XPS: FULL: FP: Collaborative Research: Model-based, Event-driven Scalable Programming for the Mobile Cloud

Gul Agha and Darko Marinov, University of Illinois at Urbana-Champaign
Daniel Jackson, Massachusetts Institute of Technology

1 Introduction

Applications running on mobile devices backed by cloud servers and storage—*mobile cloud apps*—are becoming the dominant paradigm but are not well supported by current programming technology. Developing any of the individual application parts is not hard, but developing an entire system is far more challenging than it needs to be.

At the same time, a major opportunity beckons. The development of cloud technology makes massive computational resources available to all. And yet the current frameworks for developing applications provide no simple way to map user-level application code to these resources in a flexible way.

These are the challenges our project addresses. We propose a new methodology for building mobile cloud applications that can leverage cloud resources in a scalable way, while dramatically simplifying the development effort, thus reducing the cost of their construction and maintenance. With the results of our research, a development team will be able to build and deploy a distributed mobile application that can exploit massive concurrency and data storage with even less work than is required today to build a typical web application.

Our key idea is an instance of one of the best known and most widely applied ideas in computer science, namely *separation of concerns*. On the one hand are the concerns of the interaction designer, who wants rapid and easy development, in which user-level behaviors can be expressed directly and succinctly. On the other hand are the concerns of the system architect, who wants to achieve scalable performance and reliability, by directly controlling how user-level tasks are mapped to computational resources. Characteristically for all separations of concerns, each party not only wants to focus on their own concerns but wants to be able to ignore the other. Thus the interaction designer would like to write code that deals with the semantics of the application data and how actions performed by users affect and are affected by this data, but would like to ignore all the architectural complexities that typically become intertwined with the higher level code (most notably concurrency control and low level communication primitives). Likewise, the architect would like to be able to design and implement load balancing schemes, for example, without getting bogged down in small details of business logic.

Applications built following our methodology will have two distinct levels: a *user level* concerned with the semantics of user interaction, and an *architecture level* concerned with allocation of computational resources. At each level, behavior is expressed in a simple but powerful language that abstracts away many of the burdens of today's technology. At the user level, the language supports declaration of typed relational data and atomic events that read and write the data. The data store is regarded (abstractly) as if it were global and centralized; of course, in the implementation it will be distributed and mapped to different representations (relational databases, key-value stores, in memory data structures, etc.). At the architecture level, the application is defined using the actor model, a distributed computational model in which all communication is mediated by messages between actors.

Our confidence in this methodology rests in part on the solidity of these two computational models. The relational/event model has been used for more than a decade in modeling languages such as Alloy [58] and Event-B; moreover, we have recently built a prototype implementation that supports this model (on top of Rails, a traditional web framework) [100]. The actor model has an even stronger provenance, and has been widely applied as a flexible framework in many application areas [2].

1.1 Intellectual Merit

The novelty of this methodology—and the primary research challenge—is twofold. First is achieving this radical separation of concerns, and bringing the two levels together so each language can be used in its own natural domain, with the two descriptions brought together in a single system. This will require the development of a precise but straightforward semantic mapping between the two languages. It will also involve, at the relational/event level, extension of the language as we encounter the full range of application behaviors that we need to express. Second is automating the mapping between levels. Part of this challenge is like traditional compilation, albeit between non-traditional languages. But what makes this hard (and powerful) is that the system architect will be able to determine the key features of the mapping from actors to computational elements, leveraging her insights about how best to distribute load across machines and memories.

Applications that run on mobile platforms and in the cloud are proliferating, and computation is becoming increasingly distributed and event-driven. Despite advances in the technologies for various components (communications, databases, web servers), writing mobile cloud applications remains hard, and the resulting code, because it mixes different concerns, is complex and hard to modify. The complexity tends to result in bugs, and the difficulty of making architectural changes without a complete rewrite tends to result in poor scalability. Together, a lack of scalability and complexity exacerbated by workarounds damages reliability.

A model-based programming paradigm, specialized for particular domains, would simplify the task. Ideally, a programmer would represent a distributed application as though it were a simple sequential program, with atomic actions updating a single, shared global state. But such a model-based program needs to be translated to a runtime environment that executes it with high concurrency, automatically handling (and hiding from the programmer) complications such as network communication (including server push), queuing and serialization, concurrency and data races, persistent storage of data, and coordination of events while supporting scalability through parallelism and scalability. The actor model provides a natural implementation with these properties.

Our research project will develop the modeling languages, transformation tools, and development environment to enable this new style of development. Using a collection of challenge applications representative of the major classes of important application styles, we will demonstrate the expressiveness, flexibility and succinctness of the user-level language, and the ability of the mapping to the actor language to obtain scalable performance that can exploit high levels of concurrency. The simplicity and uniformity of the user-level language will also enable new forms of analysis, including directed testing, model checking and other forms of verification.

1.2 Team's Qualifications

We have formed a team of researchers with different and distinct expertise relevant to the XPS program focus areas. Our team combines complementary expertise in parallel programming (Agha), modeling languages (Jackson), and software testing (Marinov). Moreover, we have been collaborating for years and published several joint papers [62, 75, 76, 86–88, 100, 122, 142].

Gul Agha has published almost 200 research articles in areas related to parallel and distributed computing. His widely cited book, *Actors: A Model of Concurrent Computing in Distributed Systems* (MIT Press, 1986) [10], has provided a basis for a number of research projects in academia and industry. Agha and his colleagues developed a precise mathematical foundation for open concurrent computing [6, 7]. He has studied the relation of actors to other models of concurrency including Petri Nets [4, 163] and the π-calculus [9, 143, 144]. Agha and his student Svend Frolund developed novel approach for expressing and implementing coordination constraints [41]. Frolund's Ph.D. dissertation was published by MIT Press [40]. Agha applied a variant of this programming model to the problem of modularizing the development of dis-

tributed software which must satisfy real-time constraints [115, 116]. Agha developed models and efficient compilers for large-scale parallel systems [11, 79, 80, 80, 111]. Recently, Dr. Agha and his collaborators showed how actor programs can be optimized in the Java VM by inferring message transfer, to avoid unnecessary copying, and using continuations [72, 106]. Agha and his student Carlos Varela developed the actor programming language SALSA for Worldwide Computing (web applications, grid computing, cloud computing) [149].

Besides research on actor languages and compilers, Agha has worked on methods for improving the robustness of large-scale parallel programs by developing techniques for efficient distributed monitoring [123, 125] and for predictive monitoring [124]. Agha and Marinov, in collaboration with Agha's student Koushik Sen developed a fast and effective technique called *concolic testing* of dynamic programs [122], as well as concurrent programs testing [121, 122]. The technique has been widely applied in testing of hardware and software systems. In further research, Agha, Marinov, and colleagues also developed techniques and tools for improved testing and model checking of actor programs [62, 86–88, 142]. Agha has also developed novel scalable formal methods for verifying parallel systems, such as computational learning for verification [147, 148], statistical model checking [126, 127], and Euclidean model checking [84, 85].

Daniel Jackson is the author of *Software Abstractions: Logic, Language and Analysis* [58], the designer of the Alloy language, and the director of the research group that developed the Alloy Analyzer and other Alloy-related tools. Courses on Alloy have been taught in more than 20 universities worldwide. In 2010, the name of the ABZ international conference was changed to include Alloy, and an Alloy track with its own committee was added; in the years since, many papers on Alloy and its applications have been published. The Alloy website [12] lists over 600 papers that discuss the use of Alloy, 100 papers describing Alloy case studies, and a third collection of papers that describe translations from more than a dozen other languages into Alloy (usually to exploit Alloy's tools).

Jackson chaired a National Academies study on software dependability (2003-2007) and was a member of the study that investigated unintended acceleration in cars (2010-2012). He was awarded a ten-year MacVicar Teaching Fellowship in 2009 (MIT's highest teaching award). He has served on the program committee of more than 40 conferences and workshops, and is an author of about 70 refereed articles on formal methods and software design.

