Partial Order Semantics of Sequence Diagrams for Mobility[1]

Piotr Kosiuczenko

Department of Computer Science, University of Leicester, Le1 7RH, UK

piotr@mcs.le.ac.uk

Abstract. There are many formalism for mobile system specification, but until very recently, there was no satisfactory graphical notation for modelling of such systems. In a previous paper, we have introduced the so-called Sequence Diagrams for Mobility (SDM), a graphical notations based on UML Sequence Diagram. This notation has been used in several case studies and proved very useful. In this paper we introduce a formal, partial order based semantics for SDM. We define the notion of run and show how to figure out the system topology from the information contained in a run. We formalize the zoom-out abstraction mechanism introduced in a previous paper and show that its application does not depend on the particular order it is applied. We formalise also the notion of lifeline introduced informally in the previous paper. We integrate our semantics with UML2.0 and show that they fit well together. We explain our approach using series of examples.

1 Introduction

The developments in areas of communication and information technology allow one to equip tools, with processors and software to facilitate their use. The tools used in everyday life are getting more and more smart due to build in electronic. One of the most important new concepts is the concept of mobile systems and of mobile computation. Code mobility, which emerged in some scripting languages for controlling network applications, 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 such as laptops, handhelds and PDAs can move between networks. Moreover, entire networks can be mobile as well, such as for example IBM's Personal Area Network (PAN) and networks of sensors in airplane or trains. Mobile computations can cross barriers and move between virtual and physical locations. The goal is to turn remote calls into local calls to avoid the latency caused by communication. But there is a price to pay since the administrative barriers and multiple access pathways interact in very complex ways.

These developments lead to enormous challenge of designing and configuring mobile and distributed systems that interact to achieve expected tasks. At the moment, this is a field of a very active multi disciplinary research. There are several aspects of such systems re-

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.

quiring different approaches. Specification, modelling and design belong to the most challenging ones. There exist several formalisms for specification of mobile systems (cf. e.g. [5, 4]), but until very recently, there was no satisfactory graphical notation for modelling mobile systems. Graphical Modelling Languages are influencing a great impact on the software development, but in the case of mobile systems, this aspect was neglected. These systems require special means for the modelling, specification and implementation.

Recently the gap was filled by designing appropriate graphical UML based notations. The so-called Sequence Diagrams for Mobility (SDM) [10] is a trace based, Sequence Diagrams like notation for the specification of mobile computation. There exists also an extension of UML Activity Diagrams for modelling of mobile system behavior [2]. The idea is similar to the idea of Ambients [4], in that a mobile object can migrate from one host to another and at the same time such an object can host other mobile 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 nested in an arbitrary way, generalizing the limited place-agent nesting of most agent and place languages. This concept generalises the Use Case Maps [3] in that we graphically model an object moving from one location to another, but also we allow moving objects to play the role of locations. The SDM notation generalizes the notion of object lifeline as defined in UML Sequence Diagrams [12].

One of the most important principles in science is the principle of abstraction. Ideally, there should be a notation allowing for displaying relevant and hiding irrelevant information. It should provide the possibility to abstract from features, which are irrelevant at a given stage of development. In the previous paper we have introduced a powerful mechanism for hiding irrelevant information [10]; the so-called zoom-out mechanism allows us to abstract from internal details of selected objects; we call such objects boundary. In particular, it allows us to hide the objects located in boundary objects and their behaviour. Similarly, we have introduced the so-called zoom-in mechanism for displaying details of objects and their behaviour.

The extension of Activity Diagrams [2] is very close to its origin, it uses only few new primitives form modelling of mobility and extends the standard UML notation a bit. The new primitives are defined using stereotypes, the standard UML extension mechanism. The SDM notation extends UML Sequence Diagrams in much more radical way and cannot be reduced so simply to the standard UML. Therefore in this paper we introduce formal partial order semantics for SDM only. We formalize the temporal ordering of event occurrences using partial order relations, more precisely quasi orders. The information stored in messages is accessible via labelling functions defined on elements of the partially ordered set. We show how to systematically define such models for SDM diagrams. We formally define the notion of lifeline, introduced in [10]. We have given some examples of its use, but we were not able to define it precisely there due to lack of proper terminology. Our formal semantics allows us to define it now in precise formal terms. We define the notion of a run and show how to figure out the system topology from the information contained in a run.

The tricky part in our semantics is the definition of object's location. We define locations only for objects participating in an event occurrence. Locations of other objects not related to the event occurrence are not fixed. For example, if an event occurrence is not temporally related to a move, then the move can happen before or after the event occurrence and the location of the moving object should not be fixed during the move.

