International Journal of Computer Science & Applications
Vol. II, No. II, pp. 101 - 117

Towards Mobile Z Schemas

M. Bettaz, M. Maouche
Department of Computer Science
Philadelphia University 19392 Jordan
{bettaz, maouch}@philadelphia.edu.jo

Abstract

In this paper we show how to extend the Z language with location and mobile operation schemas. The objective is to seek appropriate models for tackling the space and coordination dimensions of mobility in a software engineering perspective. A case study from the literature is presented to illustrate the benefits of our mobility framework.

Keywords and phrases: Mobile computing, wireless networking, location, portability, mobility, mobility models, Z notation, Mobile Z schemas.

1. Introduction

In the late seventies, the International Standards Organization [ISO] set through its Open System Interconnection [OSI] model, a general framework defining in great detail communication layers and protocols in computer networks. Peer to peer communication seemed hence to be a "natural" paradigm; however the technological trends appearing afterwards, the rapid growth of the Internet, and the great success of TCP/IP imposed de facto the client-server model as an effective model [5]. With the advent of mobile computing and wireless networking, the notion of "stable" clients and "stable" servers is becoming senseless, since both location-transparent and context-dependent services are desired. Decoupled computing is becoming the new computation paradigm [3]. Network disconnection is no longer considered as a fault, but a common event intentionally caused by the user in order to conserve power even when moving [9]. We are confronted to applications, where the network and program structure may change: nodes may dynamically appear and disappear, processes may move among nodes, and change structure. In this evolving world of computation it is yet to early to give well-established classifications, although some attempts may be cited. [13, 14] classify mobile computing into nomadic and ad-hoc computing. Nomadic computing denotes systems consisting of a fixed core network and a fringe of mobiles that is connected to it through base stations. Delivery of data to a mobile unit presupposes the ability to find the current location of the unit and to continue to send data as the unit moves from one place to another. Cellular phone systems accomplish this through a combination of broadcasting (to notify the unit about the incoming call) and

handover protocols (to maintain the connection in the face of the movement). Ad-hoc networks refer to systems consisting of mobile devices exclusively. These devices are connected to each other when they are within communication range. An individual mobile device may or may not act as ad hoc router for the benefit of its neighbors. The case study presented in this paper deals with this kind of mobility. However the notion of dummy location ($\perp$) introduced in our work, is quite general and may be associated with nomadic as well as ad-hoc networks.

This chaotic form of computing cannot be dealt with in a unique framework. In [14], authors make an exhaustive tour of the problems raised by mobility issue (as opposed to stability) and try to formulate a simple framework from a software engineering perspective. They suggest reviewing theoretical aspects of software engineering in the context of mobile computing. In the present work, we follow this suggestion, tackling the early stages of the life cycle using a reviewed version of Z schema language. Models vary greatly in the way they answer questions such as who is allowed to move, where it can go, and how context changes caused by movement are managed. Some models use the concept of location to deal with mobility. This is the case of Mobile Unity [4, 15], where location is modeled explicitly as a distinguished variable which belongs to the state of a mobile component. Changes in its value correspond to changes in the position of the component. Other models equate mobility with a more general notion of change. In π-calculus [10], for instance, there is no notion of location built into the model, and yet the structure of the system can change dynamically.

In this paper we propose to extend the Z schema language with the notion of location schemas and mobility operation schemas, getting mobile Z schemas that allow to deal with the space and coordination dimensions of mobility. As in [12], we contend that modeling the space explicitly is desirable when one hopes to take into account the physical reality of moving devices and its implications on the behavior of the software they carry. The rest of this paper is organized as follows. Section 2 is devoted to related works. In section 3 we clarify the notions of location, portability, mobility, and their domain dependent relation. In Section 4, we introduce the notion of location schemas and mobile operations schemas to obtain mobile Z schemas. The reader is supposed to be familiar with Z notation. In section 5, we adapt a case study presented in [7] to show the benefits of our Mobile Z framework. Section 6 deals briefly with proving issues. Concluding remarks are given in section 7.

