Extending Activity Diagrams to Model Mobile Systems[⋆] [⋆⋆]

Hubert Baumeister, Nora Koch, Piotr Kosiuczenko, and Martin Wirsing

Institut für Informatik
Ludwig-Maximilians-Universität München
Oettingenstr. 67
{baumeist, kochn, kosiucze, wirsing}@informatik.uni-muenchen.de

Abstract. Mobile systems are gaining more and more importance, nevertheless the means for their specifications are still underdeveloped. Existing UML diagrams can be used to conveniently model behavior, but these diagrams can be hardly used to model mobility. In this paper we present an extension to UML class and activity diagrams to model mobile systems. We assume that mobile objects can migrate from one location to another. Locations can be nested and mobile too. We introduce stereotypes to model mobile objects, locations, and activities like moving or cloning. We introduce two notational variants of activity diagrams for modeling mobility. One variant is location centered and focuses on the topology of locations. The other one focuses on the actor responsible for an activity. We compare these two types of diagrams and define a metamodel for them.

1 Introduction

The emergence of the Word Wide Web provides new computational paradigms in which computation is distributed over the net and highly dynamic, with the network itself changing continously. The network topology, which was carefully hidden in LAN, starts to play a fundamental role. This situation fostered new concepts like virtual locations for administrative domains, fire-walls, physical locations for computing devices operating in different places. One of the most important new concepts is mobile computing which gains more and more interest. Code mobility emerged in some scripting languages for controlling network applications like Tcl and is one of the key features of the Java programming language. Agent mobility has been supported by Telescript, AgentTcl, or Odyssey (cf. e.g. [7]). In addition, hardware can be mobile too. Mobile hosts like laptops, WAPs, and PDAs can move between networks. Moreover, entire networks can be mobile, like for example IBM's Personal Area Network (PAN) and networks of sensors in airplanes or trains. For example Fig. 1 shows a person having a PAN who boards an airplane, flies from one location to another, and deplanes.

[⋆] This research has been partially sponsored by the EC 5th Framework project AGILE: Architectures for Mobility (IST-2001-32747).

[⋆⋆] Appeared in Proc. of NetObjectDays, NODe 2002, Erfurt, October 2002.

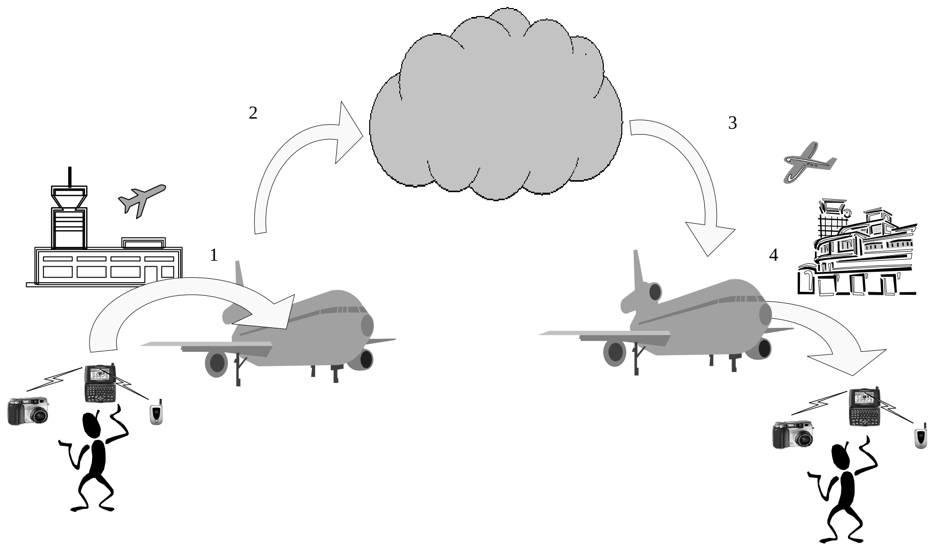

Fig. 1. Nested mobile networks.

Mobile computations can cross barriers and move between virtual and physical locations, therefore they can turn remote calls to local calls avoiding the latency limits. But there is a price to pay since the administrative barriers and multiple access pathways interact in very complex ways. This requires special means for the specification and implementation of mobile systems.

Modeling is a central activity in the software development process that leads to a clear specification and good implementation. Models are built, among others, to visualize the system architecture and to communicate the desired structure and behavior of the system. UML [13] is the standard graphical notation for modeling object-oriented software. It has the advantage of providing extension mechanisms to adapt the UML to specific domain requirements.

In this paper we present an extension of UML activity diagrams for modeling mobile systems. We introduce into UML the concepts of location, mobile object, mobile location, move action, and clone action. These concepts are defined by using UML stereotypes, tagged values, and OCL-constraints. Furthermore, we introduce two variants of activity diagrams for modeling mobility. The first variant is responsibility centered and uses swimlanes to model who is responsible for an action. The second is location centered and uses the notation of composite objects to visualize the hierarchy of locations.