Darko Marinov has published over 50 conference papers on software testing, including three that won ACM SIGSOFT Distinguished Paper awards (at ICSE 2010 [44], ESEC/FSE 2005 [122], and ISSTA 2002 [19]), of which one also won the ACM SIGSOFT Impact Paper Award (2012) "presented annually to the author(s) of a paper presented at a SIGSOFT sponsored or co-sponsored conference held at least 10 years prior to the award year". Six more papers were invited for journal submissions and/or nominated for best-paper awards (at ISSTA 2013, ICST 2012, ICST 2010, ISSTA 2007, ASE 2006, ASE 2001) [23, 24, 43, 48, 51, 96]. Together with his students, he has released public versions of several testing frameworks and datasets [13, 18, 21, 22, 56, 70, 83, 113, 114, 120, 128, 145, 146], the oldest being Korat [83, 101]. His group has developed several techniques for automated testing that were successfully used for testing real applications in both academia [25, 27, 74, 96, 135, 138] and industry [102, 136], most recently reporting dozens of bugs in popular Java applications such as Eclipse, NetBeans, or Apache Collections [42, 109, 141]. His research has had practical impact in industry, e.g., Korat for Java was parallelized and used at Google [102]; Korat for .NET was implemented in the SpecExplorer tool [39] developed at Microsoft Research [150] and used by testers at Microsoft to find bugs in real-world, well-tested applications [136]; and the work on detecting redundant tests [24, 155–157] influenced improvements in the popular Jtest tool developed by Parasoft [112].

1.3 Broader Impacts

Societal/technical: Our society heavily depends on software. With the advent of mobile platforms and the cloud, software is becoming increasingly distributed and event-driven. However, developing such software remains hard and error-prone despite recent advances such as web frameworks that make it easier to develop standalone servers and their clients. Therefore, improving methodologies for developing such software is an important task to ensure the continued primacy of the U.S. in developing software. Our proposed model-based programming paradigm could significantly simplify the task of developing event-driven scalable applications for the cloud. Such a paradigm could see a wide adoption in industry.

The PIs have been developing advanced models, techniques, and tools, some of which made impact in industry (Section 1.2). For example, Dr. Marinov's group has had successful direct collaborations with industry and research labs, including with Agitar [26], Google [103], Groupon [77], IBM [98], Intel [52], Microsoft [66,97,99,156], NASA [51], and Simula Labs [130]. Such collaborations are important to ensure that our research remains relevant and to enable evaluation of the research results in practice. Indeed, a good way to evaluate, as well as design and refine our techniques for programming, is to try them out on real code in practice.

Educational: We will continue integration of research and education on parallel computing at Illinois, MIT, and beyond. For example, the CS department at the University of Illinois has started to revamp the undergraduate curriculum to introduce parallelism concepts throughout the curriculum, including these courses: Intro to Programming (CS125), Data Structures (CS225), Architecture (CS232), Systems Programming (CS241), Programming Languages (CS421), Software Engineering (CS427), and Algorithms (CS473). Dr. Marinov included new modules in his software engineering class (CS427) on shared-memory and actor programming, including tool support from the NASA's Java PathFinder (JPF) model checker. We also offer specialized courses focused on parallel programming; e.g., Dr. Agha developed the core Programming Languages and Compilers course to include an introduction to concurrency and actors. He has taught advanced courses on Actor Programming Languages and Concurrency Theory, and developed a course on practical formal methods geared towards Masters' level students. The results from this proposed project will have an immediate and direct impact on the modules in all our courses.

At MIT, Dr. Jackson has developed a new software engineering course (6.170), first taught in the fall of 2011, focusing on web applications. It is rapidly becoming one of the most popular computer science classes, and its termly enrollment has grown from 30 to over 120. Until now, the course has used traditional web frameworks (most recently Ruby on Rails) as the programming platform, but the next version (this coming fall) will use an all-JavaScript framework and will explore more flexible concurrency and communication paradigms. The results of this research will have a direct application in the education of several hundred MIT computer science graduates each year. Concurrency also plays a major role in several other MIT computer science courses, including the foundation course in programming (6.005), whose term project is exactly the kind of distributed application addressed by this research.

The PIs will continue to actively involve undergraduate and graduate students in research. We have a substantial record involving undergrads in research. For example, Dr. Marinov has worked with more than 25 undergraduates [32], of whom 11 co-authored 13 papers [28,42,43,46,53,77,95,97,101–103,130,131], 12 enrolled in PhD programs so far, and several won department and national awards [32]. We also have a strong record of course projects that resulted in published paper, e.g., "Software Testing and Analysis" is an advanced topics course for graduate students at Illinois, which is project-based and focused on dynamic and static program analyses for finding software bugs; Dr. Marinov taught it six times so far, and 18 students who took the course published 15 papers based on their course projects [16,24,28,33–35,63–65,71,93,94,135,139,158].

Dr. Jackson has supervised 12 doctoral theses (with an additional 4 in progress), has been a reader on 25 doctoral theses, and has supervised 33 masters theses. For decades, MIT has engaged undergraduates in research through its "UROP" (undergraduate research opportunity) program, through which Dr. Jackson has worked with several dozen undergraduates. His department at MIT has launched a new program called "SuperUROP" that engages undergraduates in an even more intensive, year-long research experience. Dr. Jackson has supervised 4 SuperUROP projects to date, and would hope to supervise several for this project. These are paid for by industrial sponsors, without any constraints on the intellectual property produced, and without needing to use government research funds.

Diversity: The PIs will continue to integrate diversity in their research and educational activities. For example, Dr. Marinov has worked on research with seven students from underrepresented groups in computer science, including minority students, female students, and one person with disability. We plan to broaden opportunities for underrepresented groups, e.g., through our contacts at the Illinois Office of Minority Students Affairs which is active in the "Ronald E. McNair Postbaccalaureate Achievement Program" and funded by the U.S. Department of Education to encourage undergraduates members of underrepresented groups to pursue academic careers. Dr. Marinov worked as a mentor for one student through the McNair program.

Four women have completed their Ph.D.s under Dr. Agha, two of whom are now faculty members. Dr. Agha has also been involved in several programs to increase women and minority participation in Computer Science. The UIUC CS department sponsors engagement activities including *ChicTech* and *Upward Bound High School Students*, as part of its outreach programs. Dr. Agha has developed modules and demonstrations for some of these, including *ChicTech* designed for high school girls who also visit the CS department for a weekend, and *TeenTech* for inner city kids from economically distressed East Saint Louis. Dr. Agha has also hosted several minority students from the College of Engineering's *IMPRINT* program. These students did a summer internship in his lab. The program was designed to motivate engineering education and improve retention by providing practical experience of working in a research laboratory. We propose to continue such outreach activities.

Three women and one Hispanic student have completed doctorates with Dr. Jackson. He has made a particular effort to attract undergraduate women to research; this year, both of his SuperUROP awardees are women. His research group has employed three high school students so far under MIT's Research Science Institute Program. In his teaching, Dr. Jackson has endeavored to engage underrepresented students and to recognize the challenges they face. His software engineering course has included a session for all students on the challenges that women face in the process of applying for jobs, and he has counseled students in person and as a class on how to engage all team members equally in their final projects.

Dr. Jackson is a board member of MEET (Middle Eastern Education Through Technology) an organization founded by MIT students that brings Palestinian and Israeli teenagers together for an intensive 3 year program to learn software engineering and entrepreneurship. All the mentors in the summer component of the program (a full time, month long workshop) are MIT students. Half of the students taking the program are women. Three of the Palestinian alumni of MEET are now undergraduates at MIT. Dr. Jackson founded a "CSAIL Angels" group of faculty in his laboratory who last year contributed $50,000 from discretionary funds to MEET.

Dissemination: The PIs disseminate the results of their work through presentations, publications, open-source code, and educational activities, as discussed throughout the proposal. The PIs regularly give presentations about their work not only at research venues but also to industry and government (including Agitar, Google, Groupon, Fujitsu, IBM, Intel, Microsoft, NASA, Nokia, Original Software, and Parasoft) and in various settings in academia (including guest lectures in the "CS100: Freshman Orientation in CS" course and panels for industrial affiliates at Illinois).

