, in their private part there is just an active object of class MActivity, corresponding to the boundary element of the ORDERS-PIM, that periodically collects the invoices provided by members of the group Mail and forward to the paper mail system. Its peer model is shown in Fig. 14 together with a state chart describing the behaviour of the class MActivity.

Branch

The private part of the the branch peers consists of an active object of class OrderManager, corresponding to the executor element of the ORDERS-PIM, that periodically invoices the pending orders for whom there is available products, and cancels the orders that have been pending for more than three months. Moreover, to avoids to access the network to find a working warehouse (that it is then connected to the Productgroup) for serving clients resident in some area (denoted by its ZIP) each time an order is processed, it keeps the reference of a connected warehouse (of the corresponding peer) for each ZIP into an object of class WareHouses, and periodically updates them.

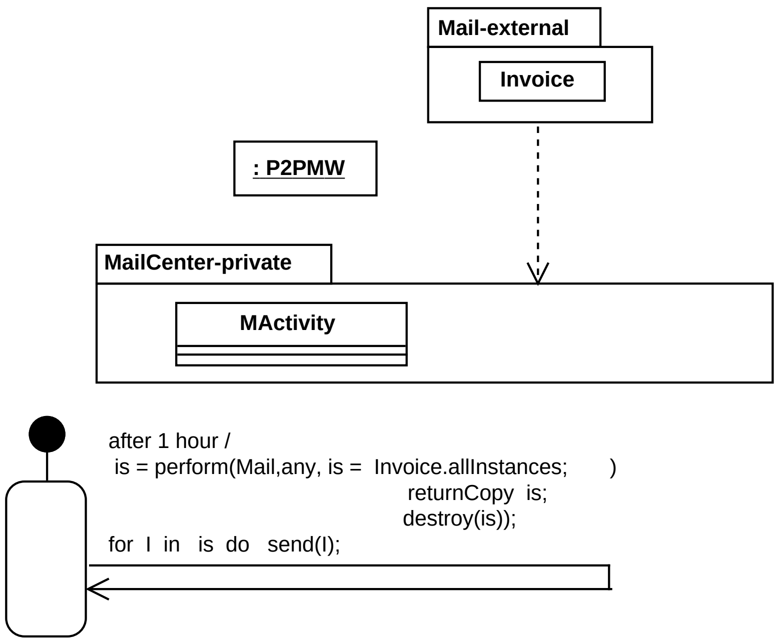


Figure 14: MailCenter Peer Model and behaviour of class MActivity

The class OrderManagerhas two new auxiliary methods with respect its definition in the ORDERS-PIM, they have been introduce to present in a clear ways its behaviour.

The methods of the classes appearing in the private part of Branchare reported in Fig. 15.

WareHouse

The private software resident on the warehouse peers, see Fig. 18, just takes care of updating the stock information, whenever some quantity of products are received.

```
method lookForWarehouse(O)
  { ws = findPeers(Product,WareHouse);
    { ws1 = ws->select( W | perform(Product,W,Stock.allInstances.where->includes(O.Z)));
    while(ws1 <> {}and not perform(Product,first(ws1),returnCopyStock.allInstances.takeQuant(O.prod,O.quant))) do
      ws1  = ws1 - first(ws1)
      if ws1  <> {}then perform(Product,W,returnCopyStock.allInstances.takeQuant(O.prod,O.quant));
      self.register(first(ws1),O.Z) } }
method cancelOldOrders():
  { perform(Company,mySelf, OrderArchive.cancellOldOrders }
method invoices():
  { os = perform(Company,mySelf,returnRef OrderArchive.allInstances.pendingOrders());
    for O in os do
      W = whs.wareOf(O.ZIP);
      if W <> Null and isConnected(W) then
      { ok = perform(Product,W,returnCopyStock.allInstances.takeQuant(O.prod,O.quant))
      if ok then perform(Firm,mySelf,OrderArchive.allInstances.invoice(O));
      else lookForWarehouse(O); }
```

Figure 15: Definitions of the methods of the private classes of Branch

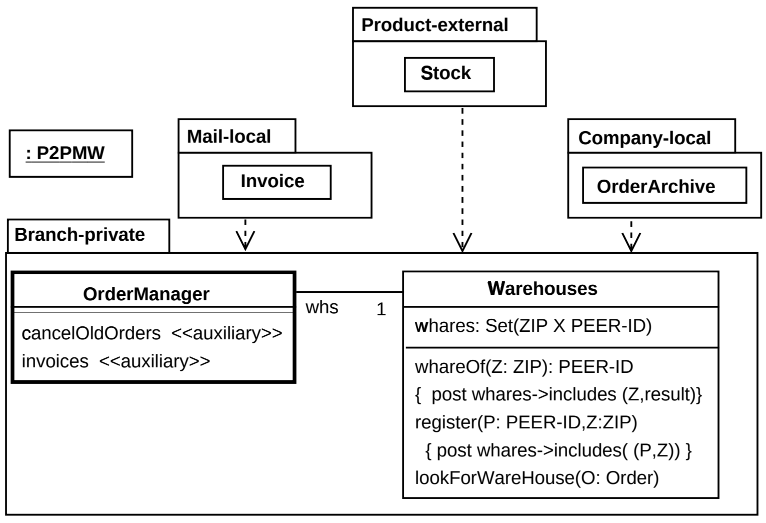

Product-external
Stock
: P2PMW
Mail-local
Invoice
Company-local
OrderArchive
Branch-private
OrderManager
cancelOldOrders <<auxiliary>>
invoices <<auxiliary>>
whs
1
Warehouses
whares: Set(ZIP X PEER-ID)
whareOf(Z: ZIP): PEER-ID
{ post whares->includes (Z,result)}
register(P: PEER-ID,Z:ZIP)
{ post whares->includes((P,Z)) }
lookForWareHouse(O: Order)

Figure 16: Branch Peer Model

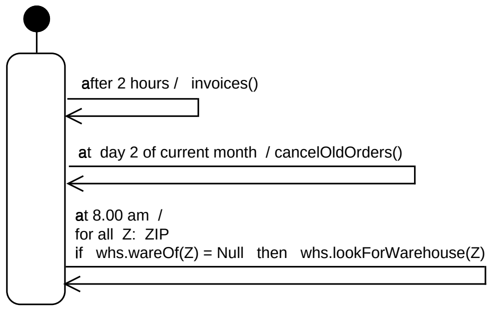

after 2 hours / invoices()
at day 2 of current month / cancelOldOrders()
at 8.00 am /
for all Z: ZIP
if whs.wareOf(Z) = Null then whs.lookForWarehouse(Z)

Figure 17: Behaviour of the class OrderManager of Branch

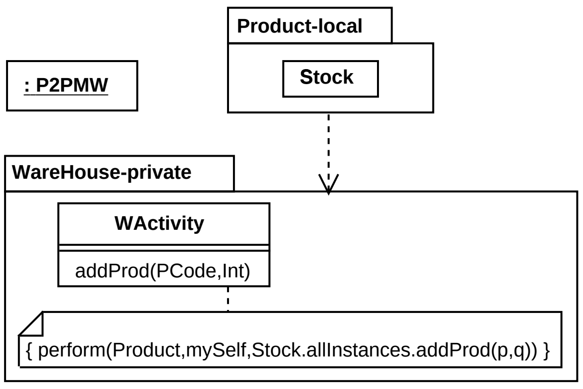


Figure 18: WareHouse Peer Model