Sequence Diagrams for Mobility[1]

Piotr Kosiuczenko

kosiucze@informatik.uni-muenchen.de
Institute of Computer Science, Ludwig-Maximilian-University
Oettingenstr. 67, Munich, Germany

Abstract. There are several kinds of UML diagrams for convenient modelling of behaviour, but these diagrams can be hardly used for modelling mobility. The situation is not very different in the case of agent languages. There exist already some proposals for modelling mobility of interacting agents by graphical notations, but these notations are rather not very intuitive and hard to read if the specification becomes a bit complex. In this paper we propose a new graphical notation for modelling interaction of mobile objects. The notation is based on UML sequence diagrams. We model behaviour of mobile objects using a generalized version of lifelines. For different kinds of actions like creating, entering or leaving a mobile object we use stereotyped messages. We provide also a zoom-out, zoom-in facility allowing us to abstract from specification details. We explain our notation in a series of examples, study its applicability and limits.

1 Introduction

The emergence of World-Wide-Web and WAN provided a qualitatively new computational infrastructure which changed our view of computing. Its emergence fostered new concepts of location like virtual location for administrative domains, fire-walls, physical location for computing devices operating in different places and so on. The Web provides rather a dynamic collection of several independent administrative domains which are very different and where the communication latency matters (cf. [9]). The computing devices differ in their power, availability and the network links differ in capacity and reliability. The network topology, which was carefully hidden in LAN, starts to play a fundamental role; it is dynamic and very complex.

There are many different concepts of computing which exploit the Web infrastructure. One of the most important is the paradigm of mobile computing which gains more and more interest. Code mobility emerged in some scripting languages for controlling network applications like Tcl. There are agent languages like Telescript and place based languages like Linda. Agents mobility has been supported by Telescript, AgentTcl or Odyssey (cf. e.g. [9]). Mobile hosts like laptops, WAPs or PDAs can move between networks. Entire networks can be mobile too like for example the IBM's Personal Area Network, networks of sensors in airplanes or trains. Here the administrative barriers and multiple access pathways interact in very complex ways. Mobile computations can cross barriers and move be-

1. This research has been partially sponsored by the IST project Architectures for Mobility funded by the European Commission as a part of the Global Computing Initiative.

tween virtual and physical locations, they can turn remote calls to local calls avoiding the latency limits. There exist several formalisms and some notations for modeling mobility, but the most relevant for our approach are Ambient Calculus [5], Maude [12], Agent UML [2] (see below). There exists still a discrepancy between these formalisms and on the other hand the graphical modeling languages capable of specifying mobility. One of the major advantages of UML [13] is its expressiveness. UML provides a variety of different kinds of diagrams which allows one for specification of software systems from different points of view. UML proved to be very useful in describing various aspects of behavior, but at the moment offers limited support for modeling mobility. In particular its sequence diagrams can be hardly used to specify even simple cases (see section 3). Similarly, the graphical languages for agent systems capable of specifying mobility become very hard to understand when the complexity increases.

In this paper we present **S**equence **D**iagrams for **M**obility (**SDM**), an extension of UML sequence diagrams for modeling mobile objects. The idea is similar to the idea of ambients or Maude, in that a mobile object can migrate from one host to another and it can be a host for other mobile objects. It may interact with other objects. Like a place, a mobile object can host other mobile objects, it can locally communicate and receive messages from other places. Objects can be arbitrarily nested, generalizing the limited place-agent nesting of most agent and place languages. To model nested and dynamically changing structure, we generalize the concept of object lifeline of UML sequence diagrams. First, we blow up the lifeline to an action box which now plays the function of lifeline and of object boundary. Second we stretch the lifeline to contain also the paths of objects migrating from one host to another. Our idea generalizes the idea of Use Case Maps [4, 1] and it allows us to specify ambients with their nested structure and mobility, in particular moving objects are treated the same way as their hosts (i.e. the hosts can be mobile too, cf. [9]). We provide also the possibility to abstract from certain details of SDM if they are unnecessary for a view, i.e. we develop the concept of zoom-in and zoom-out view, where certain details may be shown or abstracted away if not necessary for the description. The qualitatively new ideas in our approach are:

- the blowing up of object's lifeline and message arrows to model not only the communication but also the changing topology and in particular mobility
- the new concept of lifeline, resulting in equal treatment of all object independently whether they migrate, perform computations or host other objects
- the zoom-in, zoom-out modelling facility

The paper is organized as follows. In section 2, we present related work. In section 3, we specify a very simple behavior of a mobile object using UML sequence diagrams; then we gradually introduce the basic concepts of our notation. In section 4, we present more advanced concepts; we discuss the problem of identifying mobile object across complex lifelines and present a unification algorithm for that. Finally in section 5, we consider a bit more complex example trying to push our graphical notation to its limits.