2 Research Activities

Our research is aimed at a wide class of applications for which both distribution and concurrency are, from the point of view of an individual user, accidental. To illustrate our approach on a concrete example, we consider a chat room application. This example provides a skeleton for a number of cloud based web applications that involve distributed communication, including social media sites and popular online games. The most successfully scalable chat room applications (those involving tens of millions of users) have been written in an actor language. For example, the Facebook chat system is written in Erlang. As Facebook Engineering observed in their commentary on this choice:[1]

> "..the actor model has worked really well for us, and we wouldn't have been able to pull that off in C++ or Java. Several of us are big fans of Python and I personally like Haskell for a lot of tasks, but the bottom line is that, while those languages are great general purpose languages, none of them were designed with the actor model at heart."

Similarly, Twitter uses Scala. As Alex Payne, the pioneering developer of Twitter explained in a talk entitled "*How and Why Twitter Uses Scala*:"[2]

> "When people read about Scala, it's almost always in the context of concurrency. Concurrency can be solved by a good programmer in many languages, but it's a tough problem to solve. Scala has an Actor library that is commonly used to solve concurrency problems, and it makes that problem a lot easier to solve."

As we explain in Section 2.2, actor programming languages provide the kind of inherent parallelism scalability needed for such applications. However, while they make the problem of distribution and parallel execution simpler to address, they require a potentially *steep learning curve* for users who are not familiar with actor semantics. Moreover, actor languages do not address the problems of coordination. Our proposal is predicated on our conjecture that for a large class of applications, including web applications such as chat services, online games, etc., a friendlier high-level language can be provided so that *ordinary programmers can write high-level programs in a model closer to their usual mode of thinking*. These programs can then be translated into actors, preserving their original semantics while providing scalable, parallel execution.

The proposal would develop such a high-level programming language, mechanisms for supporting scalable and parallel implementations of programs written in it, and a toolset that enables easier development of such programs. We describe the research approach and proposed work at a high level in this section, and specific research tasks and organization in the collaboration plan (Section 5).

2.1 A Language for Event-Based Distributed Applications

To see how the user level would be expressed, consider a simple chat room application where users join chat rooms, read messages, and write messages. As far as the user is concerned, the application can be understood as consisting of a large building containing many rooms. The user can enter a room, and on doing so, will find a large whiteboard onto which messages can be written. From this user's perspective, the state of the application is just a set of rooms, a set of users, each of which can be inside one or more rooms, and for each room, a sequence of messages inscribed on the room's whiteboard. The observed behavior is simply a sequence of events in which users enter and leave rooms, and write messages on boards. These events are instantaneous, so there are no questions about conflicts between users attempting to write to the

[1] `https://www.facebook.com/notes/facebook-engineering/chat-stability-and-scalability/51412338919` (see comment on post)

[2] `http://blog.redfin.com/devblog/2010/05/how_and_why_twitter_uses_scala.html`

same board, for example. Formally then, from the user's perspective, there is a single global state (consisting of rooms, boards and users) and a collection of events that can be performed by users. There is potentially no parallelism nor distribution.

From a system implementor's perspective, however, a very different picture is needed. The state must be distributed across many client machines; the implementor's challenge is to give the users the *illusion* that there is a single global state, when in fact state updates will not be propagated to clients in perfect synchrony, so that were a user to be able to observe two clients, she might see slightly different views. Moreover, messages written to a board from different clients may be subject to varying delays. But again, these issues are neither observable nor of interest to the end users (so long as the delays are sufficiently short, of course). Likewise, while the user may view the system as having a single sequential execution of a stream of events, for the system implementor it will be essential to parallelize the application in order to make it scale, mapping the tasks (reading incoming messages, updating the boards, sending outgoing messages, maintaining presence information, etc.) to threads in a design that reflects an understanding of what kind of load will be presented, how computationally intensive each task is, how messages will be spread across boards, and so on.

Today, an application developer will have to address both of these aspects at once. At the user level, she will need to design the interactions that users will experience in terms of the event model. But this alone will not suffice; she will have to back up this view with a low-level implementation that takes account of the concurrency and distribution aspects. The difficulty is not only that the developer will have to handle both levels of the application, and manually implement the desired distribution and concurrency with low-level mechanisms such as threads, locks, message queues, etc., but that it is not generally possible to separate these two levels cleanly in the code. Consequently, changes to the user-level behavior that should be independent of the lower level concerns are much harder to make than they should be. Of course the developer can build on top of existing libraries, but the lack of a single, uniform framework makes it a daunting task to stitch all these libraries together. In a typical application, separate libraries are needed for the client-side user interface, for server-side request handling, for database access, etc. Traditional frameworks such as Rails and Django combine many of these features but at the expense of constraining the kinds of applications that can be easily built (for example, treating communications initiated by the server as a special category of "push" messages that require special features, and by limiting the use of concurrency in handling requests).

Our proposal, therefore, is to cleanly separate these two levels. At the user level, the developer will have a language in which the structure of the state and the behavior of events can be described abstractly and succinctly. At the architectural level, the developer will have a very different language, actor-based rather than event-based, that is designed to make it easy to express idioms of concurrency and distribution. Our goal is to build tool support to assist in (and eventually automatically) translate user level code into the constructs of the architecture level, which a system architect can then map to resources with declarative pragmas. Moreover, because the architecture level code is still high level, it will be easy to extend the runtime platform with new kinds of communication, concurrency and distribution idioms.

There are already frameworks that offer some flavor of this kind of functionality, but they fall short of our vision. Rails, for example, provides a feature known as "scaffolding" that allows the developer to write a high-level model of the state, from which code is then automatically generated. But the scaffolding language is not sufficiently expressive and robust to allow development to continue at that level, and inevitably, it becomes necessary to edit the generated code. Also, only the basic CRUD actions are generated; there is no way to express events with domain-specific semantics. There are frameworks that provide a global view of the state and handle server-client communication (such as Meteor), but these embody a particular architectural idiom and don't allow the flexibility that we envisage at the actor-based level to generate

implementations that scale to much larger loads.

The user-level language will provide additional features beyond the description of state and events. Two are particularly important. Rather than sprinkle security checks throughout the events, the developer will be able to write a *declarative security policy* that determines when data can be read or written by users. A rich policy language will make it possible to classify users into roles, and to make state-dependent decisions, using exactly the same data-querying language used to specify events. And for realizing graphical user interfaces, the developer will be able to write templates that tie widgets to events in a simple way, and bind queries on the global state to data display fields (in the style of functional reactive programming).

The data modeling sublanguage used to specify the global state will be essentially relational in form, and will build on the Alloy modeling language developed by Dr. Jackson, one of the PIs. This will allow powerful and succinct queries, in particular exploiting relational join and transitive closure to minimize the need for explicit iteration. Notably, the data declarations will be statically typed. Many current frameworks are only dynamically typed, which (although appealing in other respects) results in a loss of information that cannot be exploiting in code transformation. (This is why, for example, Rails, which is built on Ruby, a language without static type declarations, added a form of type declaration to scaffolding commands, since otherwise schema generation would not have been possible.)

By using a relational formalism, we expect to benefit from much recent research that uses relations as an implementation-agnostic representation. First, this will enable a variety of analyses using the Alloy Analyzer, or a similar technology that applies model finding to relational formulas. These analyses could include: generating test cases automatically, including the construction of initial global states satisfying integrity constraints; checking security policies for consistency; analyzing the relationship between security policies (e.g., whether one is strictly stronger than another); checking events to ensure that they preserve invariants; model checking sequences of events. Second, we hope to exploit work on transforming relations into concrete data structures (e.g., [54, 55]), and of course well established techniques for database schema optimizations.

2.1.1 Sample User Level Code

To illustrate our approach further, the section gives some example code written in a prototype version of our modeling language. The target application is a chat room as described in the previous section; the aim is to construct a distributed application that will run on a variety of platforms (phones, tablets, desktop computers, etc.), using one or more centralized servers to store the chat boards and user data.