2. Related Works

To our knowledge, the notion of mobile Z schemas is introduced for the first time in this work. However our contribution was inspired by similar works on mobility mainly that on Mobile Unity by G. –C. Roman et al. [12]. Other similar ideas may be found in contributions [8, 10, and 17]. [12] is building on a procedural like language. [8] is oriented towards rewriting logic, [10] is based on process algebra, while our contribution is model oriented. Although [17] is model oriented and built on Z, it is mainly oriented towards the coordination dimension of mobility, while our contribution addresses the space as well as the coordination dimensions of mobility. Another difference with [17] is that we address

the coordination dimension through mobility operation schemas, while [17] addresses this dimension using mobile agents.

3. Location, Portability and Mobility

Dealing explicitly with the mobility dimension requires introducing and clarifying the concepts of motion and location. Although their meanings are often domain dependent, some elements can be stated. Mobile systems involve a variety of physical and logical entities (devices, computers, servers, code) which can be mobile or static. An easy way to specify the location of such mobile entities is to identify the static entities which are hosting them. In the case of mobile phone networks for instance, mobile phones can migrate from one cell to another, so cell identifiers can be used to specify mobile phone locations.
In our case study, we focus on motions which can lead to location changes. We do not consider the motion of a mobile phone inside the same cell. As [18], we contend that portability does necessarily mean mobility [18].

As a result of mobility, the state of the mobile entity itself and those of the source and target hosts can be modified. Consequently we can identify two realistic situations:

- When moving, entities have no way to guess their new location, nevertheless their hosts have the ability to detect and identify going and coming entities. This justifies the introduction of the notion of dummy location ($\perp$) denoting that the location of a mobile is undefined. This is the case of a mobile which left a host, but not yet joined a new one.
- When moving, entities may be 'aware' of their target location. This is for instance the case of pieces of code stored in servers when dynamically downloaded by client code. In this case the notion of dummy location is not required, since the target location is known.

4. Adding Mobility to Z

Z is a formal specification notation based on set theory and first order predicate logic. It has been developed at the Programming Research Group at the Oxford University Computing Laboratory and elsewhere since the late 1970s. It is used by industry as part of the software (and hardware) development process in Europe, USA and elsewhere. In combination with natural language, Z can be used to produce structured powerful specifications. We may reason about these specifications using the proof techniques of mathematical logic [19, 16]. Z is not intended for the description of non-functional properties, such as reusability, performance, size, and reliability. It is neither intended for the description of timed, concurrent or mobile computing. In the Z notation there are two languages: the mathematical language and the schema language. The mathematical language is used to describe various aspects of a design: objects, and the relationships between them. The schema language is used to structure and compose descriptions. A Z specification typically defines a number of state and operation schemas. A state schema groups together variables and defines the relationship that holds between their values. At any instant, these variables define the state of that part of the system, which they model. An operation schema

defines the relationship between the "before" and "after" states corresponding to one or more state schemas [19]. An operation may be described by including both states: the state before (i.e., *State)* and the state after (i.e., *State'*).

Operation

State
State'

---

predicates on state variables

In the following, we show how the notion of location schemas and mobile operation schemas allow enlarging Z in order to cope with physical and logical mobility in an explicit way. Location schemas involve a distinguished variable whose value specifies the current location of a mobile entity. Our operation schemas may be coupled with usual Z operations to update the state schemas associated to mobile entities as well as their last and new hosts. The suggested location and operation schemas are called mobile Z schemas. The fact that Z may be combined with natural language reinforces the motivation for our choice of Z notation for this purpose.

Mobile operation schemas may define the relationship between the "before" and "after" states, as in the previous operation, but first of all, "before" and "after" locations, or both "before" and "after" states and locations, as illustrated by the following two operations.

Operation

Location
Location'

---

predicates on location variable

Operation

State
Location
State'
Location'

---

predicates on state and location variables

Mobile Z schemas include the following two conventions. Δ *Location* to describe operations that may change only the location, and Δ *State&Location* to describe operations that may change the state and the location at the same time.