The definition of our semantics is based on well defined topological artefact such as cross points, arrow directions and the relation of being located inside. Let us point out that there exist some formal partial order based semantics of Message Sequence Charts (MSC) [6], a graphical notation analogous to UML Sequence Diagrams (cf. e.g. [9]). MSC are a subject of a very intense research (cf. e.g. [11]).

We define semantics of the abstraction mechanism. The idea is to abstract from the information concerning hidden objects but to keep the partial ordering on visible communication events. We show, that the order, in which this mechanism is applied, does not matter. The local definition of object's location works fine also for the abstraction mechanism.

We show that our semantics fits very well to the concept of interaction defined in UML 2.0. Interestingly enough, the concept of SDM fits well to UML 2.0, but it was really hard to integrate with earlier versions of UML. We integrate the notion of partial order and run with the notions of trace and `GeneralOrdering` from UML 2.0. We formalise also the notion of lifeline as it is defined in UML 2.0.

Our paper is structured as follow. Section two presents the basic ideas of Sequence Diagrams for Mobility. In section three we define the formal model, which is the base of our semantics; we show how to define the semantics for concrete SDM diagrams. In section four we formalize the notion of abstraction. We conclude our paper with some remarks on the applications of our semantics.

2 The SDM Notation

Mobility is the ability to cross barriers. In our approach, a mobile object is also a location where interaction may happen. Action boxes are indicated by different locations. The action boxes describe what is inside and what is outside; they allow one also 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 modelling firewalls, administrative domains networks and so on. For example, a personal area network may be located in a car, which is located in a ferry; the ferry may enter a harbour and so on. We assume that the nested structure has the form of forest, i.e. an object can be located in at most one object and there no cycles of objects such that one is contained in another.

In the paper [2], we have introduced the stereotype `<<location>>` and the stereotype `<<mobile>>` to specify objects which can play the role of locations and objects which can be mobile, respectively. Each object of a class having stereotype `<<mobile>>` possesses attribute `atLoc`; this attribute has values of a class having stereotype `<<location>>`. If an object is not mobile, it does not possess this attribute. The idea is that a change of location of a mobile object is modelled by the change of the attribute `atLoc`. Objects which are locations only, and in general objects, which are not stereotyped with `<<mobile>>`, do not possess this attribute.

Mobile objects may interact with other objects by sending messages and changing locations. In UML, objects can communicate in synchronous as well as asynchronous way. We stick to this principle. Unlike Ambients Calculus [4], in our notation it is possible to express actions at a distance even if many barriers are involved.

A description of a mobile object's behaviour starts with a box containing optionally the object name and/or its class. A mobile object may move into another object, or move out of an object. If an object moves 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 move arrow.

A mobile object cannot continue its operation outside a 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 move. The start of operation of a mobile object (and if this object was not active before elsewhere) is indicated by a box as in the case of sequence diagrams. 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 (cf. [10]).

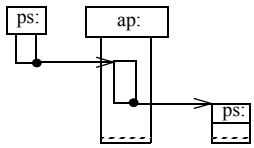

Fig. 1. Object mobility

Fig. 1 shows what a mobile object looks like. The 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. Than `ps` deplanes `ap` and starts its operation outside `ap`. The name in the last action box is not necessary either, since the identity of `ps` can be uniquely traced [10]. No message arrow is attached to the action boxes except of the move.

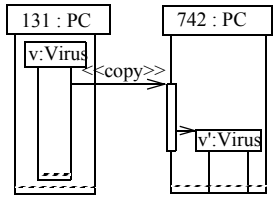

Fig. 2. Object copying and cloning

Fig. 2 shows a virus `v` located in PC `131`. The virus proliferates attacking other PCs. We use here a message arrow with UML stereotype `<<copy>>`, the copied virus `v'` is assumed to behave as its origin would do inside the new location (cf. [12]).

3 Partial Order Semantics

In this section we introduce partial order semantics for SDM. We define a mathematical model containing a partially ordered set and a number of labelling functions, which allow us to extract information from the elements of this set. We present a general method of extracting such models from SDM diagrams. We show how to apply this method to concrete SDM diagrams. We formally define the notion of lifeline.

3.1 The Formal Model

In this subsection we define the formal model which will be the base of our semantics. The model is based on a partial order formalizing temporal relationship between event occurrences; it contains labelling functions for extracting information. This model is constructed in a similar to the partial order semantics of Message Sequence Charts (cf. e.g. [9, 11]). The tricky point in our semantics is the definition of object locations. We define locations only for objects participating in an event occurrence; locations of other objects not related to the event are not fixed. For example, if an event occurrence is not temporally related to a move, then the move can happen before or after the event occurrence and the location of the moving object should not be fixed during the move. If the object moves from location `a` to location `b`, then when the event occurs it can be in `a` or in `b` (see below).