We assume that anyone can create a chat room (provided that a room with the same name does not already exist) and that the existing rooms are public, so anyone can join and subsequently write messages to a room's board. With most applications of this kind, the web GUI must be responsive and interactive, automatically refreshing parts of the screen whenever something important happens (e.g., a new message is received), without reloading the whole page.

Figure 1 shows the application's user-level code. The code consist of several different *models* of the system, describing different aspects. The *data model* consists of a `User` record (which specializes a library `AuthUser` record and adds a status field), a `Msg` record (where each message has a textual body and a sender), and a `ChatRoom` record (each room has a name, a set of participating users, and a sequence of messages that have been sent). These fields are defined using the `refs` and `owns` keywords: the former denotes aggregation (simple referencing, without any constraints), and the latter denotes composition (implying that (1) when a record is deleted, all owned records should be deleted, and (2) no two distinct records can point to the same record via the same owned field).

The *network model* in this example consists of two machines, namely `Server` and `Client`. The `Client`

(a) data model

```
record User < AuthUser do
  # inherited fields
  #   name: String,
  #   email: String,
  #   pswd_hash: String,
  refs status: String
end
```

```
record Msg do
  refs text: Text,
       sender: User
end
```

```
record ChatRoom do
  refs name: String,
       members: (set User)
  owns messages: (seq Msg)
end
```

(b) network model

```
machine Client < AuthClient
  refs user: User
end

machine Server < AuthServer
  owns rooms: (set ChatRoom)
end
```

(c) event model

```
event CreateRoom do
  from client: Client
  to   serv: Server

  params roomName: String

  requires {
    client.user     &&
    roomName        &&
    roomName != "" &&
    !serv.rooms.
      find_by_name(roomName)
  }

  ensures {
    room = ChatRoom.create
    room.name = roomName
    room.members = [client.user]
    serv.rooms << room
  }
end
```

```
event JoinRoom do
  from client: Client
  to   serv: Server

  params room: ChatRoom

  requires {
    u = client.user
    client.user &&
    !room.members.include?(u)
  }

  ensures {
    room.members << client.user
  }
end
```

```
event SendMsg do
  from client: Client
  to   serv: Server

  params room: ChatRoom,
         msgText: String

  requires {
    client.user &&
    room.members.include?
      (client.user)
  }

  ensures {
    msg = Msg.create
    msg.text = msgText
    msg.sender = client.user
    room.messages << msg
  }
end
```

(d) security model

```
policy HideUserPrivateData do
  principal client: Client

  # restrict access to passwords
  # except for owning user
  restrict User.pswd_hash.unless do |user|
    client.user == user
  end

  # restrict access to status messages to
  # users who share at least one chat room
  # with the owner of that status message
  restrict User.status.when do |user|
    client.user != user &&
    ChatRoom.none? { |room|
      room.members.include?(client.user) &&
      room.members.include?(user)
    }
  end
end
```

```
policy FilterChatRoomMembers do
  principal client: Client

  # filter out anonymous users (those who have not
  # sent anything) from the 'members' field
  restrict ChatRoom.members.reject do |room, member|
    !room.messages.sender.include?(member) &&
    client.user != member
  end
end
```

Figure 1: A model of a simple chat room application, written in our prototype modeling notation.

machine has a corresponding `User`, whereas the `Server` machine maintains a set of active `ChatRoom`s. They respectively inherit from the library `AuthClient` and `AuthServer` machines, to bring in some fairly standard (but library-defined, as opposed to built-in) user management behavior, such as new user registration, sign-in and sign-out events, etc.

To implement the basic functionality of the chat room, we define an *event model* with three event types: `CreateRoom`, `JoinRoom`, and `SendMsg`, as shown in Figure 1(c).

Each event has an appropriate precondition (given in the `requires` clause) that checks that the requirements for the event are all satisfied before the event may be executed. For instance, events `CreateRoom`, `JoinRoom`, and `SendMsg` all require that the user has signed in (`client.user` is non-empty), `SendMsg` requires that the user has joined the room, etc.

A specification of the effects of an event (given in the `ensures` clause) is concerned only with updating relevant data records and machines to reflect the occurrence of that event. For example, the effects of the `JoinRoom` event amount to simply adding the user requesting to join the room to the set of room members; the runtime system will make sure that this update is automatically pushed to all clients currently viewing that room. Actions such as updating the user interface are specified elsewhere, independently of the event model; this is a key to achieving separation of concerns.

By default, all fields in our models are public and visible to all machines in the system. To restrict access, we defined a security policy as a collection of declarative rules.

The `HideUserPrivateData` policy in Figure 1(d) says that the value of a user's password should not be revealed to any other user and, similarly, that the status message of a user should not be revealed to any other user, unless the two users are currently both members of the same chat room. Note that this latter rule is dynamic, i.e., it depends on the current state of the system (two users being together in a same chat room) and thus its evaluation for two given users may change over time.

In addition to restricting access to a field entirely, when a field is of a collection type, a policy can also specify a filtering condition to be used to remove certain elements from that collection before the collection is sent to another machine. The `FilterChatRoomMembers` policy hides those members of a chat room who have not sent any messages (this simulates, to some extent, "invisible users", a feature supported by some chat clients).

Policies will be automatically enforced; in our prototype, policies are checked at every field access; if any policy is violated the access is forbidden simply by replacing the field value with an empty value.

2.2 Scalable Execution

In order for high-level parallel languages to be practical, they must not only simplify programming by separating design concerns from implementation details but provide efficient implementation. This requires translating the high-level code into an intermediate form which facilitates scalable and parallel execution. The actor model provides flexible low-level primitives for such scalable execution [3, 10].

Actor semantics provides encapsulation (isolation of local state), fair scheduling, location transparency (location independent naming), locality of reference (facilitating security), and transparent migration. These properties enable compositional design and simplify reasoning [7], and improve performance as applications and architectures scale [80]. For example, because actors communicate using asynchronous messages, an actor does not hold any system resources while sending and receiving a message. This is in contrast to the shared-memory model where threads occupy system resources such as a system stack and possibly other locks while waiting to obtain a lock. Thus actors provide failure isolation while potentially improving performance.

Asynchronous messaging is a key source of nondeterminism in Actor programs: the order in which messages are processed affects the behavior of an actor. In many applications, application programmers want to prune some of the nondeterminism by restricting the possible orders in which messages are processed. Two commonly used abstractions that constrain the message order are *request-reply messaging* and *local synchronization constraints*. Local synchronization constraints capture ordering requirements (cf. session types [20] projected onto an individual actor). Such constraints will be expressed in the requirements on events, as well as in the declarative policy language described in Section 2.1.

At the semantic level, an actor language extends a sequential object (or, alternately, a functional) programming language with three primitive concurrency operators:

send transmits a message to a specified actor. The message will be buffered in a *mail queue* at the destination until the actor is ready to process it. Each actor has a unique *mail address* which is used to specify a target for communication. Mail addresses may also be communicated in a message, allowing for a dynamic communication topology.

new is used to dynamically create an actor (with a fresh address) using the specified actor behavior (class name) and parameters.

become (or **ready**) marks the end a method, the actor can now process the next message in its queue.

The ability to create new actors facilitates the representation of dynamic applications such as our example chat system where new users may be created. The ability of actors to communicate mail addresses of actors enables actions such as join or leave a chat room. Moreover, natural extensions such as the ActorSpace model [5] allow (potentially overlapping) groups of actors to be defined, facilitating multicast in a dynamically changing environment.

Asynchronous communication between actors reduces synchronization overhead and network traffic, increasing system throughput. Because actors are self-contained and location independent, dynamically tuning performance by relocating server applications becomes easier. For example, the load of computers on a cloud may be dynamically balanced by remote creation, replication, or migration of applications. Alternatively, specific policies may be separately specified and used to facilitate trade-offs such as availability versus consistency.

