Report Number: WUCS-95-25

1995-01-01

# Reasoning about Program Interactions in the Presence of Mobility

Gruia-Catalin Roman and Peter J. McCann

Mobile computing is emerging as an important new paradigm which has the potential to reshape our thinking about distributed computation. Mobility has far-reaching implications on what designers and users can assume about communication patterns, resource availability, and applciation behaviors as components move from one location to another while joining or leaving groups of other components in their vicinity. New distributed algorithms are likely to be required as the nature of applications shifts with the emergence of this new kind of computing environment. Formal methods have an important role to play in the midst of these developments both in terms... Read complete abstract on page 2.

# Reasoning about Program Interactions in the Presence of Mobility

Gruia-Catalin Roman and Peter J. McCann

Complete Abstract:

Mobile computing is emerging as an important new paradigm which has the potential to reshape our thinking about distributed computation. Mobility has far-reaching implications on what designers and users can assume about communication patterns, resource availability, and applciation behaviors as components move from one location to another while joining or leaving groups of other components in their vicinity. New distributed algorithms are likely to be required as the nature of applications shifts with the emergence of this new kind of computing environment. Formal methods have an important role to play in the midst of these developments both in terms of helping the research community better understand fundamental issues germane to mobile computing and by providing pratical solutions to difficult design problems.

# Reasoning about Program Interactions in the Presence of Mobility

Gruia-Catalin Roman
Peter J. McCann

WUCS-95-25
Position Paper

1 August 1995

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

# Reasoning about Program Interactions in the Presence of Mobility

(Concurrency and Distribution—Position Paper)

**Gruia-Catalin Roman**
**Peter J. McCann**

## Abstract

Mobile computing is emerging as an important new paradigm which has the potential to reshape our thinking about distributed computation. Mobility has far-reaching implications on what designers and users can assume about communication patterns, resource availability, and application behaviors as components move from one location to another while joining or leaving groups of other components in their vicinity. New distributed algorithms are likely to be required as the nature of applications shifts with the emergence of this new kind of computing environment. Formal methods have an important role to play in the midst of these developments both in terms of helping the research community better understand fundamental issues germane to mobile computing and by providing practical solutions to difficult design problems.

**Correspondence**: All communications regarding this paper should be addressed to

Dr. Gruia-Catalin Roman
Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

office: (314) 935-6190
secretary: (314) 935-6160
fax: (314) 935-7302

http://swarm.cs.wustl.edu/~roman/
roman@cs.wustl.edu

## Background

Recent advances in miniaturization and wireless communication are creating a powerful and flexible new computing paradigm known as "mobile computing." For our purposes, this term can be broadly defined as any wireless networking technology that allows for communication between pairs of computing hosts where at least one is able to move freely throughout a given coverage area. This type of network yields an important advantage over other networking technologies: it allows the user to bring computing resources quickly and flexibly to any location, without taking the time or paying the cost to set up a more traditional wired connection, and to cost-effectively move those resources to another location when necessary. This flexibility can greatly increase the efficiency and productiviy of the people and organizations using those resources. Also, speed and flexibility of resource deployment are absolutely essential in a crisis management or battlefield scenario, and mobile computing may provide solutions for the information needs of these types of environments.

Because wireless communication is characterized by frequent voluntary and involuntary disconnections and configuration changes, designers of mobile systems are faced with unprecedented technical challenges particularly in the areas of dependabilty and modularity. Without dependability, the benefit of the added flexibility is lost. In an essentially open system with components moving freely from one location to another, modular design must ensure the correctness of components over a very broad range of environments and must facilitate adaptability to changing resource availability and computing/communication contexts. We contend that achieving dependability will require a greater reliance on pragmatically-minded formal methods as well as on novel design strategies. Furthermore, we believe that the notion of modularity and composition must be reexamined, both at the formal level and from a practical perspective, in light of the presence of mobility. Finally, we are convinced that location will emerge as a key concept both in reasoning about and in the design of certain classes of mobile systems. These and related issues define the main themes of this position paper.