According to UML 2.0 [12], an `InteractionFragment` consists of a number of so-called `GeneralOrderings`. A `GeneralOrdering` represents an ordering of two event occurrences; it specifies that one event occurrence must proceed the other in a valid trace. This concept provides the ability to define partial orders of event occurrences. In UML, a message is a specification of a particular communication between instances in an interaction. A communication can be raising a signal, invoking an operation, creating or destroying an instance. Message specifies not only the kind of communication, but also the roles of the sender and the receiver, the dispatching and the relative sequencing of messages within the interaction. A message may have two message ends corresponding to two event occurrences: sending and receiving of a message (cf. Fig. 3). Event occurrences corresponding to message ends can be ordered using `GeneralOrdeing`.

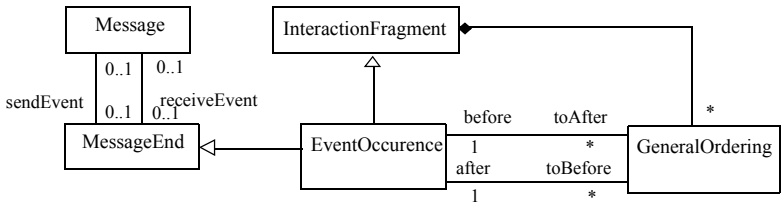

Fig. 3. UML event diagram

Our model is a tuple of the form: $(E, \leq, e_0, lab, p, lP)$. The binary relation $\leq$ is a quasi order, in particular a partial order. It corresponds to UML `GeneralOrdeing`. Conceptually, E is the set of event occurrences. We consider here four kinds of events: send event, receive event, method or constructor return and cross event; the last one corresponds to the event of crossing objects boundary[2]. Further, we identify three kinds of send events: message send, method call, constructor call, object departure and send message. Similarly, we identify three kinds of receive events: message receive, method start, constructor start, object arrival and receive message. The relation $\leq$ defines temporal ordering of such event occurrences. We assume that, there exist an initial, auxiliary, event occurrence e_0, which we use to define the initial topology of a mobile system (see below). This event precedes all other event occurrences, i.e. for all $e \in E$, $e_0 \leq e$.

2. One could split such an event into to a send event and a receive one.

An event occurrence includes information about the values of all relevant objects at the corresponding point in time [12]. In our model, the information corresponding to an event occurrence can be accessed using labelling functions. There are three labelling functions: lab, p and lP.

The function lab labels elements of the set E specifying the corresponding communication kind; it can be sending or receiving of a message, method call or return and so on (we have listed the types of events above):

$$lab : E \rightarrow EventKind$$

Every communication involves some objects; for example a message or a moving object may cross several object boundaries. The function p identifies objects participating in an event occurrence; it returns a finite set of event participants:

$$p : E \rightarrow Fin(ObNames)$$

We assume that in the case of object departure and arrival, function p identifies the moving objects and all objects hosted in the moving object (see the next subsection).

Let MO be the set of mobile objects, i.e. objects whose class has the stereotype `<<mobile>>`. Similarly, let Loc be the set of locations, i.e. objects whose class has the stereotype `<<location>>` (see section 2). We define a partial function:

$$lP : MO \times E \rightarrow Seq(Loc)$$

lP(o, e) returns a sequence of locations, if o participates in event occurrence e. In the other case, the value is undefined. We call this sequence location path of object o (see below).

In UML 2.0 [12], the semantics of an interactions is given in the term of traces. A trace is a sequence of event occurrences of the form $<e_0, e_1,..., e_n>$. Two interactions are equivalent if their trace-sets are equal. Similarly, in our model, system behavior can be specified as a set of system runs. A partial run of model $(E, \leq, e_0, lab, p, lP)$ is an not empty sequence of elements $e_0, e_1,..., e_n$ such that the following conditions are satisfied:

- If $e_i \leq e_j$, then $i \leq j$.
- If $e \leq e_j$, then there exists an index $i \leq j$ such that $e_i = e$.

In other words, a partial run is a linearization, which preserves the temporal ordering; moreover for every event occurrence belonging to the run, all preceding event occurrences belong to the run as well. Let us observe, that any not empty prefix of a partial run is a partial run.

Partial runs define particular moments in system execution. For such moments we can determine system topology. Let $e_0,..., e_n$ be a partial run, we say that after this run object o2 is located in location o1 (inside o1, resp.) iff there exists an event occurrence e_i and an object o such that:

- The location path lP(o, e_i) has the form $l_1,..., l_n$, contains locations l_j = o1 and l_{j+1} = o2 (contains locations l_j = o1 and l_k = o2, such that $j < k$, respectively).
- For all event occurrences e_k, such that $i < k$, and for all objects $o \in p(e_k)$, the location path lP(o, e_k) does not include o2.

