Conceptual Modelling of Mobile Object Systems: Some Language Constructs

Peter Ahlbrecht, Silke Eckstein, and Karl Neumann

Technical University Braunschweig, Information Systems Group,
P.O.box 3329, D-38023 Braunschweig, Germany,
{ p.ahlbrecht | s.eckstein | k.neumann }@tu-bs.de,
WWW home page: http://www.cs.tu-bs.de/idb/welcome_e.html

Abstract. Mobile hardware and software gain wide acceptance, and a couple of corresponding specification and modelling languages have been developed alongside. As these languages so far do not allow to distinguish between the ever-mobile units and the stationary subsystems forming their context—which is an essential distinction for many scenarios—in this article we present some suitable constructs as an extension of an already existing specification language.

1 Approaches to the Specification of Mobile Systems

Mobility has numerous effects influencing both hardware and software, and several languages have been developed to specify and model these effects. As the currently most widespread semi-formal specification language, already the "basic" version of the Unified Modeling Language (UML) provides sufficient means to model mobile objects with its ≪become≫ stereotype Additionally, several extensions to the UML have been proposed to make the language more suitable for specifying mobile agent systems, as, e.g., in [1]. Numerous formal methods that treat the concept of mobility explicitly are process algebras, with the π-Calculus [2] being its most well-known representative. It adopts a notion of mobility that the area of movement is the space of linked processes in which the moving entities are the *links*. Instead of movement of processes the π-Calculus is rather concerned with the movement of *access* to processes. A frequently mentioned argument in favour of mobile agents, however, is that in case a computation is expected to work on a large amount of data, it will be more efficient to move the computation to the data than to transfer the input to the computation and evaluate it locally [3]. This argument may be extended to mobile systems in general: if performance of a system depends on its location, then a model which captures mobility by movement of access may not be adequate. To represent such scenarios the Ambient Calculus [4] or the Algebra of Itineraries [5] seem to be better suited. An ambient is a bounded named place where computation takes place. The boundary captures location-dependent qualities. Location is also explicitly represented in [5], which concentrates on providing means to model itineraries of movement in a flat structure of places. In contrast to this

ambients can be nested. They may access each other when they are nested at the same level within the same super-ambient, and movement can be expressed by *capabilities* which allow them to enter, exit, and open another ambient if its name is known. Just like Mobile Maude [6], the Distributed Join-Calculus [7] focuses on mobile agents. Like the enriched π_1-Calculus [8], in addition to offering means for specifying location-dependent qualities the Distributed Join-Calculus also offers constructs for modelling failure of locations. Mobile Unity [9], too, provides means to explicitly represent location. In contrast to the other approaches this is achieved via a location attribute and constructs which reflect location dependency in communication, namely transient variable sharing and transient action synchronisation, which allow entities to share data and synchronise actions when in close proximity. A strict separation between mobile and stationary units, together with the possibility of grouping them, is thus so far not available. As declarative languages are considered to be especially suited for developing programs from the results of the requirements analysis [6], it was reasonable to take the declarative object-oriented specification language TROLL [10] developed at our group and extend it with constructs which facilitate making such a distinction.

2 Modelling Mobile Object Systems

The object-oriented language TROLL has been developed for specifying information systems at a high level of abstraction. In this approach information systems are regarded as being communities of concurrent, interacting (complex) objects. Objects are units of structure and behaviour. They have their own state spaces and life cycles, which are sequences of events. A TROLL specification is called *object system* and consists of a set of object class and datatype specifications, a number of object declarations, and the definition of global interaction relationships. Module concepts facilitate structuring system specifications by treating classes and objects having a common task or a common field of functions as a logical unit [11]. Such structured specifications may be regarded as trees whose internal nodes represent complex subsystems, which in turn may contain other subsystems and their interaction relationships. The leaves—apart from specially marked interfaces—represent the base classes and objects. According to [12], spatial configurations, too, can be represented as trees, and mobility can then be expressed as the time-dependent variation of these configurations. Our proposal seizes these two utilisations of trees and allows to distinguish between the mobile and stationary parts of a complex system by means of providing language constructs allowing to combine stationary and mobile parts. Furthermore, it also permits to describe how communication links between such units are established and released. The distinction between the ever-mobile units and those which provide the fixed subsystem as the context for the mobile entities is reflected syntactically in the way that a stationary part may contain additional stationary as well as mobile parts, whereas a mobile part may only contain other mobile parts:

<*mobiSpec*> ::= **mobile** <*ident*>
(<*mobiSpec*> | <*subSpec*>)

{ ';' <*mobiSpec*> | <*subSpec*> |<*mobiInstanceDecl*> }
[**onEntering** { <*linkDef*> ';' } [<*actionTerm*>] **end**]
[**onExiting** { <*unlinkDef*> ';' } [<*actionTerm*>] **end**]
end_mobile

<*statSpec*> ::= **stationary** <*ident*>
(<*statSpec*> | <*mobiSpec*> | <*subSpec*>)
{ ';' <*statSpec*> | <*mobiSpec*> | <*subSpec*> |
<*statInstanceDecl*> | <*mobiInstanceDecl*> }
[**onEntry** { <*linkDef*> ';' } [<*actionTerm*>] **end**]
[**onExit** { <*unlinkDef*> ';' } [<*actionTerm*>] **end**]
end_stationary