The paper is organized as follows. The rest of this section discusses related work. In Sect. 2 the basic mobility concepts used in our extension are presented. In Sect. 3 we present the two variants of UML activity diagrams, followed by a formal presentation of the proposed UML profile. Finally, in Sect. 5 a conclusion and an outlook to future work is presented.

Related Work There exist several formalisms for specifying mobility, among them are Seal [15], Basic-Sail [14], and KLAIM [12]. The most relevant for our approach is, however, the ambient calculus [3]. In this formalism, ambients are playing the role of processes and physical or logical locations. Ambients can enter and leave other ambients and perform computations. The topology constrains the communication and mobility and can be explicitly specified. The advantage of this calculus is that it provides abstractions for modeling mobility across nested locations and allows one to specify security constraints.

One of the earliest formal notations capable of specifying mobile objects was Maude [10], although this was not its primary goal. It is a very flexible formalism for specifying complex communication patterns with hierarchical object structures, similar to the ambient calculus. Mobile Maude [5] is an extension of Maude to model mobile objects and processes, i.e. located, computational environments where mobile objects can reside.

These calculi are good in formalizing simple systems, but for large systems a visual notation is needed to easier grasp the specifications and to specify the system from different points of views. There exist already some proposals (cf. [16, 8, 11, 9]) to extend the UML to model mobile systems, which are complementary to our work. To our knowledge none of these extensions cover activity diagrams.

In [16] an extension of collaboration diagrams is presented to model dynamic change of the composition relationship. It defines a form of aggregation link between objects — a component link — with the additional semantics that a component link may change over time. In addition, it proposes the use of component boundaries to emphasize the relationship between a component and its immediate components. It is an interesting approach, but it does not explain how these extensions fit into the UML metamodel.

In [11] an architecture description language (ADL) for the design of mobile agent systems is presented. This ADL is defined as a UML profile. Modeling of agent migration is supported by the stereotyped flow relationship ≪become≫, an operation move, and operations beforeMove and afterMove that prepare the agent for the migration. It proposes a graphical representation using deployment and component diagrams.

Another extension is presented in [8]. It is similar to the early idea of Use Case Maps [2]. Stereotyped classes and packages are used to model mobility. Objects moving from one location to another are modeled by stereotyped messages. This approach can be used when there are only two kinds of objects: mobile objects and static locations. It is not well suited for modeling objects which are mobile and also play the role of locations.

In [9] UML sequence diagrams are extended to model complex mobility patterns. These diagrams generalize the concept of object lifeline of Use Case Maps [2]. The diagrams provide also the possibility to abstract away from irrelevant details. Their semantics is similar to that of ambients or Maude in that a mobile object is a location and a mobile process as well.

2 Mobility Concepts

In the following we introduce the main mobility concepts we use in this paper: locations, mobile objects, and actions moving mobile objects.

2.1 Locations

The concept of location plays an important role in the case of mobile systems. To denote classes whose instances are locations we use the stereotype ≪location≫. For example, the airport Charles de Gaulle (CDG) is an instance of the stereotyped class Airport. Similar to the ambient calculus [4] we allow locations to be nested (cf. Fig. 2). For example, the airport Charles de Gaulle is contained in France, an instance of class Country, which is also a location. We require that any location is contained in at most one location and that a location cannot be contained in itself (directly or indirectly). We do not require that the hierarchy of locations has a single top element. Thus the hierarchy of locations forms a forest. Note that these assumptions, in particular the assumption that a location is contained in at most one location, simplifies the semantics and in consequence the analysis of mobile systems (cf. e.g. [4]).

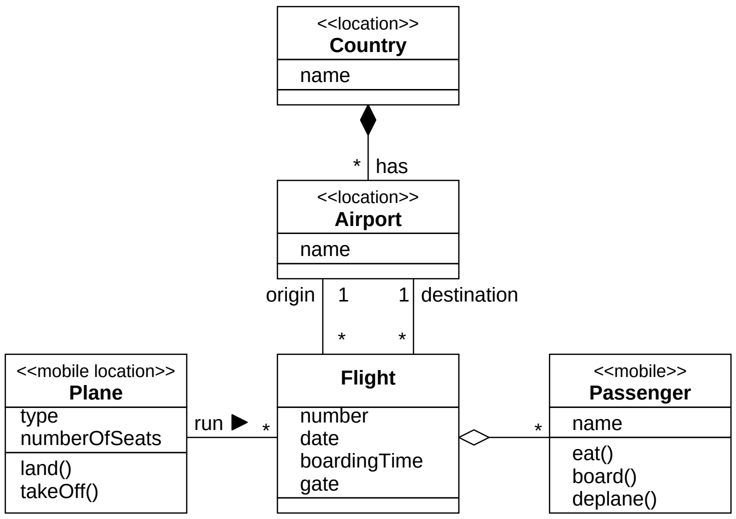

Fig. 2. A simplified class diagram modeling an airport.