The location of an object is determined by the most recent information concerning its location. Similarly, we can define location path of an object after a run. The topology of a mobile system at a particular moment of time is derived from the information contained in the events occurring in the corresponding partial run; it is the sum of object locations.

Models that are semantics of SDM diagrams have certain properties, which formal models in general may violate. We call such properties consistency conditions. For example the topology of a mobile system should not change during method call or return. The topology changes only in the case of object departure or arrival and constructor return. Moreover, departure and arrival change only the location (i.e. the `atLoc` attribute) of the moving object. Locations of all other objects remain unchanged. Constructor return doesn't change locations of already existing objects. As mentioned in section 2, we assume that the locations form a forest. Consistency conditions can be used when proving or model-checking properties of SDM diagrams.

3.2 Definition of the Partial Order Semantics

In this section, we introduce a generic method for defining a partial order semantics for a concrete SDM diagram. This method includes three steps: identification of event occurrences, derivation of the partial ordering and labelling of the occurrences.

We associate an element of the set E to every graphical artefact of the form: beginning and end of a message arrow (e.g. `move` message), a message arrow crossing an object box by going in or out as well as method and constructor termination indicated by a corresponding rectangle. Let us point out that if an arrow crosses an object box twice, then it does not interfere with the action box, such as for example a message arrow crossing a lifeline of an object in UML sequence diagram[3].

The definition below does not guarantee that we obtain a partial ordering. It yields only a quasi order, i.e. a reflexive and transitive relation. In fact, it depends on the SDM diagram being formalized, if we obtain a partial order or not. As in the case of general Sequence Diagrams, it is possible to draw diagrams with circular dependences. In such a case, the resulting quasi order is not a partial order.

For simplicity, we do not talk about the topological artefacts, but about the corresponding event occurences (see above). The temporal ordering of event occurrences $\leq$ is defined as the smallest reflexive and transitive relation satisfying the following conditions:

- If event occurrence e_1 is located above event occurrence e_2 on a rectangle being a border of the same action box, then $e_1 \leq e_2$.
- If e_1 and e_2 are located on the same message arrow and e_1 proceeds e_2 in respect to the direction of the arrow, then $e_1 \leq e_2$.

For every element of E we define the values of functions lab, p and lP. The function lab returns the type of an event: message send, message receive, method or constructor call, method or constructor start, method or constructor return, cross event; method call, object departure and object arrival.

The function p identifies participants of a communication event e. If it is a send of a message, arrival of a message, method call, start of a method execution, method return, then p(e) includes only the sender, message receiver, caller, executing object, respectively. In the case of object departure and arrival, we assume that the event participants are the moving object and additionally all objects located directly or indirectly in the moving object; those objects

3. For simplicity, we do not consider cases when a message arrow crosses object boxes of the same object several times.

can be identified by figuring out if the corresponding action boxes are located inside the action box of the moving object. The reason for including such object is that an object located in a moving object changes its location path when its host moves. The move is also a kind of caesura for the participating objects (see below).

The auxiliary element e_0 defines the initial topology of the system. $p(e_0)$ includes all objects which exist initially, i.e. objects whose object boxes start with a rectangle bearing a name of an object (see for example `131` on figure 4). If an event occurrence e corresponds to a message arrow crossing an action box of object o, then we assume that $p(e) = \{o\}$. If an event e corresponds to the moving of object o, then $p(o)$ contains all objects with object boxes located in the object box of o before the move.

For every event e and every object o taking part in this event (i.e. for every $o \in p(e)$), location path $lP(e, o)$ is defined as the sequence of objects such that their object boxes contain the object box of o on which e is located; the objects are listed from the inner most to the outer most. If an object o' neither takes part in e nor occurs at a location path of an participating object, then the value of $lP(e, o')$ is undefined. This is due to the fact that we do not want to restrain locations of objects not involved in an event. Concurrently executing objects may change locations independently. Let us observe that location paths are defined for all objects existing initially, since they belong to $p(e_0)$. Consequently, the initial topology is fully determined. Let us also observe that in the case of move, the locations of participating objects before and after the move are defined by the location paths corresponding to the departure and arrival event occurrences, respectively. In general the topology in different moments of time is determined by location paths. The location of an object is determined by the last relevant event.

3.3 Example

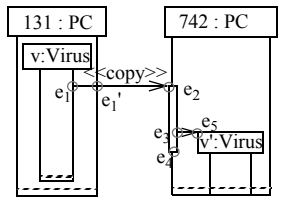

Fig. 4. Object copying and creation

In this subsection we show how to define the partial order semantics for a concrete SDM diagram. For simplicity, we assume that all objects occurring in the diagrams are mobile, i.e. the corresponding classes have the stereotype `<<mobile>>` (see section 2). Fig. 4 makes explicit the event occurrences from Fig. 2: e_1 is a method call, e_1' corresponds to crossing the boundary of `131`. e_2 is the start of the method execution, e_3 is a constructor call, e_4 is the termination of the method and e_5 is the start of constructor execution.

