RESEARCH SUMMARY

EARLY WORK (1975-1990)

Software engineering has been a central theme of my research throughout my career. The starting point was my dissertation, which demonstrated the potential for dependability and productivity improvements through systematic use of formal methods. Nevertheless, my early research spanned a broad range of investigative domains including biomedical simulation, formal languages, distributed databases, computer graphics, and image understanding. Some of the most interesting results have been those stemming from fruitful collaborations with colleagues in the department: one of the first fully distributed algorithms for concurrency coordination in distributed databases, a novel pipelined architecture for real-time hidden surface elimination, a logic-based model for geographic data processing, and algorithms for real-time stereo matching.

FORMAL DESIGN METHODS (1985-1995)

An important goal of my research has been the industrial application of formal design methods, i.e., methods that have a primarily mathematical rather than empirical foundation. Design methodologies play a critical role in this process by shaping the body of knowledge designers must acquire, the skills they need to master, and the kinds of tools they might employ. In my group, we viewed program derivation (in its many forms) as the principal venue for the study and development of formal design methods. Because we wanted to bridge formal studies and industrial practice, we explored program derivation techniques that were amenable to both formal and semi-formal application. Because we sought to uncover and understand fundamental issues facing concurrent systems design, the models we employed exhibit parsimony and extensions to these models were kept to a minimum.

Our investigation into formal design methods centered around the UNITY model (proposed by Chandy and Misra) and was directed towards extending the applicability of various program derivation techniques to novel design domains. Refinements of the UNITY notation and logic, new program derivation strategies, and detailed case studies have been the principal technical results of this investigation. In several cases, our efforts were among the first of their kind in the respective areas.

The relation between systematic (but informal) design practices and the UNITY-style specification refinement was studied in the context provided by the design of a message router. In an attempt to overcome the static features of UNITY we developed a highly dynamic variant called Swarm. The two models share a common proof logic but the static variables and statements of UNITY were replaced in Swarm by dynamically created tuples and transactions over tuple spaces. Swarm facilitated the introduction of formal verification and derivation techniques to tuple-based languages à la Linda and to rule-based systems à la OPS5. Several case studies involving the derivation of programs meant to execute on specific kinds of parallel or distributed architectures led to the development of an assertional-style technique for formal specification of architectural constraints and to the emergence of a new program derivation strategy that integrates specification and program refinement. Swarm was also instrumental in the study of methods for specifying and reasoning about dynamic forms of synchrony among atomic actions.
PROGRAM VISUALIZATION (1990-2000)

I founded the Computer Visualization Laboratory as a research facility whose mission was to investigate methodologies for integrating computing and visualization, to explore technologies that make feasible the fusion of the two disciplines, and to provide video production and multimedia capabilities for use in research and instruction. Our research on program visualization achieved rapid visualization of program executions, eliminated (often completely) the need to access the program code when building a visualization, offered unprecedented expressive power with minimal notation, and provided several capabilities important to the introduction of program visualization into the industrial setting. The notion of declarative visualization, embodied in the design of a distributed system called Pavane, was the key technical contribution that enabled us to achieve these goals and a high degree of visibility. The declarative paradigm, pioneered by our group, treats visualization as a mapping from program states to animated three-dimensional scenes. An extensive evaluation of Pavane in a wide range of settings (formal design, distributed algorithms, scientific visualization, requirements validation, monitoring, debugging, etc.) showed that complex animations of system behaviors can be constructed within three to six hours, often using less than two dozen declarative rules. In addition, Pavane offers a broad range of navigational capabilities to viewers and animators as well as through-the-screen interactions. The last project associated with the laboratory was the visualization of executing distributed computations using a method called query-based visualization. It treated a distributed computation as an evolving database with queries being submitted using graphical representations and with the results being presented as real-time animations of the system state.

MOBILE COMPUTING (1995-present)

Formal Models for Mobility. Our research group has been in the forefront of efforts to apply formal design techniques to mobile computing. Sometime in the early nineties, we asked ourselves whether UNITY style specifications and their refinement could be applied in a straightforward manner to the design of mobile systems. It was this investigation that ultimately resulted in the development of Mobile UNITY. In Mobile UNITY the description of a system is separated into mobile programs and interactions. Programs represent the basic unit of mobility and are described using a notation that closely resembles UNITY. The key differences are the fact that variables with the same name appearing in distinct programs are presumed to be distinct and a distinguished variable denoting the location is added to each program. Mobile UNITY uses the location variable to capture movement. Access patterns to the location variable allow one to differentiate among programs which are oblivious to location changes, those which are aware of their current location, and those that control their own movement. By modeling movement as value assignment, the existing UNITY proof logic easily accommodates reasoning about space and motion. Interactions among mobile units are captured using a minimalist coordination language whose constructs allow one to specify asynchronous data transfer, reactive processing, and statement inhibition. Inhibitors provide a mechanism for strengthening the guard of an existing statement without modifying the original. This construct effectively redefines the scheduling mechanism to avoid executing certain statements when their execution may be undesirable. For example, one can prevent a disconnected clock from advancing an unbounded number of steps by inhibiting the statement which increments the clock. Reactive statements provide a powerful mechanism to capture interrupt service mechanisms and to respond to changes in the system, e.g., a loss of connectivity or degradation in the quality of services. Technically, the reactive statements form a program that is scheduled to execute to fixed-point after each individual assignment statement. The expressive power of the coordination language is derived from the constructs’ ability to access variables, including location information, across program boundaries. To evaluate the expressive power of Mobile UNITY, we showed how these constructs could express low-level communication protocols (e.g., serial interfaces), novel high-level coordination constructs (e.g., transiently shared variables and transient statement synchronizations across mobile units), and clock synchronization. Most importantly, we extended the UNITY proof logic to accommodate the new constructs and the desired coordination semantics.