2 Related Work

There exist several formalisms and some notations for modeling and specification of mobility. We mention here the most relevant for our approach. A very interesting formal notation is provided by the Ambient Calculus [5]. In this formalism, on one hand the ambients are playing the role of physical or logical locations and on the other hand they are playing the role of processes. The ambients can move around entering or leaving other ambients and performing computations. The topology can be explicitly observed and constraints the communication and mobility. The ambients barriers model security constraints like those provided by fire-walls. This calculus is based on local synchronous communication and mobility. The advantage of this calculus is that it provides nice abstractions for modeling mobility across nested locations and allows one to specify security constraints.

One of the earliest formal notation capable of specifying mobile objects was Maude [12], although specification of mobility was not its primary goal. Maude is a very flexible formalism for specifying complex communication patterns where synchronous as well as asynchronous communication is supported and where hierarchical object structures are allowed, like in ambient calculus. Mobile Maude is an agent language extending Maude for specification of mobile computation [6]. It uses a reflection mechanism to obtain a declarative mobile language design.

UML [13] gained wide acceptance as a modeling language. There exist already several proposals for extending UML to model new artifacts. There is an extension, called Agent UML, for modeling agents and their interactions protocols [2, 3]. Class and interaction diagrams are extended for specification of complex agent interaction protocols. There are facilities including agents roles, multi-threaded lifelines, extended message semantics and various kinds of protocols. Packages are used to express nested protocols. In particular, sequence diagrams are extended by some constructs analogous to MSC inline expressions [8].

There exist two interesting extensions of UML which can be used for modeling mobility. The first one [10] is similar to an early idea of Use Case Maps, where for example the behavior of a traveler was modelled by a line going from one location to another [4, 1]. To model locations, stereotyped classes are used, object moves are modeled by stereotyped messages. This approach is well suited for the case when there are only mobile objects and static locations, but not for modeling objects like ambients which are both locations able to host other ambients and mobile devices. The second one [14] extends UML collaboration diagrams to model dynamic change of composition relationship. It was not meant to model mobility, but it can be used for this goal too. This approach is well suited for simple cases and provides very compact but hard to read specifications for larger ones, specially when the objects perform many jumps (cf. section 5). On the other hand, the duality between collaboration diagrams and sequence diagrams is not preserved and requires further research.

Let us mention an approach to three-dimensional animation of UML diagrams [7]. We have to mention also Message Sequence Charts (MSC) [8], a graphical language similar to sequence diagrams but with much more constructs included for specifications like inline expressions or High Level MSC. Unlike sequence diagrams, MSC are aimed at strictly asynchronous systems and can not model method calls.

At the moment, we do not try to provide means for specifying complex behaviors or protocols. To model the choice, one can use the proposal of [3] or the composition operation like inline expressions [9]. We do not model security issues concerning the problem of whether an object can cross a barrier or communicate with another object.

3 The SDM Language

In this section we study the possibility to model mobile systems using sequence diagrams. In subsection 3.1 we show that sequence diagrams, as specified by [13], do not suffice to model mobile systems. In subsection 3.2 we describe the basic ideas of Sequence Diagrams for Mobility (SDM), an extension of the sequence diagrams.

3.1 Modeling in Sequence Diagrams

Let us consider a simple example. A passenger `ps` boards an airplane `ap` at Warsaw airport `WAW`, flies to Munich `MUC` and then deplanes (cf. figure 1). It is not easy to model this using sequence diagrams only, since there exist no direct means for modelling change of state nor change of topology (for example the fact that a passenger is in airport and then in a plain). The state description can be contained in the box; the fact that the person is at Warsaw airport is indicated by `at = WAW`. This kind of modelling has the disadvantage, that when an object changes its state, we need a new box. Such boxes can be connected by a message arrow with stereotype `<<become>>`. Let us observe that boarding an airplane involves an airplane and a passenger, but here boarding is modelled indirectly as the change of state of a passenger. To disallow a plane to fly without a passenger we need `notice()` message to inform the plane that the passenger boarded.