2.2 Mobile Objects

A mobile object is an object that can change its location. A class representing mobile objects is indicated by the stereotype ≪mobile≫. The current location of a mobile object is indicated by the atLoc relation. As in the case of locations, a mobile object can only be contained in at most one location. In our airport example, a particular passenger is a mobile object, as he may move from one location to another, for example, from Munich to Paris (cf. Fig. 2).

Note that the atLoc relation is not explicitly presented in Fig. 2. One reason is that this would unduly complicate the diagram. For example, a passenger can be located either at a plane, an airport, or a country. The second reason is that the existence of the atLoc relation is implied by the use of the mobility stereotypes (cf. Sect. 4.1).

Locations can be mobile too. This allows us to model passengers in an airplane and flying the airplane from one airport to another. In this case the stereotype ≪mobile location≫ is used. The stereotype ≪mobile location≫ inherits from the stereotype ≪location≫ and the stereotype ≪mobile≫ for mobile objects. This was the only way to define mobile locations by stereotypes with the UML 1.3, because a model element could have only one stereotype attached to it. UML 1.4, however, makes it possible to attach more than one stereotype to a model element. In this case we could give the class Airplane the stereotypes ≪mobile≫ and ≪location≫ to denote that it is a mobile location. However, we feel that using the stereotype ≪mobile location≫ conveys better the concept of mobile locations.

For mobile locations we require that the atLoc relation inherited from mobile objects is the same as the atLoc relation inherited from locations. To ensure this, stereotypes ≪mobile≫ and ≪location≫ inherit from a common stereotype ≪spatial≫ which denotes classes of objects that can be at a location (cf. Fig. 8).

2.3 Actions

Basically, there are two primitives that change the location of a mobile object. A mobile object can move from one location to another — a so called move action; or a copy of an object is moved to a new location [6] — a so called clone action. Move and clone actions act on objects and their containment relationship wrt. locations. Given a move action on an object o which is contained in location l, i.e., $o.\mathsf{atLoc} = l$, to another location l', then after performing the move operation object o is contained in location l', i.e. $o.\mathsf{atLoc} = l'$. Clone works similar; however, instead of moving the object itself, first a copy of the object is created which is then moved to the new location.

The stereotypes ≪move≫ and ≪clone≫ for action states in activity diagrams are used to denote move actions and clone actions, respectively. Actions have two additional attributes, the first one indicates who is performing the action, and the second one is the location where the action is performed.

Calculi for mobility restrict these primitives further by omitting the clone operation. Instead, the clone operation is defined as the composition of a copy operation followed by a move operation. For notational convenience we decide to take clone as a primitive. Commonly, these calculi also restrict the target location of a move, for example, to move only one level in the containment hierarchy [4, 14].

3 Notations

In the following, we present two notations for the above mentioned mobility concepts in the context of activity diagrams. The first notation is responsibility centered and focuses on *who* is performing an action and is based on the standard notation for activity diagrams. The second notation is location centered and focuses on *where* an action is performed, given by the atLoc relation between mobile objects and locations, and how activities change this relation.

3.1 Responsibility Centered

The first notation uses object-flow states with classifier in states to model the atLoc relation. In the airport example consider a passenger Hubert who is boarding a plane at the airport of Munich. This can be modeled as a move action as shown in Fig. 3. The source of the move action is the object-flow state Hubert:Passenger [atLoc = MUC:Airport] and the target an object-flow state Hubert:Passenger [atLoc = LH123:Plane]. The passenger Hubert moves from his previous location, Munich airport (MUC), to his new location, the plane LH123. More precisely this means, if in an object configuration there is a passenger Hubert, an airplane LH123, and an airport MUC such that Hubert is contained in MUC and also LH123 is contained in MUC, the move operation changes the configuration in such a way that Hubert is no longer directly contained in the airport MUC, instead it is contained in the plane LH123. The containment of the plane does not change; therefore Hubert is still indirectly contained in MUC. Swimlanes can be used to show who is performing the action; in this case it is the passenger who boards the plane.

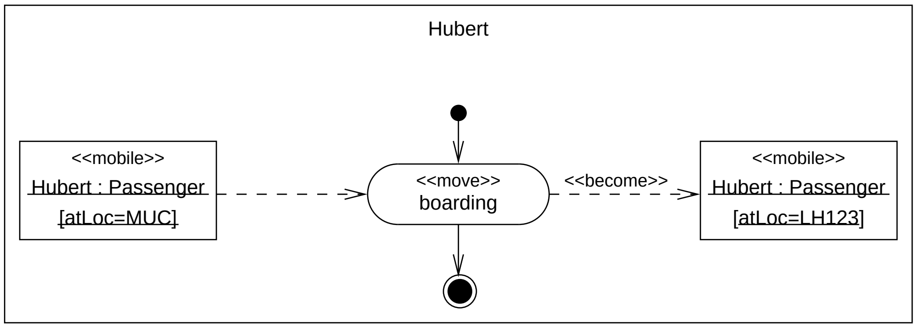

Fig. 3. The move action.