2.2.1 Translating Event-Based Code to Actors

In order to execute high-level event based code on a distributed architecture, it needs to be translated into a concurrent language. Because we want scalability, our target will be an actor language as an intermediate step. In the chat room example, all client users might be represented initially as distinct actors. These actors send message such as `JoinRoom` or `SendMsg` requests to an actor representing a `Server`. The `Server` supports handlers for all these messages, which are executed in a non-preemptive manner. The server processes order to create/join a chat room and to broadcast a message to everybody in a room. Since actors can be created and their mail addresses communicated, new users can be added to the system and their interconnection topology can evolve over time. Moreover, an actor may only send messages to other actors whose address has been communicated to it. This facilitates security (access control) as well as automatic garbage collection [151].

Obviously, a single server in this case would become a bottleneck. Fortunately, in the chat room example as in many cases in the real-world, there is a logical or natural decentralization and limited number of links or connections between different actors. We propose to use both static and dynamic analysis tools to infer patterns of interaction and thus detect logical clustering of data and actions which can be used to create actors. This will allow us to fission large actors into equivalent system of multiple actors. The actors need not be entirely asynchronous in the sense that there may still have consistency requirements between them local states. We will address this issue in Section 2.2.2.

While actors make it is possible to express scalable parallel code, a naive translation can introduce bottlenecks. For example, because of a user actor, there may be a dependence between the states of different chat rooms. However, naively implementing all the chat rooms on a single server would cause a bottleneck. Avoiding such a bottleneck requires inferring the extent of the interactions and creating actors to capture the logical decentralization. Moreover, efficient implementation also requires appropriate placement and

distribution strategies. For this reason, implementations can require expert parallel programmers. To make such performance benefits accessible to a larger group of programmers, an inference tool could observe the interactions between actors, learn characteristic patterns, and suggest respective placement pragmas to the programmer [37].

While actors into which the source program is translated operate asynchronously, certain orderings of messages are often necessary. Actor languages allow local synchronization constraints to specify correct orderings. These synchronization constraints are enforced by delaying messages received before the target actor is in a state where it can process a message. For example, a constraint on `Server` specifies that a chat room must exist before another user actor can join the chat room. Thus a message creating the chat room must be processed before requests to join it can be processed.

2.2.2 Distributed Coordination of Actors

While a relational notation is friendly for high-level programmers, it requires translation into constraints such as atomicity between actions. Moreover, such constraints can be context dependent. In some contexts, strict consistency requirements are essential, in others, eventual consistency suffices. Other systems may tolerate speculative actions and rollback. We propose to use a constraint language which relates events specified in the high-level language. These constraints will be enforced on the actor execution guided by pragmas. We will build on previous work using an efficient reflective middleware (e.g. [14,137]), augmented with a scoping mechanism that prevents the abuse of re-configuration privileges by malicious or faulty actors [36].

2.2.3 Translating Pragmas

Satisfying requirements for timeliness of response, security, performance, energy use or other QoS parameters, often requires careful mapping of computations to underlying resources [110]. Such mapping may be assisted by use of pragmas. Using our approach, orthogonal design requirements may be expressed as pragmas which are implemented by meta-level actors in the middleware. Such a meta-level customization would be a generalization of standard middleware architectures which only provide flexible naming and communication (cf. [15, 152]). Pragmas will be used to guide customization of components by changing the communication between them, by scheduling them differently, or by recording or restoring their local state [115, 152].

2.3 Development Toolset

We plan to develop a set of tools for automated reasoning and analysis of the programs written in our proposed language. Although the proposed language simplifies the development of mobile cloud apps, and by construction eliminates a whole class of concurrency bugs, it does not eliminate all possible bugs. The user implementation of events can still fail to satisfy the functional requirements of the application. Applying the standard software quality assurance techniques to the new programs is, therefore, still of high importance. In fact, we propose to design the language with this in mind, and this section discusses how our programming paradigm is amenable to techniques like automated testing, model checking, and software verification. We will build on our joint prior work in this area, the PIs Agha and Marinov have collaborated on testing, in particular for actor programs [62, 86–88, 142] and on automated test generation [122], while the PIs Jackson and Marinov have collaborated on model and program analysis [75, 76, 100].

2.3.1 Testing

Testing is the most widely used method for checking program correctness. Testing an event-driven system is both challenging and time consuming, because one needs to generate *realizable* traces (sequences of events).

The challenging part in discovering realizable traces is that the preconditions need to hold for each event in the sequence, and the time-consuming part is that the traces can be long, and therefore, there can be too many of them to explore manually. Having both preconditions and postconditions of each event formally specified in our event model allows us to use symbolic-execution techniques [81], and build on recent successes in this domain [154], to discover possible traces automatically.

A symbolic execution engine would start with an empty path condition; at each step, the engine would consider all events from the model and discover the ones that are realizable from the current state (this can be done by using an automated SMT solver [17,38] to check if there exists a model in which both the current path condition and the event's precondition are satisfied). When an event is found to be realizable, a new state is created and the event's postcondition is appended to the path condition for the new state. Since at each step of this process multiple events may be found to be realizable, the algorithm proceeds by exploring the entire graph, effectively yielding a *state diagram* by the end. Each node in the diagram describes a symbolic state, and each edge describes a transition that can happen when the condition on the edge is satisfied and the event is executed. For example, moving from the initial state to the next state requires that a user initiates a `SignIn` event and provides a correct name and password. This transition results in the execution of the postcondition of the `SignIn` event.

In addition to automated testing of traces, a state diagram can be used to automatically create a test environment – the state necessary before the execution of a test – for all unit tests. For example, if a developer wants to test a `SendMsg` event for a chat room, there should be a registered user in a room. To create such a state, a sequence of other events have to be executed before `SendMsg`, e.g., `SignIn` and `CreateRoom` event handlers must be executed. Executing these events requires solving the precondition of each event on the path to produce the required pre-state.

Functional unit testing of events also becomes easier. A black-box unit test for the `SendMsg` event would have to check that the message sent indeed gets added to the list of messages of the given room, that it gets added to the very end of the list, that no other messages get dropped from that list, etc. In our proposed paradigm, this can be done directly, without having to set up any mock objects, e.g., to abstract the network and the actual peer points, as no network is required.

In a traditional event-driven system, an implementation of a single functional unit is often fragmented over several classes. Consider the `SignIn` event: the user sends his or her credentials to the server, the server checks the authenticity, sends back the result, and based on that result, both the server and the client update their local state. In the traditional model, the client-side code can initiate the event (by sending a message to the server), and schedule a continuation that will run when a response is received. The continuation, which is typically a separate procedure, then implements the logic for updating the local state based on the server response. Such fragmented code is very hard to test as a unit, so it is often turned into an integration test, and integration tests are typically more laborious to write and require more elaborate setup. In contrast, because of the shared view of the global data in our approach, there is no need for such fragmentation; the event handler can be a single procedure that updates only the global data model, meaning that it can easily be tested as a unit.

2.3.2 Model Checking

Loosely coupled events without explicit synchronization and communication allow model checking to scale. The source of non-determinism in our models is the order in which events are executed. Because of the non-determinism in scheduling, a model may exhibit different behavior for the same input (i.e., the same values of event parameters) with a different order of event execution. The goal of software model checking is conceptually to explore all orders to ensure the correct execution. Note that the exploration need consider only

the semantics of the model and not the semantics of the underlying runtime system. Based on our prior experience with model checking actor programs [86, 142], X10 programs [51], and database applications [49], we believe that an efficient model checking approach can be developed for our new paradigm.

For example, a model checker can be used to check end-to-end properties for all scenarios that the system can possibly exhibit. One such property could be "it is impossible that at one point two different chat rooms have two different users with the same name". The tool can automatically either confirm that the property in question always holds, or find a scenario (i.e., a sequence of events leading to a state) in which the property is violated.

The technique of discovering realizable sequences of events can also be used to synthesize higher-level operations. For example, a novice chat user may wonder what are the steps that need to be taken in order to post a message in a chat room. Given such an end-goal, a tool can discover that one possible scenario to achieve that goal is to first `SignIn`, then `JoinRoom`, and finally `SendMsg`. An alternative solution would be to `CreateRoom` instead of `JoinRoom` at the second step. These scenarios can be displayed to the designer and serve as a guide to better understanding possible functionalities of the system (which can be especially useful for bigger systems with many possible events).