Much mobile computing research has been focused on relatively narrow issues. These include building mobility support into the Internet Protocol (IP) [4, 5] and distributed operating systems issues such as memory and file management [7, 9]. At the applications level, researchers are attempting to predict what new kinds of applications will be demanded by users of mobile computing [1, 10] and are trying to produce new design practices and methodologies to meet the software engineering challenges brought on by mobile computing [2, 8]. A key question is how to produce mobile-aware applications quickly and dependably. Researchers are just now beginning to recognize the importance of finding the right language-level abstractions [2, 6]. Our hope is that the right abstractions, coupled with pragmatically-minded formal treatment of problems, can help the developer meet these requirements.

Our reasearch goal is to understand the fundamental issues facing mobile computing and to develop practical design solutions for the flexible and rapid deployment of mobile applications. Towards this aim, we are involved in the development and evaluation of new models, constructs, design techniques, and distributed algorithms.

## Formal Model

**Research issues.** A formal treatment of mobility requires an appropriate model, an associated notation, and the means to analyze programs and specifications. Much of the formal arsenal can be borrowed from existing work on distributed computing but it must be augmented to account for the distinctive features of mobile computing. The representation of, operations permitted upon, and ways to reason about the location of programs are some of the most basic issues we face in this endeavor. They affect the manner in which one specifies properties of mobile programs, codes mobile applications, and verifies their correctness. Related to location and movement is the issue of how best to model the transient and location-dependent nature of the interactions among programs. To complicate things further, interactions may be considered at different levels of abstraction and mobile computations may have to be treated as open systems in which new programs may come into existence or disappear at will. We favor an application-oriented perspective and we are investigating modes of interaction that simplify the rapid development and deployment of applications.

Two issues which many consider to be central to mobile computing are the power and bandwidth limitations. While we acknowledge their practical significance and we expect them to impact our work on distributed mobile algorithms, we do not see these issues as fundamental but more akin to architectural and performance constraints in traditional distributed computing. One analyzes the power and bandwith implications of a particular design choice without employing a programming notation in which these concepts appear explicitly. This is also consistent with the way in which performance and computational complexity are treated today.

**Base model.** In selecting a model to build upon, we believe that it is important to consider one that is visible within the distributed computing community, is least likely to introduce modeling artifacts, has well understood analytical methods, and facilitates direct transfer of results to other established models. There are about half a dozen models of concurrency that might be considered to meet these requriements. We favor UNITY [3], a model originally proposed by Chandy and Misra. The UNITY programming notation reduces distributed computing to a very small number of constructs: conditional multiple-assignment to shared variables and weakly-fair statement selection. The UNITY proof logic is simple, a specialization of temporal logic motivated by the desire to extricate proofs directly from program text in the tradition of sequential programming. Any results obtained in UNITY will be directly relevant for most state-based models (e.g., temporal logic, TLA, etc.). From a practial perspective, another attractive feature of choosing UNITY is the relatively easy (almost mechanical) implementation path from UNITY to traditional programming languages. In the remainder of this section we present some preliminary ideas regarding the directions we intend to pursue towards adapting UNITY to mobile computing.

**Expressing location and movement.** The UNITY model is a state-based approach to programming with an interleaved execution model. The following trivial program, for instance, initializes variables $x$ and $z$ (to an arbitrary value and zero, respectively) and places the sum of the two initial values into $z$.

```
program Sum
    declare      x, z : integer
    initially    z = 0
    assign       z := x + z || x := 0
end
```

The parallel bar combines two assignment statements into one. A box ($\Box$) is used to separate single statements when more then one is present.

By analogy with real-time modeling where time is most often introduced as a distinguished variable not accessible to the programmer, one can capture mobility by introducing a distinguished location variable for each individual program. The program above could be viewed as mobile by simply attaching a location variable (which might be named $\lambda$ by convention) and an initial location (if known) in the program declaration. A mobile version of the program above may be generated by simply writing

**program** *Sum* **at** $\lambda$

which indicates that the initial value of the location variable $\lambda$ (actually *Sum*.$\lambda$) is arbitrary. Restrictions on how such a location variable is accessed and updated would have to reflect the characteristics of the computation. Because the code in *Sum* does not change $\lambda$, it must be assumed that *Sum* can move only under outside control (to be specified) or has a fixed position. Moreover, *Sum* is neither aware of its movement nor able to control it, implicitly or explicitly. This is usually the case in a cellular network, for instance, where the location of the mobile units is determined by the car or person carrying the computer but is constrained to movements from one cell to a neighboring one (as long as the unit is on). The verification of any hand-off algorithm must rely on this assumption. Protocols involved in reestablishing connectivity at the time a mobile computer is powered up may have to assume that initial locations are arbitrary. In some applications a program may have to know its own location and behave accordingly while not in others. In the former case the location is directly accessible by the program while in the latter the location plays a role only in reasoning about the computation. As automation technology advances it is also conceivable that certain programs may have the ability to actually control the movement of their carrier which may be, for example, a robot doing deliveries in an office building. The notation introduced so far is the simplest we can conceive of and we plan to evaluate its implications further.