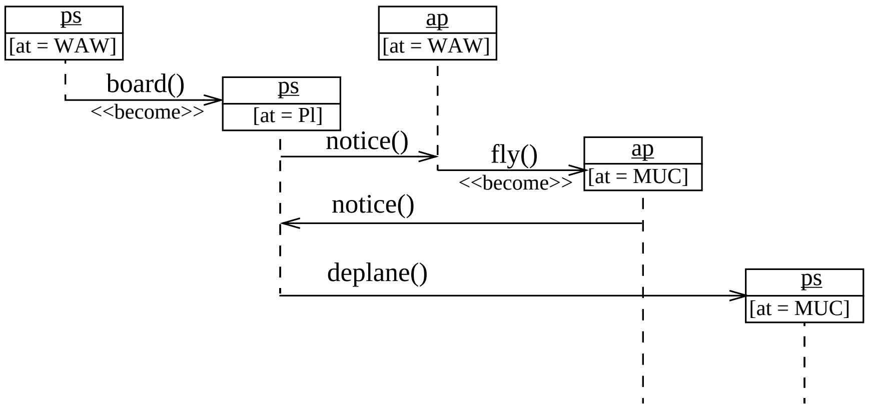

Fig. 1. Boarding flying and deplaning

As we have seen, UML sequence diagrams can model mobility in a rather very indirect way and even in this simple case are rather hard to read. There are also other possibilities to model this but they do not yield readable specifications either. Therefore in the following we introduce a new kind of sequence diagrams which are well suited for modelling mobility.

3.2 Basic features of SDM

Mobility is the ability to cross barriers. Mobile objects may interact with other mobile objects, by sending messages and changing locations. In our approach, a mobile object is also a location where interaction may happen. Different locations are separated by action boxes. The action boxes describe what happens inside and what outside and allow one to show in a transparent way message exchange and object's migration. Locations can be arbitrarily nested and form a tree structure, this is aimed at modeling firewalls, administrative domains networks and so on. For example, a personal area network may be located in a car located in a ferry which may enter a harbor and so on.

For modeling mobile objects, we use a stereotyped class `<<mobile>>` and introduce a new type of sequence diagrams. We follow the principle that sequence diagrams describe scenarios or certain system runs. Our notation allows us for specification at the level of observable traces but not on the level of simulation or bisimulation. In the UML community, there is a briskly discussion concerning the causal and temporal ordering of events in interaction diagrams (cf. [11]). We do not want to take position on that matter, while designing this notation we assumed that the causal or temporal ordering of the events is implied only by the lifelines and message flow; the fact that one object is depicted below another object does not have any meaning. This interpretation allows us for descriptions which are independent of particular observer and compress in one diagram different possible observations.

In ambient calculus communication across a single barrier is synchronous; communication across multiple barriers is performed via other ambients which navigate from one location to another. In UML but also in Maude, objects can communicate in synchronous or asynchronous way. We stick to this principle. Unlike ambients, in our notation it is possible to express actions at a distance (like RMI) even if many barriers are involved, so that multiple steps can be rendered atomic. In general, we do not want to restrict the language artificially; if something is easy to specify in our notation, then we allow it without bothering whether it is easy to implement or not. But of course if necessary one can define a dialect disallowing some expressions.

A mobile object can change its location in a jump action. For example an object may cross a firewall in a message; in this case the topology changes too (cf. [9]). To model this, object lifelines in sequence diagrams are blown up to action boxes; it models actions performed by a mobile object and indicates the boundaries of the object. Consequently, in our two dimensional representation we have two lines which denote the same thread. This implies that different arrows must be attached to different levels of an action box. Unfortunately, we can not use here dashed lines for the action boxes as in sequence diagrams, since the pictures become blured specially when the complexity increases. To avoid the visual clatter we use continuous lines.

A description of a mobile object's behavior starts with a box containing optionally the object name or class. A mobile object may jump into another object, or jump out of an object. If an object jumps into or out of another object, then the action box ends in the former location and the object is moved to another location. This move is indicated by a stereotyped message arrow which starts with a black circle; we call it jump arrow. We use here notation siilar UML state machines to indicate that after the jump the moving object starts its operation in a new location. A mobile object can not continue its operation outside of its new host, if it is already inside another host; consequently the arrow starts strictly at the end of the first action box to indicate that all action in the box must precede the jump. We assume, that the mobile objects can not be bi-located or merged, therefore an object box may have at most one jump arrow attached to the top and at most one arrow attached to the bottom. If a mobile object starts its operation (and was not active before anywhere else), then this is indicated by a special box like in the case of sequence diagrams. If a mobile object was already active somewhere else, then there must exist a jump arrow such that its sharp end is attached to the left or right upper corner of the corresponding action box (see figure 2). This requirement corresponds to the fact that mobile objects can not be merged, nor appear out of nowhere. An action box of an object which already performed a jump may optionally start with the objects name and/or class. The name is mandatory, if its lack would lead to an ambiguity (see subsection 4.3). We indicate the end of mobile object description by two horizontal lines, where the upper line is dashed. Let us point out that it does not mean that the object was terminated (see below).