The clone operation is shown in Fig. 4 for a list of passengers (*lop*). One copy of the list is kept by the airport staff and another copy of the list is moved into the plane. The difference in the semantics of this diagram to the previous diagram is that given the configuration as before, the clone operations creates a new document list of passengers lop' which differs from lop only in the fact that it is contained in LH123. In addition lop is still contained in MUC, i.e. lop has not moved at all.

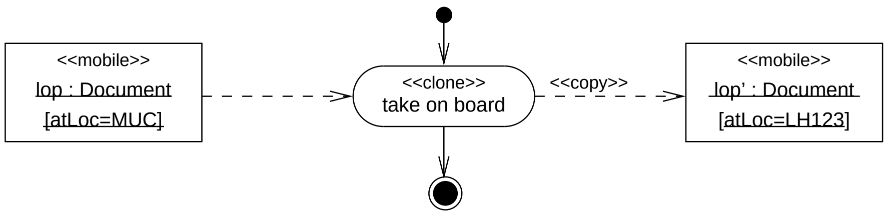

Fig. 4. The clone action.

Note that by the UML it is possible to omit the ≪become≫ stereotype in the output of the move action in Fig. 3, as it is the default that the input and the output are the same objects if the type of the object flow states are the same. In the same way, the ≪copy≫ stereotype in the output of the clone action in Fig. 4 can be omitted because this stereotype can be deduced from the stereotype ≪clone≫ of the clone action.

A more complex example is given in Fig. 5. The activity diagram starts with the boarding activity of the passenger at the Munich airport. This activity changes the location of the passenger Hubert from the airport (MUC) to the particular plane LH123. The next activity is the take-off activity of the plane. This activity changes the location of the plane from the Munich airport (MUC) to a not specified destination, that is we are not interested in the location where the plane is when it is flying. During the flight, the plane performs the flying activity and the passenger the send mail activity. These activities happen in parallel. Note that before landing, the passenger has to stop the send mail activity because the use of electronic devices is not allowed during take-off and landing. When landing, the location of the plane is changed to the destination airport, in this case the Paris airport (CDG). Finally, the passenger deplanes and is now located at the Paris airport. This notation is responsibility centered as the swimlanes are indicating who is performing a particular activity.

3.2 Location Centered

The second notation uses containment of the boxes for mobile objects/locations in the boxes of other locations to show the atLoc relation. For that we use the same UML notation as for composite objects. A difference is that the atLoc relation is not an aggregation. Another difference is that we also allow action states to be drawn inside composite objects of stereotype ≪location≫. This indicates

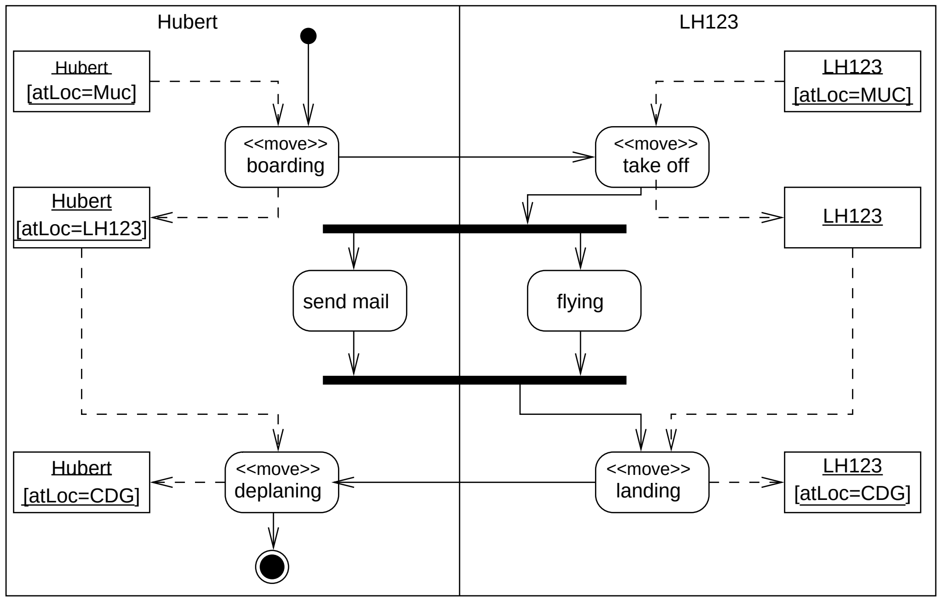

Fig. 5. The airport example using the responsibility centered notation.

that the action is performed at the corresponding location. Figure 6 shows this notation for the move operation depicted in Fig. 3.

Note, that in addition to the fact that the passenger is in the plane, we can model also that the plane is parked at the airport. This is an information that cannot be represented in the responsibility centered approach as shown in Fig. 3. What Fig. 6 also shows is that activities can be drawn inside locations to indicate that the operation is performed at that location. In the example, boarding takes place at the airport. While it is still possible to use swimlanes to indicate who is performing an action, most likely, more complex diagrams will have to concentrate on either the topology of locations or on the actor performing an activity to avoid an overloaded diagram.