**A modular notation for transient interactions.** We turn next to the question of how programs interact with each other. UNITY provides two forms of interaction or composition, *union* and *superposition*. The union of two programs is defined as an interleaved execution of all assignment statements in the two programs in a state space consisting of all variables of the two programs—variables having identical names become shared. Program $a$ is said to be superposed on program $b$ if every statement of $a$ is synchronized with some statement of $b$ and no statement of $a$ writes to (but can read) any variable of $b$. Inference rules for deriving the properites of a composite program from those of the components are also available. Both forms of composition are static in nature. Our intent is to revisit them from a mobile perspective, i.e., we are interested in the implications of a model where programs share variables or synchronize statements only under ceratin conditions usually having to do with proximity. If successful, we will provide both a novel generalization of the two UNITY methods of program composition and also some interesting new constructs for interactions among mobile programs. Of course, these are not the only forms of interactions we plan to explore, only the most direct extensions of UNITY. A more careful analysis of mobile applications will be used to identify other interesting candidates.

Success will be determined by a number of formal and pragmatic considerations. They include the development of inference rules for proving properties of composite computations from those of the individual mobile programs, an assessment of the extent to which these new constructs actually simplify the writing of mobile applications, the development of efficient implementations, etc. These are issues for future research, but in order to provide the reader with a more concrete example of the kind of constructs we are investigating at this time we consider next a simple illustration of how one might augment UNITY with transient variable sharing capabilites. For this purpose we revisit the mobile version of the *Sum* program above and transfrom it into a roving *Reader* which sums up the electric consumption in a building. This is done by collecting readings from electric meters located around the building. When near an electric meter, the variable $x$ is shared with its counterpart in the meter, call it $n$, and its value may be added to the total and reset to zero in the same step. The code for the *Reader* is obtained by tagging the variable $x$ as shared.

```
program Reader at λ
    declare      x : shared integer
                 z : integer
    initially    z = 0
    assign       z := x + z || x := 0
end
```

The code for the electric meters may assume the form

```
program Meter(i) at λ = loc(i)
    declare      n : shared integer
    initially    n = 0
    assign       n := n + 1
end
```

where loc($i$) is the unique location of the $i$th electric meter. (We assume that all variables have uniques names by implicitly extending their declared names with the program name which is unique by convention.)

The remaining problem is how to tie $x$ and $n$. We want the sharing to take place only when the *Reader* is at the same location with some *Meter*($i$) and it is desirable to hide the presence of sharing from both participants. For this purpose we add an **Interactions** section that describes the transient sharing among variables belonging to different programs.

```
Interactions
    Reader.x ≈ Meter(i).n
    when  Reader.λ=Meter(i).λ
        engage      Meter(i).n
        disengage   Reader.x = 0 || Meter(i).n = Meter(i).n
```

Here we consider only pairwise sharing but the approach generalizes to an arbitrary number of programs. The predicate following **when** specifies the conditions under which the sharing takes place, when the location of the two programs is the same. The **engage** clause specifies what value to take on when the variables become shared, the value of the electric meter. In general, this expression would reconcile two or more possibly distinct values. The **disengage** clause specifies what value each variable is left with when the transient period of sharing is over: $x$ is reset to zero and $n$ keeps the shared value—it may be its original count if the summation did not take place in the *Reader*. These statements can encapsulate data hoarding and re-integration policies like the ones in [7] and they are more general than strategies for keeping files coherent in that we make no assumptions about the values of the variables before integration. So far, small examples we worked out (e.g., electronic mail, notification propagation) have shown the potential for a highly decoupled style of computing in which programs behave correctly with little or no knowledge of the context in which they find themselves.