There set of event occurrences has the form $E = \{e_0, e_1, e_1', ..., e_5\}$. e_0 proceeds all other events. e_1 proceeds e_1' and e_1' proceeds e_2; this is due to the ordering of these occurrences on the corresponding message arrow. e_2 proceeds e_3 and e_3 proceeds e_4; this is due to the

to down ordering of events on the object box of 742. Finally, e_3 proceeds e_5. The labelling function lab returns call for e_1, since it is of type call. It returns cross for e_1', start for e_2, call for e_3, start for e_5 and return for e_4.

Initially, there exist three objects: v, 131 and 742; therefore $p(e_0) = \{v, 131, 742\}$. The participants of occurrence e_1 are: v, 131. Similarly, $p(e_1') = \{131\}$, $p(e_2) = \{742\}$, $p(e_3) = \{742\}$, $p(e_4) = \{742\}$ and $p(e_5) = \{v'\}$. The topology is defined by the function lP: $lP(e_0, v) = <131>$ and $lP(e_0, 131) = lP(e_0, 742) = <>$. The location of object v during occurrence e_1 is 131, i.e. $lP(v, e_1) = <131>$. $lP(131, e_1)$ is an empty list and $lP(742, e_1)$ is undefined (see below).

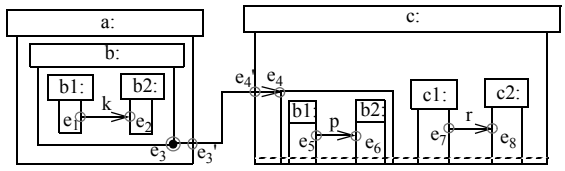

Fig. 5. Complex move

Fig. 5 shows communicating objects which change their locations. Object b moves from location a to location c. Objects b1, b2 communicate before and after the move. The communication between objects c1 and c2 does not depend temporally on the move. The initial topology is defined as follows:

$p(e_0) = \{a, b, b1, b2, c, c1, c2\}$,

$lP(b, e_0) = <a>$, $lP(b1, e_0) = lP(b2, e_0) = <a, b>$, $lP(c1, e_0) = lP(c2, e_0) = <c>$.

Object b1 is the only participant of occurrence e_1. When occurrence e_1 takes place, the location of b1 is b and the location of b is a. Consequently the location path of b1 during e_1 equals (b, a). There are three participants of the occurrence e_3, i.e. $p(e_3) = \{b, b1, b2\}$, and for participants of e_3', i.e. $p(e_3') = \{a, b, b1, b2\}$[4], this corresponds to the assumption that objects located in a moving object participate in the move. Similarly, $p(e_4') = p(e_4) = \{c, b, b1, b2\}$. The move is a kind of caesura for the involved objects. It separates events before the move and events after the move.

Let us observe, that we cannot infer from the diagram the temporal ordering of e_1 and e_7. Similarly, we cannot infer from the diagram where object b is located when occurrence e_7 happens; the location of object b during occurrence e_7 may be a or c or none of them. This explains why lP is a partial function.

4. Let us notice, that there is difference between $p(e_1')$, from figure 4 and $p(e_3')$ from figure 5. In the first case, the message causes an object creation, but the object does not exist when the message is sent. In the second case, the object move all together and therefore they are listed as event participants.

There are several partial runs of the system shown on Fig. 5. For example, e_0, e_1, e_2, e_3, e_3', e_4', e_4,e_5, e_6, e_7, e_8 is a maximal partial run in the sense that it contains all event occurrences. Similarly, e_0, e_7, e_8, e_1, e_2, e_3, e_3', e_4', e_4, e_5, e_6 is another maximal partial run. Of course, all prefixes of these runs are partial runs. Let us consider the first run, the initial topology is not changed after e_1 nor after e_2. After occurrence e_3, object `a` does not contain any other object and object `c` contains objects `c1` and `c2` only. The topology changes once more after occurrence e_4: object `c` contains now additionally `b1` and `b2`.

3.4 Lifelines

In this subsection, we formally define the notion of lifeline. We have introduced this notion in the paper [10] already. We have given some examples of its use there, but we were not able to define it precisely due to lack of proper terminology. Our semantics allows us to define the notion of lifeline in precise formal terms. This notion generalizes the notion of object lifeline as defined in UML Sequence Diagrams [12]. It generalizes also the idea of Use Case Maps [3]; this notation strictly separates mobile objects and locations. In SDM an object can be mobile, if its class has the stereotype `<<mobile>>` and at the same time it can play the role of location, if its class has the stereotype `<<location>>`.