Note that the box containing the airport may be omitted if this is not relevant for the presentation.

Figure 7 presents a location centered view of the activities of Fig. 5. Again, the first activity changes the location of the passenger from the airport to the plane. However, in contrast to the responsibility centered notation it is visible that the passenger is still located indirectly in the Munich airport, because the plane has not moved yet. Also one can see that the boarding activity happens at the airport. The next activity, the take-off, takes again place at the airport. In the location centered variant the notation indicates that the plane has left the airport after take-off. Again, during the flight the activities flying and send mail happen in parallel. In contrast to the information provided by the responsibility-

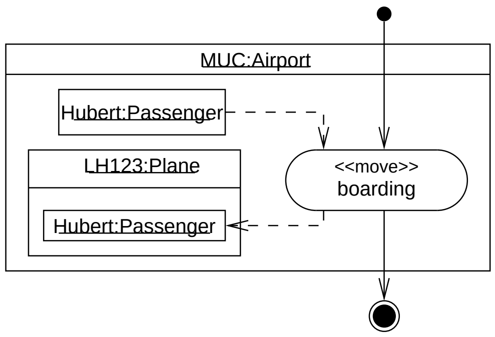

Fig. 6. The move action.

centered notation, this notation shows that the send mail activity happens *in* the plane, while flying does not take place inside the plane. Note that for simplicity reasons, the box denoting the passenger during the flight can be omitted. Landing and deplaning are similar to the activities boarding and take-off.

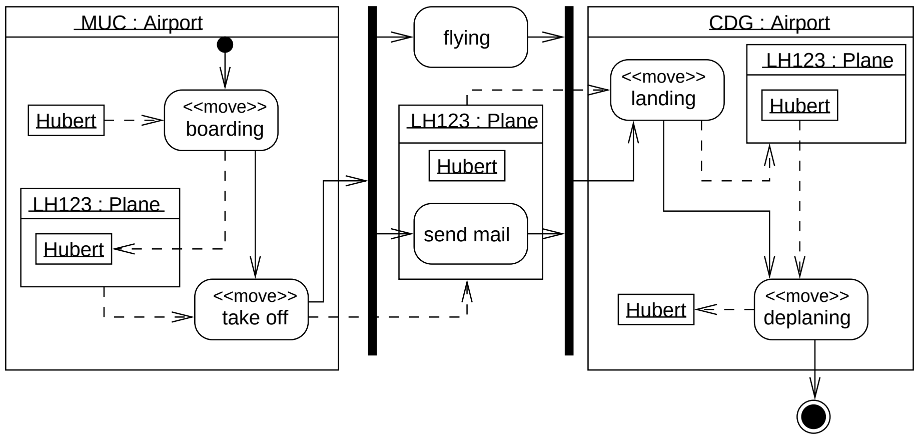

Fig. 7. The airport example using the location centered notation.

4 UML Profile

In this section we present a UML profile for modeling mobility aspects with class and activity diagrams. The profile consists of the stereotypes ≪location≫, ≪mobile≫, ≪mobile location≫, ≪spatial≫, ≪move≫, and ≪clone≫ and the tagged value where.

In this section we assume the reader to be familiar with the basic concepts of the UML metamodel and the Object Constraint Language (OCL), see [13]. We explain only those concepts of the metamodel which are important for understanding our profile for mobility aspects.

4.1 Metamodel

Figure 8 shows the metamodel for the stereotypes ≪location≫, ≪mobile≫, and ≪mobile location≫. To model the atLoc relation, we require that each class with

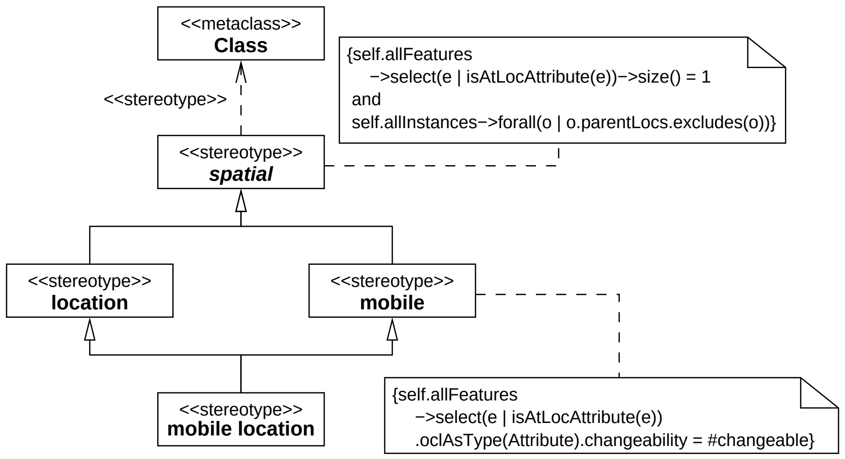
<<metaclass>>
Class
<<stereotype>>
<<stereotype>>
spatial
<<stereotype>>
location
<<stereotype>>
mobile
<<stereotype>>
mobile location
{self.allFeatures
−>select(e | isAtLocAttribute(e))−>size() = 1
and
self.allInstances−>forall(o | o.parentLocs.excludes(o))}
{self.allFeatures
−>select(e | isAtLocAttribute(e))
.oclAsType(Attribute).changeability = #changeable}