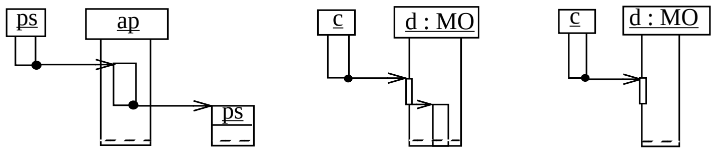

Fig. 2. Object mobility

Figure 2 shows what a mobile object looks like. As in the case of sequence diagrams, the object's names must be underlined. In the left hand side of figure 2, passenger `ps` enters airplane `ap`. Since there is no conflict concerning the identity of objects inside `ap`. The corresponding action box does not bear any name. Then `ps` deplanes `ap` and starts its operation outside `ap`, the name in the action box is not necessary either, since the identity of `ps` can be uniquely traced. No message arrow is attached to the corresponding action box except of the jump. In the middle of figure 2, a mobile object `c` enters object `d` of class `MO` by activating an operation of `d` (like a virus which sends itself in an e-mail). After the operation is finished the objects starts to operate inside `d`. In the right hand side, another scenario is shown, the object `c` does not manage to cross the barrier and disappears. We do not indicate it in any special way.

If an object sends a message to another object, then a message arrow must start at the sender's action box (at its left or right side) and must end at the receiver's action box. Figure 3 shows two communicating objects `a1` and `a2` inside the object `a`. The objects `a1` and `a2` reside inside `a` from the very beginning.

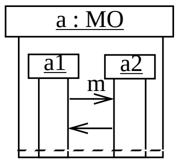

Fig. 3. Communication

Figure 4 shows an object `b` which creates a new object `c`. We use here a message arrow with stereotype `<<create>>`. Similarly for cloning an object, we use a message with stereotype `<<copy>>` [13], the copy is then assumed to behave as its original would do inside the new location (cf. [2]). Let us observe that the end line of the action box of the virus and the end line of the action box of the `131` PC are different. This does not have any meaning, but allows us for better grouping of object boxes (see sections 4 and 5).

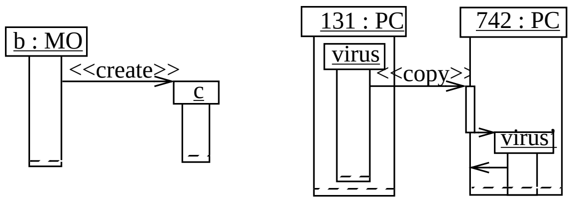

Fig. 4. Object createation and cloning

Another important operation is `open`, this operation opens an object making its hosted objects visible. If a mobile object is opened, then it ends its life, but its sub-objects continue to operate. This operation is similar to operation `open` in ambient calculus [5], but it may be synchronous as well as asynchronous, depending on the type of message used.

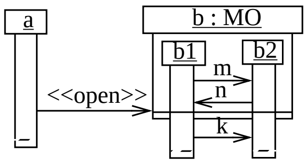

Fig. 5. Opening an object

In figure 5, we have shown an object `b`, which is opened by object `a`. The opening of an object is indicated by a horizontal line. Object `a` sends message `open` to `b`, then `b` is opened and the hosted objects `b1` and `b2` cease to operate. A mobile object can be also terminated, in this case all its hosted objects are terminated too, it can be of course expressed by a series of `open` operations. The recursive termination is indicated by a continuous line. For the recursive termination caused by an other object we use a message with stereotype `<<destroy>>` (cf. [13]). The SDM diagrams can model immediate open (cf. figure 5) and destroy (see the left hand side of figure 6) as well as cases where the opened or destroyed objects continue to operate, in this case the sharp end of the message arrow is placed above the termination line.

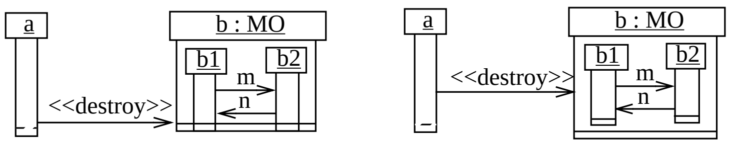

Fig. 6. Destroy

In figure 6, object `a` terminates object `b`. After terminating `b`, all its sub-objects are terminated too. The termination is indicated by a continuous line stretching across all objects. The left hand side of the figure shows immediate termination and the right hand side shows asynchronous termination.

4 Advanced Concepts

In this section we present some more advanced concepts and study the topology of nested objects more carefully. In the first subsection, we introduce the concept of zoom-in and zoom out view. In the second subsection, we define the notion of lifeline in general terms and give an example of a complex lifeline. In the third subsection, we present an algorithm for naming action boxes in complex lifelines. This algorithm can be used to figure out an object's identity along complex lifelines.