Let in UML 2.0 "The semantics of the lifeline (within an interaction) is the semantics of the interaction selecting only event occurrences of this Lifeline." Our definition of object lifeline corresponds strictly to this definition: a lifeline of an object is the set of all events the object participates in. Formally: let $(E, \leq, e_0, lab, p, lP)$ be a SDM semantics, and let o be one of the participating objects, the *lifeline* of the object o is the set

$\{e \in E \mid o \in p(e)\}$

The partial order relation on the set E orders the event occurrences belonging to a lifeline. So the temporal ordering of the lifeline is simply inherited from the superset E.

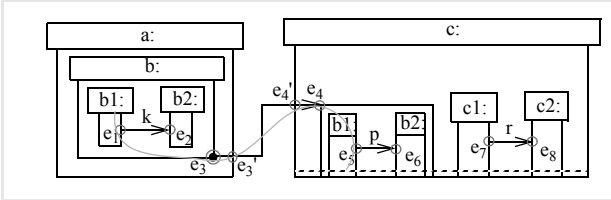

Fig. 6. Lifeline

As example let us consider the diagram in Fig. 6. The lifeline of object `b1` has the form: $\{e_0, e_1, e_3, e_3', e_4', e_4, e_5\}$. The lifeline of object `b` consists of occurrences: e_0, e_3, e_3', e_4' and e_4.

4 Abstraction Mechanisms

One of the most important principles in science is the principle of abstraction. Ideally, there should be a notation of abstraction allowing one for displaying relevant and hiding irrelevant information. In the previous paper we have introduced a powerful graphical mechanism for hiding irrelevant information [10]. The zoom-out mechanism allows us to abstract from in-

ternal details of so selected objects, called boundary. In particular, it allows us to hide the objects located in boundary objects and their behaviour. Similarly, we have introduced the so-called zoom-in mechanism for displaying details of objects and their behaviour.

In the first subsection we formalize the zoom-out mechanism. In the second subsection we explain how the formal machinery works using an example. In the third subsection we show how to formalize zoom-in to a move.

4.1 Formalization of the Zoom-Out Mechanism

In this subsection, we formalize the zoom-out mechanism. This mechanism allows us to abstract from the internal structure and behavior of selected objects, which we call boundary. We define which event occurrences are visible and which not. The zoom-out mechanism can be applied to whole models, but it can be applied as well to selected time intervals. The definitions prove to be simple, thanks to the local definition of object's locations.

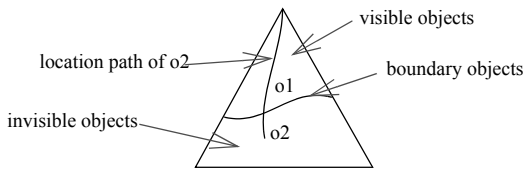

Fig. 7. Boundary objects

Fig. 7 shows the basic idea of boundary objects. The objects located in a selected object can be seen as a tree. The objects located in boundary objects are invisible, i.e. an object is invisible, if its location path contains a boundary object, in the other case it is visible. The objects located above boundary objects are visible. For example, let `o1` and `o2` be two objects participating in an event occurrence; `o1` is visible, but `o2` not. Similarly, we can define visible and invisible events.

Formally, let BO be a set of boundary objects, and let e be an event occurrence, we say that e is *not visible* in respect to BO iff one of the following conditions is satisfied:

- For all objects $o_1 \in p(e)$, the location path of o_1 contains a boundary object.
- e is a send event, the initiating object (i.e. the caller, the departing object or the sender), respectively is a boundary object and the target object (i.e. the receiver, the target location or the receiver, respectively) is located within the caller.

A *event occurrence is visible* if it is not invisible. In other words, an occurrence is not visible, if all objects participating in the occurrence are invisible or it is a send event, the initiating object is a boundary object and the target object is not visible. Partial runs correspond to points of execution. We say that an *object is visible after a partial run*, if this object is not located within a boundary object (see subsection 3.1).

Let us observe, that in the case of formal models satisfying the consistency conditions, for every two partial runs ending with the same event occurrence e, the sets of visible objects participating in this event occurrence are the same. In other words, for every occurrence and every object participating in the occurrence the fact whether the object is visible or not, does not depend on run the event is part of. More generally, if two event occurrences

concern the same object, then they are temporally related. This follows from the fact that event occurrences on the same object box are temporally related. If an object moves or if its host moves then the object is involved and the move is a kind of ceasura.

We define an abstraction function F. This function has two arguments: a set of boundary objects and a model:

$F(BO, (E, \leq, e_0, lab, p, lP)) = (E', \leq', e_0, lab', p', lP')$, if the following conditions are satisfied:

- $E' = \{e \in E \mid e$ is visible in respect to $BO\}$.
- $\leq'$ is the restriction of $\leq$ to E'.
- lab' is the restriction of lab to the set E'.
- $p'(e) = \{\circ \in p(e) \mid lP(\circ, e)$ does not include objects from $BO\}$.
- $lP'(\circ, e)$ is defined as $lP(\circ, e)$, if e is a visible event occurrence and $\circ \in p'(e)$; lP' is undefined for other pairs of objects and occurrences.