Δ *Location*

Location
Location'

Δ *State&Location*

State
Location
State'
Location'

Mobile Z schemas also include the following two conventions. Ξ *Location* to describe operations that do not affect the location, and Ξ *State&Location* to denote operations that do not affect the state or the location. θ Location is the characteristic binding of Location and θ State is the characteristic binding of State.

Ξ *Location*

Δ *Location*

θ *Location* = θ *Location'*

Ξ *StateLocation*

Δ *State*
Δ *Location*

θ *State* = θ *State'*
θ *Location* = θ *Location'*

We identify three mobile operations, the generic forms of which are defined in the remaining part of this section.

Move
This operation updates the Location Schema. A new value is allocated to the location attribute defined in the Location Schema.

Sense* and *Locate
These two operations are mainly used in the situation where the mobile entity moves towards an unknown location. They allow to detect if the mobile entity is currently moving or not, and are also used to determine the target location of the mobile entity in direct (*locate*) or indirect (*sense*) way.

- ***Locate*** takes the identity of the mobile entity (specified in the state schema modeling the entity) as an input parameter and returns either $\perp$ (the mobile entity is still in movement) or an actual location value (the mobile entity did not move from its last location or has already reached its new location).
- ***Sense*** senses if the mobile entity is moving away from its current location thus moving towards a new location. So the new location of the mobile entity can be derived from the result of this operation. The operation *Sense* is not used in the case study presented in section 5. However an example illustrating the usefulness of this operation may be found in [2].

5. Case Study: A Mobile Phone System

5.1 Background

A mobile phone system [7] involves a collection of mobile phones distributed over a set of disjoint spatial cells. At a given instant a mobile phone belongs to only one cell, and any cell hosts a (eventually empty) subset of mobile phone devices. Mobile phones can move from one cell to an adjacent one, without causing any service interruption. Each cell owns a base station which is, on one side in contact with the hosted mobile phones, and on the other side is wired to a central office. A complete study necessitates the specification of the procedures *Call Making*, *Answering*, and *Disconnection* [1]. In the current study we focus on the *Handover* procedure, noting that [7] is treating this procedure in such a way that mobility dimension is merely bypassed.

When a mobile phone crosses a cell boundary during a call, it must switch to another frequency available at the new cell. This is handled by a *HandOver* protocol which consists of:

- Assigning the mobile phone to the new cell,
- De-allocating the frequency which was allocated to the mobile by the old cell,
- Allocating a frequency to this mobile by the new cell.

5.2 Mobile Phone System Specification

Let *CellId*, *MobileId*, *FreqId* and *PhoneId* be the basic types of our mobile system specification, where *CellId* denotes the cell identifiers, *MobileId* denotes the mobile phones identifiers, *FreqId* denotes the frequencies to be allocated, *and PhoneId* denotes all valid phone numbers for fixed and mobiles phones devices.
[*CellId, MobileId, FreqId, PhoneId*]

Let *LocId* denote the identifiers of mobile locations. A mobile location is either the location of a cell to which the mobile is currently "attached", or $\perp$, that is, the mobile has left its current cell but has yet not joined a new one.

$LocId = = CellId \cup \{\perp\}$

The states of a mobile are specified by a free type *State* declared as follows.
State::= idle | paged | ringing | paged_access | call_access | offhook

In the remaining part of this section, we first describe the state and location schemas of the main entities (Mobile, Cell, Mobile Phone System) and then the associated mobile operation schemas.

5.3 State and Location Schemas

5.3.1 Mobile

A mobile can be described by two schemas: *Mobile_State* giving the mobile overall state, and *Mobile_Location,* giving the mobile location

Mobile

ms : Mobile_State
ml : Mobile_Location

Mobile_State involves three attributes. *id* denotes the mobile phone identifier (unique and constant), *state* denotes its current state, and *freq* denotes a set of frequencies which is either empty (no frequency has been allocated to the mobile phone) or involves just one frequency (the one allocated to the mobile by the corresponding cell).