3 Results from Prior NSF Support

Gul Agha has been supported in part by three current and recent NSF grants. One grant is CNS-1035562 "CPS: Medium: Collaborative Research: Cyber-Physical Co-Design of Wireless Monitoring and Control for Civil Infrastructure," collaborative with Purdue and Washington University ($500,000, 10/2010–08/2014). The grant resulted in the development of the Wireless Cyber-Physical-Simulator. Two simulations based realistic case studies have been carried out, each combining a structural model with wireless traces collected from real-world environments including a deployment of 113 sensor nodes at the Jindo Island bridge by the PI and collaborators. Its **intellectual merit** includes developing methods for efficient low power listening, low latency and high precision data acquisition. Ongoing work includes work on smart sensor platform for wireless structural monitoring and control [8, 91]. Its **broader impacts** are that the simulations can help improve the design of bridges, and the methods enable continuous autonomic monitoring of bridges for damaged elements.

Second grant is OCI-1030454 "EAGER: Bio-Inspired Smart Sensor Networks for Adaptive Emergency Response" ($300,000, 06/2010–05/2013). Its **intellectual merit** includes developing new techniques for identifying individuals' location and status inside buildings and implementing these techniques in a software system on the Android platform for mobile devices (the system is called iRescue, for "Illinois Rescue", available publicly online [57], and estimates the status and the location of victims). Its **broader impacts** are that iRescue facilitates emergency response to help victims trapped inside the building in a catastrophic accident.

Third grant is CMMI-0928886 "Bio-Informed Framework Enabling Multimetric Infrastructure Monitoring," ($435,528, 08/2009–07/2013). It developed a service-oriented architecture that will result in modular, easy to maintain applications that can be adapted for monitoring different types of structures and will manage the complexity demanded by multimetric sensing. The **intellectual merit** of the project is providing a service-oriented architecture for parallel and distributed computing to facilitate scalable sensor network applications by domain specialists. The project also developer new algorithms that take advantage of the availability of a variety of measurands at multiple temporal and spatial scales. The **broader impacts** of the project are fusing together multi-scale sensed data with new decentralized modal analysis and damage detection strategies, which achieves a more accurate assessment of structural behavior and health [68, 69, 78, 92, 104, 105, 117–119].

Daniel Jackson has received 7 NSF grants. The most relevant (and the only one granted in the last 5 years) is the award 0707612 "CRI: CRD–Development of Alloy Tools, Technology and Materials" ($800,000, 08/2007–07/2012, extended to 01/2015). Alloy is a modeling language for data-intensive systems, with an automatic analysis tool, the Alloy Analyzer, that uses a SAT solver to find counterexamples to claimed properties and to generate sample executions. The Alloy website [12] includes a list of over 600 papers that discuss Alloy, 100 papers describing applications of Alloy, and papers on translating a dozen other languages into Alloy. Alloy has been taught at universities throughout the world, at the undergraduate and masters level. The Alloy book [58] was published in 2006 with a revised edition in 2012.

This is the only research grant that funded the core work on Alloy. Outcomes of the project have been primarily improvements to the Alloy language and tools, but its **intellectual merit** also included some major new advances: the development of Kodkod, an entirely new model finder (the computational engine of the Alloy Analyzer), that exploits symmetry breaking and sharing to dramatically improve the tools performance and scalability; a new algorithm using using unsatifiable cores to find inconsistencies and obtain a measure of property coverage; an experimental reduction of Alloy to logic programming; a new semantics and analysis scheme for integers that accounts for the difficulties of operations that might overflow. Improvements to the Alloy Analyzer included a reengineering of the Alloy Analyzers API and a new visualization engine. A series of tools for applying Alloy to code were developed, including: a converter from JML to Alloy; Forge, a tool for checking Java code with Alloy; WebAlloy, a tool for synthesizing web applications from Alloy models; Squander, a tool for executing declarative specifications in the context of a running Java program; and Rubicon, a tool for checking security properties of Ruby on Rails applications using Alloy. New kinds of tools based on Alloy were developed, including: Moolloy, a multi-objective optimizer; contributions to ConfigAssure, a network configuration tool developed in collaboration with Telcordia for DARPA; and a tool for analyzing Apache configuration files that uses the Alloy Analyzer to find vulnerabilities and generate explanations. Finally, some extensive case studies were performed, including: a flash file system challenge problem; an analysis of the specifications of the NSAs Tokeneer system; an analysis of a new electoral protocol; and dependability analysis for a proton therapy machine at Massachusetts General Hospital.

In terms of **broader impact**, the project included engagement of high school students in summer internships; training of women graduate students; presentations in a local middle school; collaborations with colleagues, and application to their work, in other disciplines (notable Aeronautics and Astronautics). In 2010, the name of the ABZ international conference was changed to include Alloy, and an Alloy track with its own committee was added.

Darko Marinov has had several NSF grants, but note that three are expiring in 2014. The most relevant award on testing is CCF-0746856 "CAREER: Systematic Software Testing Using Test Abstractions" ($400,000 + $6,000 REU supplement, 06/2008–05/2013, extended to 05/2014). Its **intellectual merit** is a novel testing approach in which the users do not manually write individual tests but instead specify *test abstractions* from which tools can generate a large number of tests. Its **broader impacts** include the use of test abstractions in various open-source projects to find over 200 bugs [13, 120, 145, 146], training of students (2 PhD theses [59, 89], 4 MS theses [67, 82, 129, 140], and many undergraduates as described in Section 1.3), and increased diversity (including seven minority students, female students, and one student with disability). So far the award partially supported 40 papers [28–31, 42–48, 50, 51, 60–65, 73, 77, 86–88, 90, 100, 107–109, 130–134, 141, 142, 159–162] (including an award-winning ICSE 2010 paper [44], 2 papers invited for journal submissions [43, 48], and one more paper nominated for best-paper award [51]) and public release of 11 testing tools and datasets [13, 18, 21, 22, 56, 70, 113, 114, 120, 145, 146].

References

[1] SNAP'N'SHOT home page. http://www.snapnshot.info/.

[2] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, MA, USA, 1986.

[3] G. Agha. Concurrent object-oriented programming. *Commun. ACM*, 33(9):125–141, 1990.

[4] G. Agha. Modeling concurrent systems: Actors, nets, and the problem of abstraction and composition. In *Application and Theory of Petri Nets*, pages 1–10, 1996.

[5] G. Agha and C. J. Callsen. Actorspaces: An open distributed programming paradigm. In M. C. Chen and R. Halstead, editors, *PPOPP*, pages 23–32. ACM, 1993.

[6] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. Towards a theory of actor computation. In *CONCUR*, pages 565–579, 1992.

[7] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation. *J. Funct. Program*, 7(1):1–72, 1997.

[8] G. Agha and B. F. Spencer. The illinois structural health monitoring project. http://shm.cs.illinois.edu.

[9] G. Agha and P. Thati. An algebraic theory of actors and its application to a simple object-based language. In O. Owe, S. Krogdahl, and T. Lyche, editors, *Essays in Memory of Ole-Johan Dahl*, volume 2635 of *Lecture Notes in Computer Science*, pages 26–57. Springer, 2004.

[10] G. A. Agha. *ACTORS - a model of concurrent computation in distributed systems*. MIT Press series in artificial intelligence. MIT Press, 1990.

[11] G. A. Agha and W. Kim. Actors: A unifying model for parallel and distributed computing. *Journal of Systems Architecture*, 45(15):1263–1277, 1999.

[12] Alloy web site. http://alloy.mit.edu.

[13] ASTGen home page. http://mir.cs.illinois.edu/astgen/.

[14] M. Astley and G. Agha. Customizaton and compositon of distributed objects: Middleware abstractions for policy management. In *SIGSOFT FSE*, pages 1–9. ACM, 1998.