4.1 The Zoom-in and Zoom-out View

In this subsection we show how to zoom into and zoom out of objects to see or to abstract from the internal details, respectively. The left hand side of figure 7 shows communicating mobile object `c`, in the zoom-in view, which displays the hosted object `c1` with its topological details. The zoom-out view of `c` does not show its sub-objects but only its external communication. The right hand side of figure 7 shows once more the `open` operation performed by object `a` on object `b`. In this case the two sub-objects emerge, this is indicated by the fork operation which is analogous to the fork operation of state machines (cf. [13]).

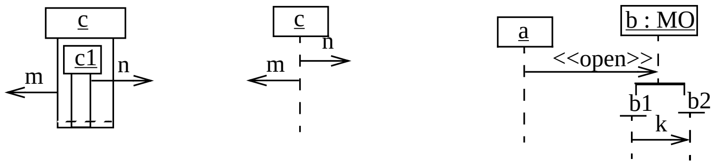

Fig. 7. Zoom-in and zoom-out

Let us observe that in the case of high parallelism zooming out may yield a more complex diagram, especially when the diagram contains a hierarchy of nested and in parallel operating objects. In such a case not only parallel forked lifelines for the emerging traces may be necessary, but also more powerful forms of combining lifelines such those provided by MSC inline expressions [8].

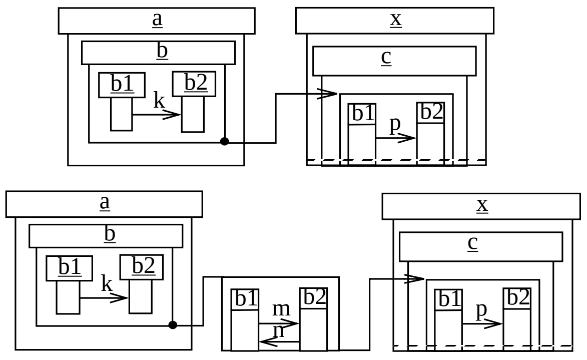

Fig. 8. Zooming into a message

It is possible also to zoom into an object's jump arrow to see the behavior of the participating objects. Figure 8 shows object b which, together with its hosted objects, jumps into object c contained in object x. In the top of the figure, the jump is shown in the zoom-out view. Below we show the zoom-in view of the jump. It displays the communication between b1 and b2. The zoom-in version of this arrow has only one black circle and one sharp end. We introduce this notation in order to make explicit that the communication happens between start of the jump and the end of the jump. Let us observe that it is not equivalent to the combination of jump out of a and jump into x, which would mean that the communication took place outside a and x (see section 5).

The possibility to mix the views is very convenient, since a specifier may chose the appropriate level of details displaying or abstracting from details. Figure 11 shows an example, where we abstract from the details of a flight which are not visible for an observer (see section 5).

4.2 Lifelines

The topology of nested objects changes during objects life. Therefore, we have to trace objects along performed jumps. An object's lifeline starts there where in the diagram the object appears for the first. The lifelines contains all jump arrows of the object and its hosts. We have to consider also the jumps of hosts, since the object may move with its hosts. The lifeline ends there where the object's description ends or there where the object terminates.

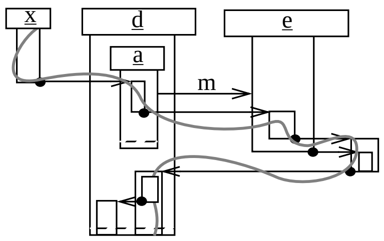

Fig. 9. Object's lifeline

Figure 9 shows objects performing complex jumps. The object x jumps into the object a which is already in d. Then a communicates this fact to e and afterwords x leaves a (and d) and jumps into e. It jumps out of e, then e jumps into x. x jumps into d, and then e jumps out of x. The curve shows the lifeline of object x.

4.3 Matching Action Boxes

To figure out an object's lifeline, it is necessary to trace the object's identity. In order to do this, we have to match nested action boxes before and after a jump. If the zoom-in view of the arrow is concerned, we have to consider also the action boxes before the jump, in the beginning of the jump as well as at the end of the jump and after the jump. The matching must be consistent along all object's lifelines. It may prove hard since the hosted objects may be rearranged and moved without changing the semantics.

On the graphical level, we may perform graphical manipulation by rearranging the boxes so that the boxes match. This matching can be formalized using pattern matching. The tree structure of terms mirror well the tree structure of complex objects. We formalize the snapshots of the mobile objects by terms, so the object's states before and after a jump will be modeled by two terms. We use the function symbol

$$f : \text{Names} \times \text{Objects} \longrightarrow \text{Objects}$$

to model mobile objects. Each object is assumed to have a name and may contain other objects. To compose objects we use the commutative and associative operation (cf. [12])

$$* : \text{Objects} \times \text{Objects} \longrightarrow \text{Objects}$$

To model the fact that an object does not contain any other object we use the constant

$$\varepsilon : \longrightarrow \text{Objects}.$$

If an action box does not have name, then we use a variable for the unknown name.