Mobile Protocol Verification. In an effort to show that Mobile UNITY provides meaningful abstractions for real problems in mobility, we used Mobile UNITY to specify and prove the
correctness of Mobile IP, a protocol designed to facilitate the delivery of packets over the Internet to hosts that change the point of connection. We exploited the modular nature of Mobile UNITY by separately specifying the network, mobile agent, home agent, and foreign agent. The interactions section succinctly captured the synchronization and data sharing for the components, including the real-time constraint on the bounded clock drift between a mobile agent and its home agent. The formal verification of the protocol provided insights into how common informal notions of correctness relate to the conditions and assumptions under which the protocol is expected to behave properly.

**Code Mobility.** In a separate case study we explored the applicability of the Mobile UNITY constructs to mobile code. We investigated a single application and compared design solutions that employed various mobile code paradigms, including remote evaluation, code on demand, and mobile agents. The study showed that the constructs of Mobile UNITY, particularly the specification and proof logic, apply to the area of logical mobility. Our foray into code mobility also led to the development of a fine-grained model of code mobility (CodeWeave). The first of its kind, this model allows for the movement of single variables, single statements, or complex block structured programs to locations on other hosts or even at points within other code fragments. Researchers of University College London are currently examining practical implications of this model.

**Middleware for Mobility.** The LIME (Linda In a Mobile Environment) middleware provides a set of abstractions for enabling rapid development of dependable applications that exhibit physical mobility of hosts, logical mobility of agents, or both. LIME extends the Linda model of process coordination to allow tuple spaces to be distributed in time and space and transiently shared when connectivity is available. Movement, logical or physical, results in implicit changes to the contents of the tuple space accessible to the individual components. The system, not the application program, manages the movement of tuples and the tuple space restructuring associated with connectivity changes. LIME provides the application programmer with the standard set of Linda operations as well as a second set of operations dealing with the distribution of the tuples with respect to location. This expanded set of primitives allows the programmer to control the level of transparency of mobility depending on specific application needs. Several applications were developed to demonstrate the effectiveness of the LIME middleware in the development of mobile applications. Furthermore, we partnered with Ford Motor Co. to evaluate the applicability of LIME to vehicle-to-vehicle communication under highway conditions. LIME is available under an open source licensing agreement.

**Algorithms for Mobility.** We have also been active in the development of new algorithms for mobile computing. Some of the work is set in the base station mobility model which is similar to the cellular telephone network with a set of fixed base stations that communicate with the mobile units as they move within a region. Nodes represent mobile base stations and channels represent the communication connections among these base stations. Interestingly, we found that it was possible and advantageous to treat mobile units as messages. We accomplished this by making a small adjustment in the handover protocol to give the appearance that mobile unit movement among base stations takes place in a FIFO manner. This strategy enabled us to convert traditional distributed algorithms to novel ones in the mobile environment. For example, we developed an algorithm for reliable multicasting of a message to all mobile units in a region. The algorithm, based on the Chandy-Lamport distributed snapshot algorithm, has several attractive properties with respect to delivery guarantees in rapidly changing environments and network overhead. A second algorithm, based on the model of diffusing computations by Dijkstra and Scholten, has been developed to track the movement of a mobile unit within a region of base stations. With this tracking information, the broadcast of a message can be limited in range, involving less network traffic while still allowing for rapid mobile unit movement.

**Context maintenance.** Central to our work on middleware for mobility is the notion of constructing and automatically updating contextual information directly determined by physical and logical relations among mobile units. One aspect of this deals with maintaining a consistent view of group membership in ad hoc mobile environments in order to meet the needs of a specific application. For example, an application may restrict group membership such that a message sent from one group member to another is guaranteed to arrive at the destination. Maintaining such a
guarantee is challenging in the mobile setting because mobile hosts can suddenly move out of communication range, becoming unable to receive messages. Our approach ensures that this kind of unannounced disconnection does not occur between group members by detecting and announcing the imminent departure of a mobile unit from the group. A novel aspect of the approach is the reliance on location information in forming and breaking up groups.