Fig. 8. Metamodel for stereotypes ≪location≫, ≪mobile≫, and ≪mobile location≫.

stereotype ≪location≫ or ≪mobile≫ provides its instances with an attribute atLoc. Since we want to state the requirement only once, we introduce the abstract stereotype ≪spatial≫ and state the requirement for that stereotype. Then the stereotypes ≪location≫ and ≪mobile≫ inherit the requirement. To express this as an OCL-constraint, we define an additional predicate isAtLocAtribute on features, i.e. instances of metaclass Feature. In the metamodel each class is associated with a set of features describing the methods and attributes of the class and its instances. A feature e is an atLoc attribute, i.e. `isAtLocAttribute`(e), if e is an instance attribute, has the name atLoc, and its multiplicity is zero or one. Further, the attribute can hold instances of classes having stereotype ≪location≫:

$$
\begin{aligned}
&\texttt{isAtLocAttribute}(e : \texttt{Feature}) = \\
&\qquad e.\texttt{oclIsKindOf}(\texttt{Attribute})\ \texttt{and} \\
&\qquad e.\texttt{name} = \text{'atLoc'}\ \texttt{and} \\
&\qquad \texttt{let}\ e' = e.\texttt{oclAsType}(\texttt{Attribute})\ \texttt{in}
\end{aligned}
$$

e'.ownerScope = #instance and
e'.multiplicity = $0..1$ and
e'.targetScope = #instance and
e'.type.oclIsKindOf(Class) and
e'.type.stereotype.name->includes('location')

Now we require that each class with stereotype ≪spatial≫ has a unique atLoc attribute and that the atLoc relation does not contain cycles:

```
self.allFeatures->select(e | isAtLocAttribute(e))->size() = 1 and
self.allInstances->forAll(o | o.parentLocs->excludes(o))
```

The additional operation parentLocs computes the set of all parent locations for an instance of a class with stereotype ≪spatial≫:

```
self.parentLocs = self.atLoc->union(self.atLoc.parentLocs)
```

For mobile objects we require in addition that they are able to change their location, which means that their atLoc attribute can change its value. This can be expressed by requiring that the changeability attribute of atLoc has the value #changeable for all classes with stereotype ≪mobile≫, in addition to the existence of an atLoc attribute — which is inherited from stereotype ≪spatial≫:

```
self.allFeatures->select(e | isAtLocAttribute(e))
                .oclAsType(Attribute).changeability = #changeable
```

The operation allFeatures is an additional operation on Classifier defined in the UML 1.4 semantics. It collects the features of a classifier together with all features of its parents.

The metamodel for move and clone action states is shown in Fig. 9. The

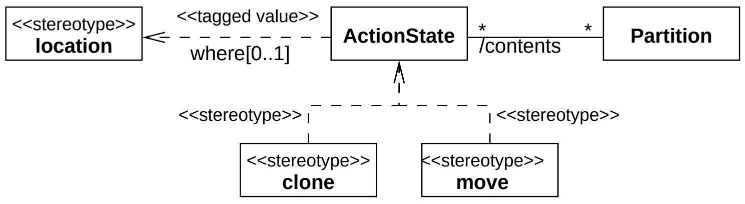

Fig. 9. Metamodel for stereotypes ≪move≫ and ≪clone≫.

association between Partition and ActionState is inherited from the association between classes Partition and ModelElement defined in the UML 1.4 semantics for activity graphs. According to the UML 1.4 semantics, a tagged value can either be associated to a stereotype — which is the preferred way — or to any model element. However, neither does it make sense to introduce a new stereotype for action states just for the purpose of adding a tagged value, nor to put the tagged value where on an arbitrary model element. Therefore we draw a dependency from ActionState with stereotype ≪tagged value≫ to indicate that we only want to apply the tagged value where to action states.

Each action state is connected via incoming and outgoing transitions to other states (cf. Fig. 10). We require that a move/clone action state has at most one incoming and outgoing transition associated to an object flow state satisfying the following conditions:

– Either the classifier of the object flow state has the stereotype ≪mobile≫ (or its subtype ≪mobile location≫)
– or, if the type of the object flow state is a classifier in state, its type has the stereotype ≪mobile≫.

In case both, an incoming and an outgoing transition, are connected to such an object flow state, the types have to be the same. This is to ensure that the type of the mobile object that is input to a move/clone action is the same as the type of the output. Note that a move/clone action may have additional non-mobile objects as input and output.