For example, the state of object c before it performs a jump can be formalized by the term $f(c, f(e, \varepsilon) * f(d, f(z, \varepsilon)))$ (see figure 10)). The action box corresponding to object x after the first jump is formalized by the term $f(X_1, f(X_2, f(X_3, \varepsilon)) * f(X_4, \varepsilon))$.

A lifeline provides a set of pairs of terms corresponding to object's states before and after a jump (like those two terms above). These pairs of terms have to be unified using the same substitution. We require that for a sequence of term pairs $t_1, t_1',..., t_n, t_n'$ there exists exactly one substitution σ such that t_i^{σ} equals $t_i'^{\sigma}$ modulo commutativity and associativity, for i = 1,..., n. The existence condition assures that a consistent naming exists and the uniqueness condition assures that naming is uniquely determined.

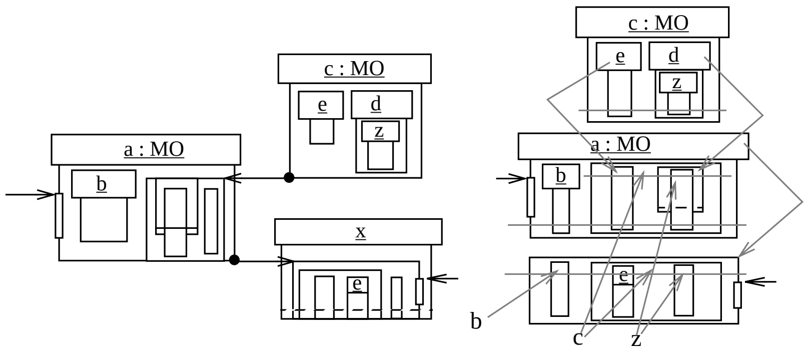

Fig. 10. Matching action boxes

Let us consider the following rather sophisticated example (see figure 10): The object c contains object e and object d which in turn contains object z. c jumps into object a, which already contains object b. Then one of the nested objects within a is opened. Further, object a, with all its sub-objects, jumps into the object x.

It is not easy to identify the object names in host x. We use the matching algorithm to identify the object's identities through this complex history.

The snapshot of the object c before jump can be formalized by the term

$$t_1 = f(c, f(e, \varepsilon) * f(d, f(z, \varepsilon)))$$

The snapshot of object a after c jumps in can be formalized by the term

$$f(a, f(b, \varepsilon) * f(X_1, f(X_2, f(X_3, \varepsilon)) * f(X_4, \varepsilon)))$$

In particular the snapshot of x after the jump is

$$t_1' = f(X_1, f(X_2, f(X_3, \varepsilon)) * f(X_4, \varepsilon))$$

The snapshot of object a before its jump is given by the term

$$t_2 = f(a, f(b, \varepsilon) * f(X_1, f(X_3, \varepsilon) * f(X_4, \varepsilon)))$$

The snapshot of object x is formalized by the term

$$f(x, f(Y_1, f(Y_2, f(Y_3, \varepsilon) * f(e, \varepsilon)) * f(Y_4, \varepsilon)))$$

The snapshot of object a after its jump is formalized by the term

$$t_2' = f(Y_1, f(Y_2, f(Y_3, \varepsilon) * f(e, \varepsilon)) * f(Y_4, \varepsilon))$$

We have to unify the terms t_1 and t_1' and at the same time the terms t_2 and t_2'. The unification is performed modulo associativity and commutativity of *. It is not hard to check that the following function unifies these terms:

$$\{X_1 \rightarrow c, X_2 \rightarrow d, X_3 \rightarrow z, X_4 \rightarrow e, Y_1 \rightarrow a, Y_2 \rightarrow c, Y_3 \rightarrow z, Y_4 \rightarrow b\}$$

The right hand side of figure 10 shows the result of the matching algorithm.

5 A More Complex Example

In this section we consider an example of a flight from Warsaw to Munich (cf. [5]) seen from two different perspectives. The first version is very simple. Then we refine this version adding several details pushing our notation to its limits. Let us point out that specifying this system using a process algebra for mobility or collaboration diagrams would be much harder.