[15] M. Astley, D. C. Sturman, and G. Agha. Customizable middleware for modular distributed software. *Commun. ACM*, 44(5):99–107, 2001.

[16] S. Badame and D. Dig. Refactoring meets spreadsheet formulas. In *Proc. of the 28th IEEE International Conference on Software Maintenance (ICSM 2012)*, pages 399–409, Trento, Italy, Sept. 2012.

[17] C. Barrett and S. Berezin. CVC Lite: A new implementation of the cooperating validity checker. In *Proceedings of the International Conference on Computer Aided Verification*, pages 515–518, 2004.

[18] Basset home page. http://mir.cs.illinois.edu/basset/.

[19] C. Boyapati, S. Khurshid, and D. Marinov. Korat: Automated testing based on Java predicates. In *Proc. of the International Symposium on Software Testing and Analysis (ISSTA 2002)*, pages 123–133, Rome, Italy, July 2002.

[20] M. Charalambides, P. Dinges, and G. Agha. Parameterized concurrent multi-party session types. In N. Kokash and A. Ravara, editors, *FOCLASA*, volume 91 of *EPTCS*, pages 16–30, 2012.

[21] CoDeSe home page. http://mir.cs.illinois.edu/codese/.

[22] Coverage home page. http://mir.cs.illinois.edu/coverage/.

[23] M. d'Amorim, S. Lauterburg, and D. Marinov. Delta execution for efficient state-space exploration of object-oriented programs. In *Proc. of the International Symposium on Software Testing and Analysis (ISSTA 2007)*, London, UK, July 2007.

[24] M. d'Amorim, C. Pacheco, T. Xie, D. Marinov, and M. D. Ernst. An empirical comparison of automated generation and classification techniques for object-oriented unit testing. In *Proc. of the 21st IEEE/ACM Conference on Automated Software Engineering (ASE 2006)*, pages 59–68, Tokyo, Japan, Sept. 2006.

[25] M. d'Amorim, A. Sobeih, and D. Marinov. Optimized execution of deterministic blocks in Java PathFinder. In *Proc. of the 8th International Conference on Formal Engineering Methods (ICFEM 2006)*, volume 4260 of *LNCS*, pages 549–567, Macau, China, Nov. 2006.

[26] B. Daniel and M. Boshernitsan. Predicting effectiveness of automatic testing tools. In *Proc. of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008)*, pages 363–366, L'Aquila, Italy, Sept. 2008.

[27] B. Daniel, D. Dig, K. Garcia, and D. Marinov. Automated testing of refactoring engines. In *Proc. of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2007)*, Dubrovnik, Croatia, Sept. 2007.

[28] B. Daniel, D. Dig, T. Gvero, V. Jagannath, J. Jiaa, D. Mitchell, J. Nogiec, S. H. Tan, and D. Marinov. ReAssert: A tool for repairing broken unit tests. In *Proc. of the International Conference on Software Engineering, Demonstrations Track (ICSE Demo 2011)*, pages 1010–1012, Honolulu, HI, May 2011.

[29] B. Daniel, T. Gvero, and D. Marinov. On test repair using symbolic execution. In *Proc. of the International Symposium on Software Testing and Analysis (ISSTA 2010)*, pages 207–218, Trento, Italy, July 2010.

[30] B. Daniel, V. Jagannath, D. Dig, and D. Marinov. ReAssert: Suggesting repairs for broken unit tests. In *Proc. of the 24th IEEE/ACM Conference on Automated Software Engineering (ASE 2009)*, pages 433–444, Auckland, New Zealand, Nov. 2009.

[31] B. Daniel, Q. Luo, M. Mirzaaghaei, D. Dig, D. Marinov, and M. Pezzè. Automated GUI refactoring and test script repair (position paper). In *The First International Workshop on End-to-End Test Script Engineering (ETSE 2011)*, pages 38–41, Toronto, Canada, July 2011.

[32] Darko Marinov's Students. `http://mir.cs.illinois.edu/marinov/students.html`.

[33] D. Dig, C. Comertoglu, D. Marinov, and R. Johnson. Automatic detection of refactorings for libraries and frameworks. In *the International Workshop on Object-Oriented Reengineering (WOOR 2005)*, Glasgow, UK, July 2005.

[34] D. Dig, S. Negara, V. Mohindra, and R. Johnson. ReBA: A tool for generating binary adapters for evolving java libraries. In *Proc. of the 30th International Conference on Software Engineering, Demo Track (ICSE Demo 2008)*, pages 963–964, May 2008.

[35] D. Dig, S. Negara, V. Mohindra, and R. Johnson. ReBA: Refactoring-aware binary adaptation of evolving libraries. In *Proc. of the 30th International Conference on Software Engineering (ICSE 2008)*, pages 441–450, May 2008.

[36] P. Dinges and G. Agha. Scoped synchronization constraints for large scale actor systems. In M. Sirjani, editor, *COORDINATION*, volume 7274 of *Lecture Notes in Computer Science*, pages 89–103. Springer, 2012.

[37] P. Dinges, M. Charalambides, and G. Agha. Automated inference of atomic sets for safe concurrent execution. In *PASTE*, pages 1–8, 2013.

[38] B. Dutertre and L. de Moura. A fast linear-arithmetic solver for DPLL(T). In *Proceedings of the International Conference on Computer Aided Verification*, pages 81–94, 2006.

[39] Foundations of Software Engineering, Microsoft Research. SpecExplorer home page. `http://research.microsoft.com/specexplorer`.

[40] S. Frølund. *Coordinating distributed objects - an actor-based approach to synchronization*. MIT Press, 1996.

[41] S. Frølund and G. Agha. A language framework for multi-object coordination. In *ECOOP*, pages 346–360, 1993.

[42] M. Gligoric, F. Behrang, Y. Li, J. Overbey, M. Hafiz, and D. Marinov. Systematic testing of refactoring engines on real software projects. In *Proc. of the 27th European Conference on Object-Oriented Programming (ECOOP 2013)*, pages 629–653, Montpellier, France, July 2013.

[43] M. Gligoric, A. Groce, C. Zhang, R. Sharma, A. Alipour, and D. Marinov. Comparing non-adequate test suites using coverage criteria. In *Proc. of the ACM International Symposium on Software Testing and Analysis (ISSTA 2013)*, pages 302–313, Lugano, Switzerland, July 2013.

[44] M. Gligoric, T. Gvero, V. Jagannath, S. Khurshid, V. Kuncak, and D. Marinov. Test generation through programming in UDITA. ICSE '10, New York, NY, USA, 2010. ACM.

[45] M. Gligoric, T. Gvero, S. Khurshid, V. Kuncak, and D. Marinov. On delayed choice execution for falsification. Technical Report LARA-REPORT-2008-008, EPFL, IC, Lausanne, Switzerland, Oct. 2008.

[46] M. Gligoric, T. Gvero, S. Lauterburg, D. Marinov, and S. Khurshid. Optimizing generation of object graphs in Java PathFinder. In *Proc. of the 2nd International Conference on Software Testing, Verification, and Validation (ICST 2009)*, Denver, CO, Apr. 2009.

[47] M. Gligoric, V. Jagannath, Q. Luo, and D. Marinov. Efficient mutation testing of multithreaded code. *Software Testing, Verification and Reliability*, 23(5):375–403, Aug. 2013.

[48] M. Gligoric, V. Jagannath, and D. Marinov. MuTMuT: Efficient exploration for mutation testing of multithreaded code. In *Proc. of the 3rd International Conference on Software Testing, Verification, and Validation (ICST 2010)*, pages 55–64, Paris, France, Apr. 2010.

[49] M. Gligoric and R. Majumdar. Model checking database applications. In *TACAS*, 2013.

[50] M. Gligoric, D. Marinov, and S. Kamin. CoDeSe: Fast deserialization via code generation. In *Proc. of the International Symposium on Software Testing and Analysis (ISSTA 2011)*, pages 298–308, Toronto, Canada, July 2011.

[51] M. Gligoric, P. C. Mehlitz, and D. Marinov. X10X: Model checking a new programming language with an "old" model checker. In *Proc. of the Fifth IEEE International Conference on Software Testing, Verification and Validation (ICST 2012)*, pages 11–20, Montreal, Canada, Apr. 2012.