Given the current UML metamodel, the difference between move and clone actions — i.e., that the move action moves the same object while the clone action moves a copy — is not expressible without a more precise semantics of activity diagrams. This is due to the fact that an object flow state is associated to a classifier and it is therefore not possible to reference the precise object that is input or output to a move/clone action.

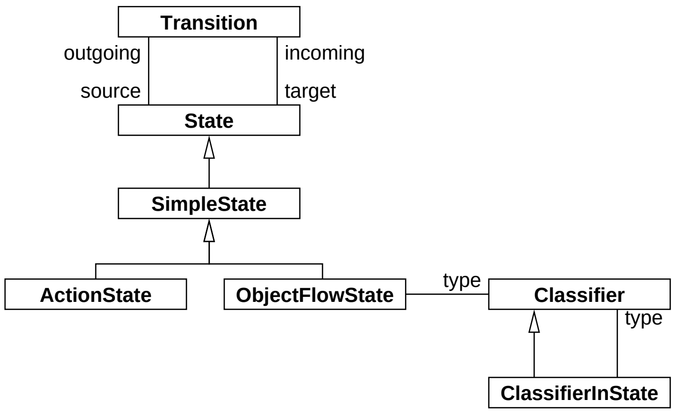

Fig. 10. An excerpt of the UML 1.4 metamodel for object flow states.

4.2 Mapping to the Metamodel

The responsibility centered notation uses standard activity diagrams, therefore can be mapped to the metamodel as described in the UML semantics (cf. [13], Sect. 3.84).

The containment of objects in the location centered notation is mapped to a constraint on the atLoc attributes. These attributes connect objects in the corresponding object-diagram describing the state of the system at a given point in time. That is, if the box of a mobile object or of a location is drawn inside a location then the atLoc attribute of that object or location has to contain the location it is in. If an action is drawn inside a composite object then the where tagged value of the action is set to the object the action is inside.

Source and target of a dashed arrow going from an object inside a composite state are mapped to corresponding object flow states. For example, if *mo* is directly contained in the composite object *loc*, and *mo* is a mobile object, *loc* a location, and *mo* is either target or source of a dashed arrow connected to a move/clone action state, then *mo* is mapped to an object flow state having as its classifier in state *mo* [atLoc = *loc*]. Figure 11 illustrates this mapping.

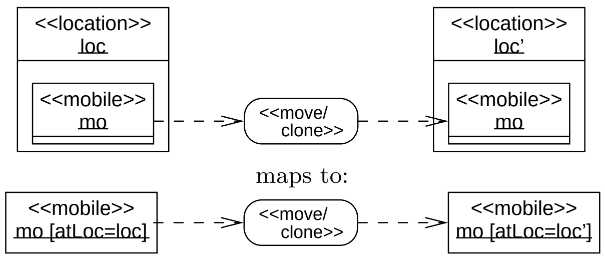

Fig. 11. Illustration of the mapping of the location centered notation.

5 Conclusion and Future Work

In this paper we have presented extensions to UML activity diagrams — a so called UML profile — to model mobile systems. We have defined stereotyped classes to model locations and mobile objects, as well as stereotyped action states to model move and clone actions.

Another approach would be to use the UML without stereotypes for the mobile concepts. In this case locations, mobile objects, and the atLoc relation could be modeled explicitly as abstract classes and associations in class diagrams and mobile objects, like Passenger and Planes, would inherit from these classes (cf. Fig. 12). However, such an approach is practical only for simple models of mobile systems. To model more complicated systems it is desirable to have the concepts for expressing mobility as part of the language — as we have done in this paper — instead of modeling these concepts explicitly every time they are used. In addition, our proposal for a common profile for mobility concepts permits the definition of a common semantics for these concepts.

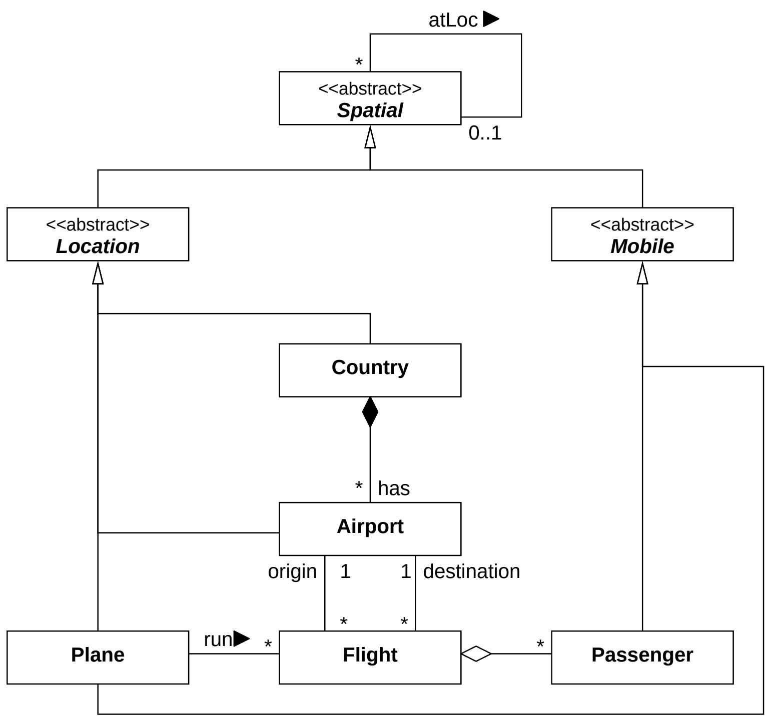

Fig. 12. The same class diagram as in Fig. 2, but modeling the mobility concepts explicitly.