E' is the set of visible events. p' contains objects visible during occurrence of e. lP' is defined as lP for objects visible during event occurrence, for other objects it is undefined. Let us observe, that by definition, for all visible event occurrences e and all $\circ \in p'(e)$, $lP'(\circ, e)$ does not contain boundary objects.

We may perform abstraction several times; the result should not depend on the order we apply the zoom-out mechanism. The following statement says that this requirement is satisfied. It is due to associativity and commutativity of set theoretical union.

Statement

$F(BO_1, F(BO_2, (E, \leq, e_0, lab, p, lP))) = F(BO_1 \cup BO_2, (E, \leq, e_0, lab, p, lP))$

The statement follows from the fact, that it does not matter whether we abstract from event occurrences invisible in respect to BO_1 first and then from occurrences invisible in respect to BO_2, or we abstract from occurrences invisible in respect to $BO_1 \cup BO_2$ in one step. Consequently, the resulting set of visible occurrences depends only on the union. The resulting partial order is just the restriction of the initial partial order. The resulting participation function and the function returning location paths depend only on the union $BO_1 \cup BO_2$.

We may apply the zoom-out mechanism in a much finer way. It can be applied not only to whole lifelines of objects, but also to particular time intervals when the behaviour and internal structure of selected objects is unimportant.

Let $(E, \leq, e_0, lab, p, lP)$ be a partial order semantics of a mobile system and let N be a subset of E. We say that N is convex iff for every three occurrences e_1, e_2, e_3, if $e_1, e_2 \in N$ and $e_1 \leq e_2 \leq e_3$, then the element e_2 belongs to N as well. The definitions above can be formulated for convex sets of occurrences N:

An event occurrence is invisible relative to N iff it belongs to N and it is invisible in the above defined sense. We can redefine the functions lab, p, lP for event occurrences from N in an analogous way.

4.2 Zoom-Out: Examples

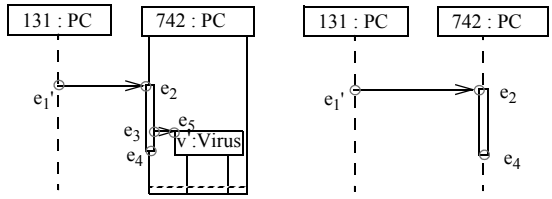

Fig. 8. Abstracting from internal details

In the first example we show how to gradually abstract from the details of the interaction shown on Fig. 2. We present two views on the interaction. The left hand side of Fig. 8 shows the receiver view. The receiver of a virus usually cannot see the structure of the virus sender, but it may figure out who the sender of the virus was. A network observer can see only the communication over the network, but not the internal structure of the communication participants.

In the case of the first diagram, `131` is the only boundary object, i.e. $BO_1 = \{131\}$. e_1 is the only invisible event, consequently $M' = \{e_0, e_1', ..., e_5\}$. There are initially two objects: `131` and `742`; therefore, $p'(e_0) = \{131, 742\}$. `131` is the only participant of the occurrence e_1', i.e. $p'(e_1') = \{131\}$. $p'(e_2) = \{742\}$, and so on. The the function $lP'(\_, e_0)$ is defined for objects `131` and `742`.

In the network view, the set of boundary object is equal {`131`, `742`}. There are only three event occurrences in this view: e_1', e_2 and e_4. Only the visible objects participate in these events.

The next example concerns partial zoom-out. We can abstract from internal structure and behavior of selected objects during certain time intervals. Fig. 9 abstracts from the behaviour and internal structure of `b` after it moves.

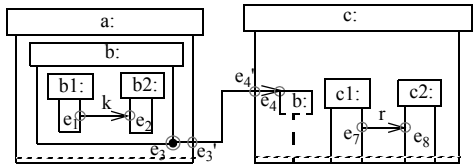

Fig. 9. Partial zoom-out

Objects `b1`, `b2` communicate before and after the move. The initial topology is defined as follows: $p'(e_0) = \{a, b, b1, b2, c, c1, c2\}$, $lP'(b, e_0) = \langle a \rangle$, $lP(b1, e_0) = \langle a, b \rangle$, $lP'(b2, e_0) = \langle a, b \rangle$ and so on. There are four participants of occurrence e_4: $p'(e_4) = \{c, b, b1, b2\}$. $lP'(\_, e_4)$ is defined only for objects `b` and `c`.