**Proof logic implications.** Because locations are simply variables, reasoning from first principles about the behavor of a mobile computation such as the one sketched above is relatively straighforward. However, the assumed atomic nature of the engagement and disengagement protocol can bring about significant complications, especially when one considers complex patterns of variable sharing among multiple programs, the transitivity of the sharing relation, and the potential for simultaneous engagements and disengagements. At this point we have a good grasp of the complexity of these issues. Future work will need to address many important and challenging technical issues. They include the development of inference rules similar to the ones existing today for union and superposition and the study of specialized interaction forms that are easy to verify and implement.

## Design Schemas

Because verifying a completed program is very costly, our prior research has pursued program derivation as a more promising alternative. We plan to continue our investigation into pragmatic design strategies but turn our attention to the use of application-driven design schemas, both of a syntactic and semantic nature. We hope to identify schemas that can be readily used by the average designer. The above meters/reader program is one example of what we mean by a schema. The program does not make use of the full generality of transient interaction, and this can make proofs simpler. For instance, the interactions are limited to pairwise variable sharing. Because there is never a group of three or more variables that get shared, we don't have to worry about the transitive quality of variable sharing. Note also that the disengagement value does not change one of the variables, and sets the other variable to zero. This preserves several important properties of the program, most notably the fact that no spurious values are introduced into the summation.

## Distributed Algorithms

Traditional distributed algorithms are designed for networks of static hosts. The assumptions made by these algorithms are often not valid in the mobile setting. Mobility challenges us to re-evaluate the implicit assumptions and requirements behind these algorithms and to develop new ones for a novel class of algorithms. The density of mobile components can affect the strategies for route maintenance and a message broadcast, e.g., in highly populated regions a broadcast message can spread like a ripple on the surface of the water with the guarantee that any component that traverses the frontier of the ripple will receive the broadcast message. Movement vectors can be used to optimize information transfers, e.g., predicting where a mobile unit will be and delivering a message to that location for later pickup. When information about the movement pattern of components is available it may help reduce the overall message volume. The impact of these and other factors specific to mobile applications are a central theme of our investigation.

## Conclusions

Our approach to the study of mobile computing is distinctive in its emphasis on formal methods, application-oriented design strategies, and program visualization (not discussed in this paper). The principal research concerns of our project are: (1) a UNITY-like model of mobile computing; (2) a design methodology based on use of schemas and novel modular strategies; and (3) new distributed algorithms targeted to the specifics of mobile applications.

## References

[1] Acharya, A., Imielinski, T., and Badrinath, B. R., "DATAMAN project: Towards a Mosaic-like Location-Dependant Information Service for Mobile Clients," *MOBIDATA: An Interactive Journal of Mobile Computing*, vol. 1, no. 2, 1995.

[2] Badrinath, B. R., and Welling, G., "Event Delivery Abstractions for Mobile Computing," Rutgers University, Technical Report LCSR-TR-242, 1995.

[3] Chandy, K. M., and Misra, J., *Parallel Program Design: A Foundation*, Addison-Wesley, New York, NY, 1988.

[4] Myles, A., and Skellern, D., "Comparing Four IP Based Mobile Host Protocols," *Computer Networks and ISDN Systems*, vol. 26, pp. 349-355, 1993.

[5] Perkins, C. E., Myles, A., and Johnson, D. B., "The Internet Mobile Host Protocol (IMHP)," *Proc. iNET '94*, Prague, Czech Republic, 1994.

[6] Plun, J., and Roman, G.-C., "Transient data sharing among mobile programs," Washington University, Dept. of Computer Science, Technical Report WUCS-95-01, 1995. (under review for IEEE Transactions on Parallel and Distributed Systems)

[7] Satyanarayanan, M., et al., "Experience with Disconnected Operation in a Mobile Computing Environment," School of Computer Science, Carnegie Mellon University, Technical Report CMU-CS-93-168, 1993.

[8] Schilit, B. N., and Theimer, M. M., "Disseminating Active Map Information to Mobile Hosts," *IEEE Network*, IEEE Computer Society, pp. 22-32, 1994.

[9] Tait, C. D., and Duchamp, D., "An Efficient Variable Consistency Replicated File Service," *File Systems Workshop, USENIX, Ann Arbor MI*, pp. 111-126, 1992.

[10] Voelker, G. M., and Bershad, B. N., "Mobisaic: An Information System for a Mobile Wireless Computing Environment," *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, 1994.