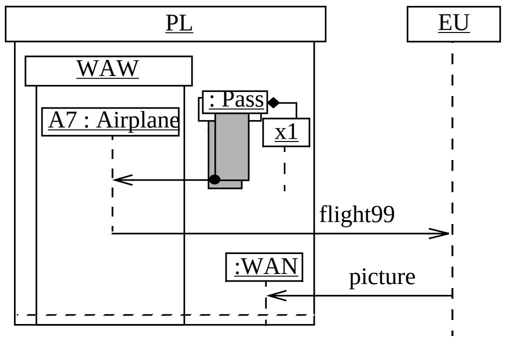

Fig. 11. Flight example

Figure 11 shows a simple story of a passenger x1 who boards an airplane in Warsaw airport, flies to Munich and publishes a picture in a WAN. This story is described from the perspective of an observer on the polish side. The person x1 together with other passengers enters the airport and then boards the airplane A7. The airplane flies to Munich (the flight number is 99), but the only thing the observer can see is that the airplane is airborne but not what happens inside the airplane nor further details of this flight. The next event which the observer is able to notice is the appearance of a picture in the WAN. To model several passengers (i.e. objects of class Pass), we use the multi-object notation [13], which allows us to present in a compact way several passengers playing the same role. Person x1 is distinguished using composition relationship. The observer does not care about the order in which the passengers board or leave the lane and what they do during the flight. We abstracted here from the architecture of WAN and the person's possession.

This simple view shows some of the barriers person `x1` has to cross while flying from Warsaw to Munich. There are political boundaries which regulate the movement of people and devices, like airplanes, computers and so on. Within those boundaries, there are other boundaries like those protecting airports and single airplanes against intruders. Only people with appropriate passports and tickets may cross those boundaries. Therefore, in our model we make explicit those boundaries and moving across them.

In the view presented in figure 11, we have abstracted from several details. The view of passenger `x1` is much more detailed. He can see that the airplane A7 is a very active mobile computing environment, full of people who are talking, working with their laptops, calling their families, making pictures or connecting to Web via phones/modems provided in the airplane.

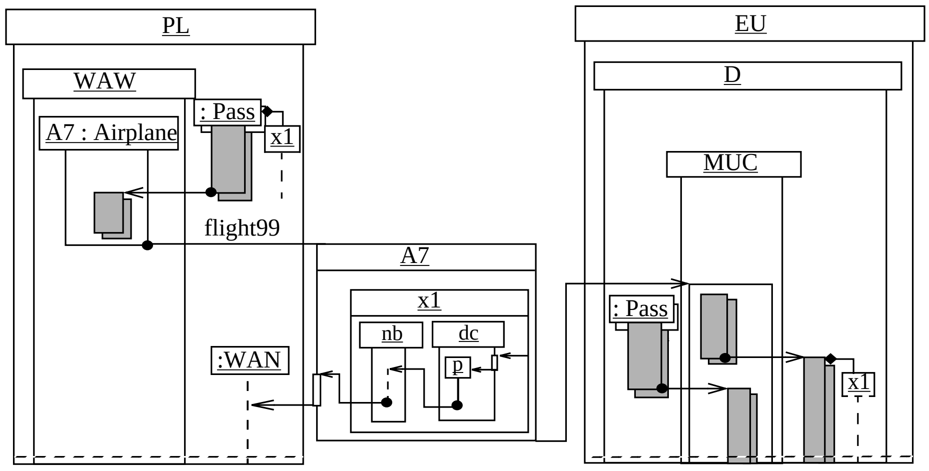

Fig. 12. Flight details

Figure 12 shows refined and extended version the previous example. We can see here, what happens inside the airplane during the flight; the jump arrow contains the action box of the airplane `A7`. Passenger `x1` makes pictures with his digital camera, the pictures are send then to the WAN. As usual, a digital camera does not allows him to send pictures directly to WAN. It is also forbidden to use mobile phones during the flight. Therefore the passenger saves the pictures to his notebook `nb`, logs into the onboard network and then transmits the pictures to WAN via the onboard network. We abstract here from the structure of the WAN network (indicated by dashed line). Let us point out that the sending of the picture by passenger `x1` is not temporally related to crossing any border like those over PL, EU and so on. The only thing we can say is that it happens between the start of the airplane and its landing. Finally, all the passengers leave the airplane and the airport. The passenger can see that the airplane is boarded by new passengers. The dashed line in the head of the last box of passenger `x1` means that the story of this passenger started earlier and that the head is a not beginning of his history.

Let us observe that the airplane is empty before the new passengers board. We assume that the action boxes determine the ordering of events, in this case the jump out arrows cross only one barrier of the airplane action box. Namely, if an arrow crosses both barriers of an action box then it means that the object is not involved in the corresponding event; in this case the arrow ordering does not matter for figuring out the behavior of the object and the corresponding ordering of events. If an arrow starts or ends at a barrier, then it is meant that the object produces or receives the event, respectively. If an arrow crosses only one barrier, then this means that another object within the barriers is involved in the event, and therefore the event can be perceived by an observer of the host object. In both cases the relative ordering of such events matters.