Mobile_State

$id : MobileId$

$freq : \mathbb{P} FreqId$; *frequency allocated to the mobile phone*

$state : State$; *mobile phone state*

$\#freq \leqslant 1$; *number of frequencies less than or equal to 1*

$freq \neq \emptyset \Leftrightarrow state \in \{ ringing, offhook \}$

Mobile_Location involves only one attribute the value of which denotes the current location of the mobile.

Mobile_Location

$Location : LocId$

$Location \in CellId \vee Location = \perp$

$\perp \notin CellId$

Hence, a mobile is located at a given cell or is crossing the border between two adjacent cells. Moreover the border between two adjacent cells does not belong to any of them.

5.3.2 Cell

A cell has four components: *id* denoting its unique identifier, *freqs* a constant predetermined set of frequencies denoting the frequencies available to the cell for allocation to mobile phones, *alloc* denotes the partial function (injection) relating the allocated frequency to the corresponding mobile, and *access* denotes the set of mobile identifiers which is either empty (when the cell has not been seized) or contains just one identifier corresponding to the (unique) mobile that has seized the cell.

$$
\begin{array}{|l}
\textit{Cell} \\
\hline
id : CellId \\
alloc : FreqId \rightarrowtail MobileId \\
access : \mathbb{P} MobileId \\
freqs : \mathbb{P} FreqId \\
\hline
\mathrm{dom}\ alloc \subseteq freqs \\
\# access \leqslant 1 \\
ran\ alloc \cap access = \varnothing \\
\hline
\end{array}
$$

5.3.3 Mobile Phone System

A mobile phone system *MP_S* has three components: *area* denoting a region comprising the set of adjacent cells involved in this mobile phone system, *guests* denoting the set of mobile phones that may be hosted by this region and *located* denoting *a* function which records the location of the mobiles belonging to *guests.* Neither two different cells, nor two different mobile devices can have the same identifier.

A mobile of *guests* is either located at a given cell of *are*a or moving towards another cell, that is its new location is not yet identified.

$$
\begin{array}{|l}
\textit{MP\_S} \\
\hline
area : \mathbb{P} Cell \\
guests : \mathbb{P} Mobile \\
located : guests \rightarrow LocId \\
\hline
\forall m \in guests \bullet (\ \exists c \in area \bullet located\ m = c.id\ ) \vee located\ m = \bot \\
\forall c_1, c_2 : Cell \bullet c_1.id = c_2.id \Rightarrow c_1 = c_2 \\
\forall Ms_1, Ms_2 : Mobile\_State \bullet Ms_1.id = Ms_2.id \Rightarrow Ms_1 = Ms_2 \\
\hline
\end{array}
$$

5.4 Operation Schemas

5.4.1Locate

Locate

Ξ *Mobile_Location*
m : Mobile
Ms : Mobile_State
Ml : Mobile_Location
mid ? : MobileId
l! : LocId ; new mobile location

Ms = m.ms
Ml = m.ml
Ms.id = mid?
(l! = Ml.Location) $\vee$ *(l!* $\neq$ *Ml.Location)* $\vee$ *(l! =* $\perp$*)*

Locate operation outputs the new location (*l!)* of a selected mobile (*mid?).* Therefore, three situations are expected: the pointed mobile is still at its last location, at the border of two adjacent cells or has already joined a new cell. Locate does not affect the variable *Location* that is still recording the last known location of the mobile.

5.4.2 Move

Move

Δ *Mobile_Location*
m : Mobile
Ms : Mobile_State
Ml : Mobile_Location
mid? : MobileId
l? : LocId ; new location

Ms = m.ms
Ml = m.ml
Ms.id = mid?
Ml.Location' = l?

Move updates the variable *Location*, by a value giving the new position of the mobile. When a mobile changes its location, this change has to be reflected onto the overall state of the mobile phone system MP_S. This is performed by an operation called *Update_MP_S* defined by the following schema.