4.3 Zooming into move

In this subsection we formalize the zoom-in mechanism allowing us to display and to hide the details of object's move. It is possible to zoom into the object's move arrow to see the behavior of the participating objects. The top part of Fig. 10 shows the move of object `b` from location `a` to location `c`. All objects hosted by `b` participate in this move. In the top of the figure, the move is shown in the zoom-out view. The second diagram in Fig. 10 shows the move details, i.e. the zoom-in view. It displays the communication between `b1` and `b2` during this move. The zoom-in version of the `move` 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 move and the end of the move.

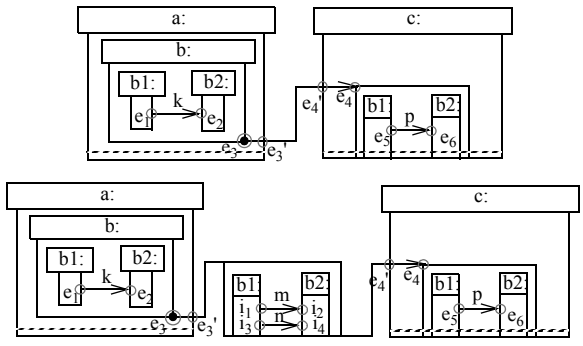

Fig. 10. Zooming into a message

The zoom-out view can be seen as an abstraction of the detailed view. In this case the convex set is the interval $(e_3, e_4) = \{e \mid e_3 \le e \le e_4\}$, and the boundary set contains the object b only. The set of event occurrences corresponding to the first class diagram has the form $\{e_0, e_1, ..., e_6\}$. The temporal order is linear: $e_1 \le ... \le e_6$. In the detailed view new occurrences are added and the temporal ordering is extended:

$e_0 \le e_1 \le e_1 \le e_3 \le e_3' \le i_1 \le i_2 \le i_4$, $i_1 \le i_3 \le i_4 \le e_4' \le e_4 \le e_5 \le e_6$.

Concluding Remarks

The formal, partial order based semantics, presented in this paper explains the meaning of Sequence Diagrams for Mobility. It allows us also to formalize the abstraction mechanism introduced in the previous paper [10]. This semantics is well integrated with UML 2.0. Let us observe that it is possible to have a bit different semantics of SDM which assigns events only to send and receive actions (cf. [10]). For example, when an object located in another object sends a message, the message may cross the object box of the outer object; we may skip event occurrences corresponding to crossing of those boxes.

In the future, we are going to use this semantics to implement tools for graphical modelling of mobile systems. We are also going to investigate to what extend the decidability results and algorithms concerning Message Sequence Charts (cf. [11]) apply to SDM.

Acknowledgments

The author expresses cordial thanks to Martin Wirsing and the PST-research group at University of Munich for their helpful remarks on this semantics. He is also thankful to the anonymous referees whose critical comments helped to improve this paper.

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. Baumeister H., Koch N., Kosiuczenko P., Wirsing M.: Extending Activity Diagrams to Model Mobile Systems. M. Aksit, M. Mezini, R. Unland (eds.): Objects, Components, Architectures, Services, and Applications for a NetworkedWorld, NetObjectDays Conference, LNCS 2591, Springer 2002, pp. 278 -293.
3. Buhr R., Casselman R.: Use Case Maps for Object-Oriented Systems, Prentice-Hall, USA, 1995.
4. Cardelli L.: Mobility and Security. Bauer, F., Steinbrüggen, R. (eds.): Foundations of Secure Computation. Proc. NATO Advanced Study Institute. IOS Press, 2000, 3-37.
5. 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.
6. ITU-T. Telecommunication Standardization Sector. Message Sequence Charts, Z.120. 11.1999.
7. Jing J., Helal A., Elmagarmid A.: Client-Server Computing in Mobile Environments. ACM Computing Surveys. Vol. 31(2), 1999, 117-157.
8. Klein C., Rausch A., Sihling M., Wen Z.: Extension of the Unified Modeling Language for Mobile Agents. Idea Publishing Group, 2001.
9. J.P. Katoen, L. Lambert. Pomsets for MSC. Proc of FBT'98, H. König and P. Langendörfer (eds.): Formale Beschreibungstechniken für verteilte Systeme, Shaker Verlag, Aachen, 1998.
10. Kosiuczenko P.: Sequence Diagrams for Mobility. Krogstie J. (ed.): Advanced Conceptual Modeling Techniques: ER 2002 Workshops, Tampere, Finland, October 7-11, 2002, LNCS 2784, Springer, Berlin, 12 pages, 2003.
11. A. Muscholl, D. Peled, Z. Su. Deciding properties of message sequence charts, Proc. of FoSSaCS'98, LNCS 1378, pp. 226--242, 1998.
12. OMG: Unified Modeling Language. Version 2.0. Superstructure Final Adopted specification. 03-08-02.