It is very useful to have not only asynchronous communication which uses objects to transfer messages between remote objects as in the case of ambient calculus [5], but also to have synchronous messages, which in this case modell more exactly events like a phone call. The behavior presented here was a simply a sequence od events. To specify more complex behavior we would need the MSC's inline expressions or the constructs proposed in Agent UML [13]. Let us observe that this example can hardly be specified in a notation like [10] since a mobile host like the airplane plays the role of host and the role of mobile object at the same time (cf. [10]). The regions, packages or agencies can hardly be used for that purpose.

Concluding Remarks

UML provides a variety of different kinds of diagrams which allows one for specification of software systems from different points of view. It is beneficial to have the possibility to specify a software system using different kinds of diagrams. UML sequence diagrams can be hardly used to specify mobility; therefore we proposed a new graphical notation for modeling object mobility. This notation proved to be very convenient and powerful; it extends sequence diagrams in a natural way.

In the future, we are going to define a UML profile for modelling mobile systems. This profile will extend/adjust different kinds of UML diagrams. We are going to study the duality between sequence diagrams and collaboration diagrams. We plan to perform a realistic case study which will test the appropriateness of our diagrams. Finally, we are going to provide a formal semantics for SDM which will allow one for precise analysis of systems, for proving or disproving their properties and which will help us to provide a tool support.

Acknowledgment

The author wants to express cordial thanks to Martin Wirsing for stimulating this research as well as to Hubert Baumeister, Alexander Knapp, Nora Koch and the anonymous referees for their helpful remarks.

References

1. Amyot, D., Mussbacher,G.: On the Extension of UML with Use Case Maps Concepts. In: Evans, A., Kent S. (eds.). The 3rd International Conference on the Unified Modeling Language, UML 2000, LNCS 1940, Springer, Berlin, 2000.
2. Bauer, B., Müller, J., Odell, J.: An Extension of UML by Protocols for Multiagent Interaction. Proc. 4th International Conference on Multi Agent Systems. IEEE Press,

2000.
3. Bauer, B., Müller, J.: Agent UML: A Formalism for Specifying Multiagent Software Systems. International Journal of Software Engineering and Knowledge Engineering, Vol. 11(3), 2001, 207-230.
4. Buhr, R. J. A. Casselman, R.S.: Use Case Maps for Object-Oriented Systems, Prentice-Hall, USA, 1995.
5. Cardelli, L.: Mobility and Security. Bauer, F., Steinbrüggen, R. (eds.): Foundations of Secure Computation. Proc. NATO Advanced Study Institute. IOS Press, 2000, 3-37.
6. Durán, F., Eker, S., Lincoln, P., Meseguer, J.: Principles of Mobile Maude. In: Kotz, D., Mattern, F. (eds.). Agent Systems, Mobile Agents, and Applications. 2000, LNCS 1882, Springer, Berlin, 2000, 73-85.
7. Gogolla, M., Radfelder, O., Richters, M.: Towards Three-Dimensional Animation of UML Diagrams. In: France, R., Rumpe, B. (eds.): UML'99 -The Unified Modeling Language. Beyond the Standard. LNCS, Vol. 1723, Springer, Berlin, 1999, 489-502.
8. ITU-TS, 2000, Recommendation Z.120. Message Sequence Charts (MSC). ITU-TS, Geneva.
9. Jing, J., Helal, A., Elmagarmid, A.: Client-Server Computing in Mobile Environments. ACM Computing Surveys. Vol. 31(2), 1999, 117-157.
10. Klein, C., Rausch, A., Sihling, M., Wen, Z.: Extension of the Unified Modeling Language for Mobile Agents. Idea Publishing Group, 2001.
11. Knapp, A.: A Formal Semantics for UML Interactions. In France, R. and Rumpe, B. (eds.). Proc. 2'nd Int. Conf. UML, LNCS, Vol. 1723, Springer, Berlin, 1999, 116-130.
12. Meseguer, J.: Parallel Programming in Maude. In Banatre, J. le Metayer, D. (eds.): Research Directions in High-Level Parallel Programming Languages, LNCS 574, Springer, Berlin, 1992, 253-293.
13. UML-OMG. Unified Modeling Language Specification. Version 1.4, September 2001.
14. Wienberg, A., Matthes, F., Boger, M.: Modeling Dynamic Software Components in UML. In France, R., Rumpe, B. (eds.): UML'99. Proceedings of the Second International Conference. Fort Collins, USA, LNCS 1723, Springer, Berlin, 1999, 204-219.