5.4.3 Update_MP_S

Update_MP_S

ΔMP_S
m: Mobile
Ms : Mobile_State
Ml : Mobile_Location
mid? : MobileId
l? : LocId ; *new mobile location*

$m \in guests$
$Ms = m.ms$
$Ml = m.ml$
$Ms.id = mid?$
$located' = located - \{ m \mapsto Ml.Location\} \cup \{ m \mapsto l?\}$

The operations *Move*, *Update_MP_S*, and other operations such as *HandOff* and *HandOn* procedures are triggered by an operation called *Switch* defined by the schema shown below. *HandOn* allocate*s a* new frequency to a mobile joining a new cell, while *HandOff* de-allocates the last allocated frequency to a mobile that left its previous cell. The formal specification of the two operations is not reported in this paper, since they are not tightly related to the mobility aspects in which we are currently interested in. A full description of the corresponding schemas is however available in [7].

5.4.4 Switch

(Ml.Location =⊥) means that the selected mobile is joining a new location by crossing the border common to two cells, while *(l? ≠⊥)* means that the mobile has reached its new location (a new cell).

(Ml.Location ≠⊥) means that the selected mobile is leaving its previous location which is a given cell, while *(l? =⊥)* means that the selected mobile has reached its new location which is the common border of two cells.

Switch

Δ *Mobile_State*
Δ *Mobile_Location*
m : Mobile
Ms : Mobile_State
Ml : Mobile_Location
mid? : MobileId ; *mobile phone engaged in the handover procedure*
l? : LocId ; *its new location*

$Ml = m.ml$
$Ms = m.ms$
$Ms.id = mid?$
$(l? \neq \perp) \wedge (Ml.Location = \perp) \Rightarrow Move \wedge HandOn \wedge Update\_MP\_S$
$(l? = \perp) \wedge (Ml.Location \neq \perp) \Rightarrow Move \wedge HandOff \wedge Update\_MP\_S$

5.4.5 Select_Mobile

This operation allows to select an arbitrary mobile (from *guests*) on which the *HandOver* procedure will be performed.

Select_Mobile

Ξ *MP_S*
m : Mobile
Ms: Mobile_State
mid! : MobileId

$\exists m \in guests \bullet (Ms = m.ms) \wedge (Ms.id = mid!)$

5..4.6 Forward_Mobile_Identity

For the sake of clarity, we need an intermediate operation allowing just forwarding the identity of a selected mobile. This operation is specified by the following schema.

Forward_Mobile_Identity

Ξ *Mobile*
mid? : MobileId
id ! : MobileId

---

id ! = mid?

5.4.7 HandOver

The *HandOver* procedure consists of: first selecting a mobile, localizing it, and then performing the necessary updates.

HandOver $\widehat{=}$ *Select_Mobile* >> (*Locate* $\wedge$ *Forward_Mobile_Identity*) >> *Switch*

6. Proofs

One of the major strengths of Z is that it provides means to perform proofs of properties in a straightforward way. A lot of significant properties related to the mobile phone system can be proven. A proof of interest is that:

A mobile phone cannot be located in two different cells at the same time.

$\forall$ m: Mobile, $\forall$ Ml: Mobile_Location $\bullet$ (m.ml= Ml) $\wedge$ (Ml.Location = l_1) $\wedge$ (Ml.Location = l_2) $\Rightarrow$ $l_1 = l_2$

Another property of interest is that different cells cannot allocate frequencies to the same mobile [7] :
$\forall$ c_1, c_2 : cell $\bullet$ $c_1 \neq c_2 \Rightarrow$ ran c_1.alloc $\cap$ ran c_2.alloc = $\varnothing$

Let us show in the following how to prove the first property. The second one may be proved in a similar way.