Further, we have introduced two variants of activity diagrams for modeling mobility: a responsibility centered variant, focusing on the actor performing the action, and a location centered one, focusing on the topology of locations. The responsibility centered variant uses the current UML notation for activity diagrams. In contrast, the location centered variant combines activity diagrams with a notation similar to composite objects to show how move/clone actions change the containment relation between locations.

We are currently investigating the appropriateness of UML for the specification of structural and behavioral aspects of mobile systems. Our next step will be to validate the proposed notations in a bigger case study within the scope of the EU-project AGILE [1], which is part of the Global Computation Initiative. The objective of AGILE is to develop an architectural approach in which mobility aspects can be modeled explicitly. This paper is the first step towards a general profile incorporating all essential aspects of mobile systems.

We plan to develop a formal semantics for extended activity diagrams to provide a precise meaning of the presented concepts which is needed for formal analysis and reasoning about models. In addition, we plan to develop tools that support animation, early prototyping, and analysis of mobile systems.

Acknowledgments We would like to thank Alexander Knapp for helpful comments on an earlier version of this paper and Stephan Herrmann for indicating an inconsistency with the UML extension mechanism.

References

1. AGILE. Architectures for mobility. www.pst.informatik.uni-muenchen.de, 2002.
2. Raymond Buhr and Ronald Casselman. *Use Case Maps for Object-Oriented Systems.* Prentice-Hall, USA, 1995.
3. Luca Cardelli. Mobility and security. In F. Bauer and R. Steinbrüggen, editors, *Foundations of Secure Computation. Proc. NATO Advanced Study Institute*, pages 3–37. IOS Press, 2000.
4. Luca Cardelli and Andrew Gordon. Mobile ambients. In Maurice Nivat, editor, *First Conference on Foundations of Software Science and Computation Structure*, LNCS 1378, pages 140–155. Springer Verlag, March 1998.
5. Francisco Durán, Steven Eker, Patrick Lincoln, and José Meseguer. Principles of Mobile Maude. In David Kotz and Friedemann Mattern, editors, *Agent Systems, Mobile Agents, and Applications, Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents, ASA/MA 2000*, LNCS 1882, pages 73–85. Springer, 2000.
6. FIPA. FIPA agent management: Support for mobility specification. www.fipa.org, August 2001.
7. Jin Jing, Abdelsalam Helal, and Ahmed Elmagarmid. Client-server computing in mobile environments. *ACM Computing Surveys*, 31(2):117–157, 1999.
8. Cornel Klein, Andreas Rausch, Marc Sihling, and Zhaojun Wen. Extension of the Unified Modeling Language for mobile agents. In K. Siau and T. Halpin, editors, *Unified Modeling Language: Systems Analysis, Design and Development Issues*, chapter VIII. Idea Group Publishing, Hershey, PA and London, 2001.
9. Piotr Kosiuczenko. Sequence diagrams for mobility. In Stefano Spaccapietra, editor, *21 International Conference on Conceptual Modeling (ER2002)*. Springer-Verlag, October 2002. to appear.
10. José Meseguer. Research directions in high-level parallel programming languages. LNCS 574. Springer, Berlin, 1992.
11. Florin Muscutariu and Marie-Pierre Gervais. On the modeling of mobile agent-based systems. In *3rd International Workshop on Mobile Agents for Telecommunication Applications (MATA'01)*, LNCS 2164, pages 219–234. Springer Verlag, August 2001.
12. Rocco De Nicola, GianLuigi Ferrari, and Rosario Pugliese. Programming access control: The KLAIM experience. In *Conference on Concurrency Theory*, LNCS 1877. Springer Verlag, 2000.

13. OMG. Unified Modeling Language (UML), version 1.4. www.omg.org, September 2001.
14. Dirk Pattinson and Martin Wirsing. Making components move: A separation of concerns approach. In *Proc. First Internat. Symposium on Formal Methods for Components and Objects, FMCO'02, Leiden, November 2002*, LNCS, 2003. To appear.
15. Jan Vitek and Giuseppe Castagna. Towards a calculus of secure mobile computations. 1998.
16. Axel Wienberg, Florian Matthes, and Marko Boger. Modeling dynamic software components in UML. In Robert France and Bernhard Rumpe, editors, *UML'99 - The Unified Modeling Language. Proceedings*, LNCS 1723, pages 204–219. Springer-Verlag, 1999.