<*subSpec*> ::= [<*offerSpec*> { ';' <*offerSpec*> }]
[<*requestSpec*> { ';' <*requestSpec*> }]
[<*specItem*> { ';' <*specItem*> } ';']

On the one hand, the <*subSpec*> production allows to state object declarations, datatype and object class specifications by resolving the non-terminal <*specItem*>. On the other hand, interface objects can be declared offering services and information to other subsystems (<*offerSpec*>) or requesting such services and information (<*requestSpec*>). For that purpose action signatures are specified inside these objects, which can then be used within the <*linkDef*> /<*unlinkDef*> productions of the **onEntering/onExiting/onEntry/onExit** clauses to describe how communication links are established or released:

<*unlinkDef*> ::= **unlink** <*ident*> '.' { <*ident*> '.' } <*actionSignature*>
<*actionSignature*> ::= <*ident*> ['('<*parameter*> {',' <*parameter*> }')']

In case of a mobile unit migrating into another mobile or stationary subsystem it has to be checked if both provide action signatures with an equal number of corresponding types of parameters. If this is the case, then the link is considered to be established. Similarly it has to be specified how the incorporating and the moving entity respond to "emigration". A link that is not specified to be released (by means of an appropriate <*unlinkDef*> specifications of the participating subsystems) persists migration. The number of the subsystems surrounding them—and thus effectively lying in between them—indicates the increasing complexity of a communication link. The asymmetry of communication between mobile and stationary components is described directly by the two types of subsystems. The initial configuration of the system is given by the declarations of the object classes and the mobile and stationary subsystems. The migration of a mobile unit—and therefore the next configuration—is specified by the <*movement*> production:

<*movement*> ::= **if** <*pFormula*> **moveTo** <*target*>

where <*pFormula*> is a proposition which can be composed of formulas using the usual logical connectives. If this proposition is evaluated to true, the unit enclosing this object migrates into the subsystem specified by <*target*>.

3 Future Work

In order to provide the formal semantics, the behavioural part of a specification in the unextended TROLL is mapped to formulae of a distributed temporal logic.

An action rule, e.g., defines the effect an occuring action has on an object. A precondition for an action may be specified after the keyword `onlyIf`, and the expression enclosed between `do` and `od` can be regarded as a postcondition. Intuitively, an action call can be understood as a simultaneous execution of the calling and called action, where the former provides the actual values for the input parameters of the latter as well as pairwise disjoint variables for its output parameters. Action rules are built on top of data terms and propositions, which are also mapped to formulae of the logic. As satisfaction of an action rule is considered from the point of view of the state immediately following the action call, the mappings are performed relating to the immediately preceeding state. An essential task is now to provide the formal semantics for the new constructs which we only introduced syntactically. In doing so, we have to analyse to what extend the distributed temporal logic that so far constitutes a part of TROLL's semantical foundations can be used or suitably extended. In addition, the usability of the language constructs should be put to the test in application development, for example in the realisation of a system for information transfer from web services to mobile devices.

References

1. Klein, C., Rausch, A., Sihling, M., Wen, Z.: Extension of the Unified Modeling Language for Mobile Agents. In Siau, K., Halpin, T., eds.: Unified Modeling Language: System Analysis, Design and Development Issues. (2001) 116–128
2. Milner, R.: Communicating and Mobile Systems: the π-Calculus. 2nd edn. Cambridge University Press (2001)
3. Lange, D., Oshima, M.: Seven Good Reasons for Mobile Agents. Communications of the ACM **42** (1999) 88–89
4. Cardelli, L., Gordon, A.D.: Mobile Ambients. In: Proc. 1st Int. Conf., FoSSaCS '98. LNCS 1378 (1998) 140–155
5. Loke, S.W., Schmidt, H., Zaslavsky, A.: Programming the Mobility Behaviour of Agents by Composing Itineraries. In: Proc. 5th Asian Computing Science Conf. LNCS 1742 (1999) 214–226
6. Durán, F., Eker, S., Lincoln, P., Meseguer, J.: Principles of Mobile Maude. In: 4th Int. Symp. on Mobile Agents, ASA/MA 2000. LNCS 1882 (2000) 73–85
7. Fournet, C., Gonthier, G., Lévy, J., Maranget, L., Rémy, D.: A Calculus of Mobile Agents. In: CONCUR '96: Proc. 7th Int. Conf. LNCS 1119 (1996) 406–421
8. Amadio, R.M.: An Asynchronous Model of Locality, Failure, and Process Mobility. In: Proc. 2nd Int. Conf., COORDINATION '97. LNCS 1282 (1997) 374–391
9. Roman, G.C., McCann, P.J., Plun, P.J.: Mobile UNITY: Reasoning and Specification in Mobile Computing. In: ACM ToSEM 6. Volume 3. (1997) 250–282
10. Hartel, P., Denker, G., Kowsari, M., Krone, M., Ehrich, H.D.: Information systems modelling with TROLL — formal methods at work. Inf. Sys. **22** (1997) 79–99
11. Eckstein, S.: Modules for Distributed Object Systems—Concepts for Structuring and Reusing Object-Oriented Specifications. infix-Verlag, (2001) In german.
12. Cardelli, L., Gordon, A.: Anytime, Anywhere. Modal Logics for Mobile Ambients. In: Proc. 27th ACM Symp. on Principles of Programming Languages. (2000)