Because mobile locations are distinct (disjoint) cells (l_1, l_2,.., l_x ..) or borders ($\perp$) between adjacent cells, we may prove our property by structural induction [19].
Our inductive hypothesis is described by the following predicate **P**:

P_ : ℙ LocId

$\forall$ l : LocId • **P** l $\Leftrightarrow \forall$ m : Mobile, $\forall$ Ml : Mobile_Location, $\forall \lambda$: LocId •
(m.ml = Ml) $\wedge$ (Ml.Location = l) $\wedge$ (Ml.Location = λ) $\Rightarrow$ l = λ

and the proof proceeds as follows:

P $\perp$ **[lemma$_1$]** **P** l_x **[lemma$_x$]**

[induction]

$\forall$ l : LocId • **P** l

[axdef]

$\forall$ l : LocId • $\forall \lambda$: LocId, $\forall$ m : Mobile, $\forall$ Ml : Mobile_Location •
(m.ml = Ml) $\wedge$ (Ml.Location = l) $\wedge$ (Ml.Location = λ) $\Rightarrow$ l = λ

[law of $\forall$]

$\forall$ l, λ : LocId, $\forall$ m : Mobile, $\forall$ Ml : Mobile_Location •
(m.ml = Ml) $\wedge$ (Ml.Location = l) $\wedge$ (Ml.Location = λ) $\Rightarrow$ l = λ

Lemma$_1$

$\forall$ l : LocId • $\forall$ m : Mobile, $\forall$ Ml : Mobile_Location •
(m.ml = Ml) $\wedge$ (Ml..Location = l) $\wedge$ (Ml.Location = $\perp$) $\Rightarrow$
(l! = Ml.Location) $\vee$ (l! $\neq$ Ml.Location) $\vee$ (l! = $\perp$)

[law of $\vee$]

$\forall$ l : LocId • $\forall$ m : Mobile, $\forall$ Ml : Mobile_Location •
(m.ml = Ml) $\wedge$ (Ml.Location = l) $\wedge$ (Ml.Location = $\perp$) $\Rightarrow$ l = $\perp$

The following formula **(1)** is true according to predicate stated in the operation *Locate*.

(1) $\forall$ l: LocId $\bullet$ $\forall$ m: Mobile, $\forall$ Ml: Mobile_Location $\bullet$ (m.ml = Ml) $\wedge$ (Ml.Location = l) $\wedge$ (Ml.Location = $\perp$) $\Rightarrow$ (l! = Ml.Location) $\vee$ (l! $\neq$ Ml.Location) $\vee$ (l! = $\perp$)

□

Lemma$_x$

$\forall$ l : LocId $\bullet$ $\forall$ m : Mobile, $\forall$ Ml : Mobile_Location $\bullet$
(m.ml = Ml) $\wedge$ (Ml.Location = l) $\wedge$ (Ml.Location = l_x) $\Rightarrow$ l = l_{x*}

According to the predicate (l! =Ml.Location) $\vee$ (l! $\neq$ Ml.Location) $\vee$ (l! = $\perp$), the mobile of interest :

(a) is still at its last location , and in this case we have
$\forall$ l : LocId $\bullet$ $\forall$ m : Mobile, $\forall$ Ml : Mobile_Location $\bullet$
(m.ml = Ml) $\wedge$ (Ml.Location = l) $\wedge$ (Ml.Location = l_x) $\Rightarrow$ l = l_x

(b) has changed its location, and in this case we have
$\forall$ l : LocId $\bullet$ $\forall$ m : Mobile, $\forall$ Ml : Mobile_Location $\bullet$
(m.ml = Ml) $\wedge$ (Ml.Location' = l) $\wedge$ (Ml.Location' = l_x) $\Rightarrow$ l = l_x

□

7. Concluding Remarks

In this paper we showed how to extend the Z schema language by a location schema allowing us to deal explicitly with the notion of mobility. Compared to similar contributions [8, 10, 12, 13, 14, 15, and 17], our approach seems to be simpler but still applicable to comparable "benchmark" case studies. The benefits of our mobile Z framework were shown using the benchmark case study of mobile phone system. Although the case study appears to "favor" one category of mobile computing: the nomadic networks, the notion of dummy location and the mobile-location component of a mobile device appears to be quite general and may be applied to other types of mobile computing. This will be dealt with in future case studies. Although the focus of this paper is not on proof techniques we showed very briefly how our framework may be used in proving some of the useful properties such as the fact that a mobile device cannot be located in two different cells at the same time. Proof techniques concerned with mobile Z will be a major concern of our future work.

Acknowledgements

This work is fully supported by the Deanship of Scientific Research at Philadelphia University.
We would like to thank Reiko Heckel who pointed to us the existence of a related work [17] by K. Taguchi, and J. S. Dong, entitled: An Overview of Mobile Object-Z.

References

1. M. Bettaz, M. Maouche, Mobile Z notation, Research Report, Department of Computer Science, Philadelphia University, March 2002. http://www.philadelphia.edu.jo/it/bettaz.pdf
2. M. Bettaz, M. Maouche, R. Heckel, Graph Transformation: A Semantic Framework for Mobile Z, European Joint Conferences on Theory and Practice of Software (ETAPS-WADT), Barcelona, Spain, Mars-Avril 2004
3. A. Lopes, J.L. Fiadeiro, Adding Mobility to Software Architectures, Proceedings 2nd Workshop on Foundations of Coordination languages and Software Architectures, FOCLASA'03, Electronic Notes in Theoretical Computer Science 97, 2004.
4. C. Mascolo, G.P. Picco, G-C. Roman, CODEWEAVE: Exploring Fine-Grained Mobility of Code, Special Issue on Distributed and Mobile Software Engineering, Automated Software Engineering, June 2004, vol. 11, issue 3 Kluwer Academic Publisher
5. D. E. Comer, Computer Networks and Internets, Prentice Hall, Third Edition, 2001.
6. G. Denker, J. Meseguer, and C. Talcott, Formal Specification and Analysis of Active Networks and Communication Protocols: The Maude Experience, DISCEX'00, IEEE Computer Society Press: 251-265, 2000.
7. R. Duke, P. King, G. Rose, and G. Smith, The Object-Z Specification Language, Department of Computer Science, University of Queensland 4072, Australia, 1991F. Duran, S. Eker, P. Linkoln, and J. Meseguer, Principles of Mobile Maude, Springer-Verlag LNCS 1882: 73-85, 2000.
8 J.J. Kidtler, M. Satyanarayanan, Disconnected operation in the Coda file system ACM Transactions on Computer Systems, 10(1): 3-25, 1992.
9. R. Milner, Communicating and Mobile Systems: The π-calculus, Cambridge University Press, 1999.
10. A.L.Murphy, G.-C. Roman, G. Varghhese, An Exercise in Formal Reasoning about Mobile Communications, Department of Computer Science, Washington University in St. Louis, 1998.
11. G. P. Picco, G. –C. Roman, and P. J. McCann, Reasoning About Code Mobility with Mobile Unity, Washington University in St. Louis, 1999.
12. G. –C. Roman, Mobile UNITY: Reasoning about Physical and Logical Mobility, PowerPoint Slides, Washington University in St. Louis, August 2002.
13. G. –C. Roman, A. L. Murphy, and G. P. Picco, A Software Engineering Perspective on Mobility, WUCS-99-31, Washington University in St. Louis, December 1999.

15. G.-C. Roman, P.J. McCann, and J.Y. Plun. Mobile UNITY: Reasoning and Specification in Mobile Computing. ACM Transactions on Software Engineering and Methodology, 6(3): 250-282, 1997.
16. J. M. Spivey, the Z Notation: A Reference Manual, Second Edition, Programming Research Group University of Oxford, 1998.
17. K. Taguchi, J. S. Dong, An Overview of Mobile Object-Z, Springer-Verlag LNCS 2495: 144-155, 2002.
18. S. Tanenbaum, Computer Networks, Third Edition, 1996.
19. J. Woodcock, J. Davies, Using Z: Specification, Refinement, and Proof, Prentice Hall, 1996.