[52] M. Gligoric, L. Zhang, C. Pereira, and G. Pokam. Selective mutation testing for concurrent code. In *Proc. of the ACM International Symposium on Software Testing and Analysis (ISSTA 2013)*, pages 224–234, Lugano, Switzerland, July 2013.

[53] T. Gvero, M. Gligoric, S. Lauterburg, M. d'Amorim, D. Marinov, and S. Khurshid. State extensions for Java PathFinder. In *Proc. of the International Conference on Software Engineering, Formal Demo Papers (ICSE Demo 2008)*, pages 863–866, Leipzig, Germany, May 2008.

[54] P. Hawkins, A. Aiken, K. Fisher, M. C. Rinard, and M. Sagiv. Data representation synthesis. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 38–49, San Jose, CA, USA, June 2011.

[55] P. Hawkins, A. Aiken, K. Fisher, M. C. Rinard, and M. Sagiv. Concurrent data representation synthesis. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 417–428, Beijing, China, June 2012.

[56] IMUnit Home Page. `http://mir.cs.illinois.edu/imunit/`.

[57] iRescue web site. `http://cee.illinois.edu/irescue`.

[58] D. Jackson. *Software Abstractions: Logic, Language and Analysis*. The MIT Press, Cambridge, MA, 2006.

[59] V. Jagannath. *Improved Regression Testing of Multithreaded Programs*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, May 2012.

[60] V. Jagannath, M. Gligoric, D. Jin, Q. Luo, G. Rosu, and D. Marinov. Improved multithreaded unit testing. In *Proc. of the 8th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2011)*, pages 223–233, Szeged, Hungary, Sept. 2011.

[61] V. Jagannath, M. Gligoric, D. Jin, G. Rosu, and D. Marinov. IMUnit: Improved multithreaded unit testing. In *the Third International Workshop on Multicore Software Engineering (IWMSE 2010)*, pages 48–49, Cape Town, South Africa, May 2010.

[62] V. Jagannath, M. Gligoric, S. Lauterburg, D. Marinov, and G. Agha. Mutation operators for actor systems. In *the 5th International Workshop on Mutation Analysis (Mutation 2010)*, pages 157–162, Paris, France, Apr. 2010.

[63] V. Jagannath, M. Kirn, Y. Lin, and D. Marinov. Evaluating machine-independent metrics for state-space exploration. In *Proc. of the Fifth IEEE International Conference on Software Testing, Verification and Validation (ICST 2012)*, pages 320–329, Montreal, Canada, Apr. 2012.

[64] V. Jagannath, Y. Y. Lee, B. Daniel, and D. Marinov. Reducing the costs of bounded-exhaustive testing. In *Proc. of the Fundamental Approaches to Software Engineering (FASE 2009)*, pages 171–185, York, UK, Mar. 2009.

[65] V. Jagannath, Q. Luo, and D. Marinov. Change-aware preemption prioritization. In *Proc. of the International Symposium on Software Testing and Analysis (ISSTA 2011)*, pages 133–143, Toronto, Canada, July 2011.

[66] V. Jagannath, Z. Yin, and M. Budiu. Monitoring and debugging DryadLINQ applications with Daphne. In *International Workshop on High-Level Parallel Programming Models and Supportive Environments*, pages 1266–1273, 2011.

[67] V. S. B. Jagannath. Reducing the costs of bounded-exhaustive testing. Master's thesis, University of Illinois at Urbana-Champaign, Urbana, IL, Aug. 2010.

[68] S. Jang, H. Jo, S. Cho, K. Mechitov, J. Rice, S.-H. Sim, H.-J. Jung, C.-B. Yun, B. F. Spencer, and G. Agha. Structural health monitoring of a cable-stayed bridge using smart sensor technology: deployment and evaluation. *Smart Structures and Systems*, 6(5):439–460, 2010.

[69] H. Jo, S.-H. Sim, K. Mechitov, R. Kim, J. Li, P. Moinzadeh, B. F. Spencer, J. W. Park, S. Cho, H.-J. Jung, C.-B. Yun, J. Rice, and T. Nagayama. Hybrid wireless smart sensor network for full-scale structural health monitoring of a cable-stayed bridge. *Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems*, 7981(1), 2011.

[70] Java PathFinder contributions from Darko Marinov's group. `http://mir.cs.illinois.edu/jpf/`.

[71] S. Kamin, B. Aktemur, and M. Katelman. Staging static analyses for program generation. In *Proc. of the 5th International Conference on Generative Programming and Component Engineering (GPCE 2006)*, pages 1–10, Oct. 2006.

[72] R. K. Karmani, A. Shali, and G. Agha. Actor frameworks for the JVM platform: A comparative analysis. In *PPPJ*, pages 11–20, 2009.

[73] S. A. Khalek, G. Yang, L. Zhang, D. Marinov, and S. Khurshid. TestEra: A tool for testing Java programs using Alloy specifications. In *Proc. of the 26th IEEE/ACM International Conference On Automated Software Engineering, Tool Demonstrations Track (ASE Demo 2011)*, pages 608–611, Lawrence, KS, Nov. 2011.

[74] S. Khurshid and D. Marinov. TestEra: Specification-based testing of Java programs using SAT. *Automated Software Engineering Journal*, 11(4):403–434, Oct. 2004.

[75] S. Khurshid, D. Marinov, and D. Jackson. An analyzable annotation language. In *Proc. of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 231–245, Seattle, WA, Nov. 2002.

[76] S. Khurshid, D. Marinov, I. Shlyakhter, and D. Jackson. A case for efficient solution enumeration. In *Proc. of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, Santa Margherita Ligure, Italy, May 2003.

[77] C. H. P. Kim, D. Marinov, S. Khurshid, D. Batory, S. Souto, P. Barros, and M. d'Amorim. SPLat: Lightweight dynamic analysis for reducing combinatorics in testing configurable systems. In *Proc. of the 9th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2013)*, pages 257–267, St. Petersburg, Russia, Aug. 2013.

[78] R. Kim, S.-H. Sim, K. Mechitov, J. Song, and B. F. Spencer. Reliability-based diagnosis of wireless sensor network communication quality using readily available data. In *Joint Conference of the Engineering Mechanics Institute and the 11th ASCE Joint Specialty Conference on Probabilistic Mechanics and Structural Reliability*, June 2012.

[79] W. Kim and G. Agha. Compilation of a highly parallel actor-based language. In *LCPC*, pages 1–15, 1992.

[80] W. Kim and G. Agha. Efficient support of location transparency in concurrent object-oriented programming languages. In *SC*, page 39, 1995.

[81] J. C. King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.

[82] M. A. Kirn. Evaluating machine-independent metrics for state-space exploration. Master's thesis, University of Illinois at Urbana-Champaign, Urbana, IL, Dec. 2011.

[83] Korat home page. `http://mir.cs.illinois.edu/korat/`.

[84] Y. Kwon and G. Agha. Performance evaluation of sensor networks by statistical modeling and euclidean model checking. *TOSN*, 9(4):39, 2013.

[85] Y. Kwon and G. A. Agha. Verifying the evolution of probability distributions governed by a dtmc. *IEEE Trans. Software Eng*, 37(1):126–141, 2011.

[86] S. Lauterburg, M. Dotta, D. Marinov, and G. Agha. A framework for state-space exploration of Java-based actor programs. In *Proc. of the 24th IEEE/ACM Conference on Automated Software Engineering (ASE 2009)*, pages 468–479, Auckland, New Zealand, Nov. 2009.

[87] S. Lauterburg, R. K. Karmani, D. Marinov, and G. Agha. Basset: A tool for systematic testing of actor programs. In *Proc. of the 18th ACM SIGSOFT International Symposium on the Foundations of Software Engineering, Formal Research Demonstration (FSE Demo 2010)*, pages 363–364, Santa Fe, NM, Nov. 2010.

[88] S. Lauterburg, R. K. Karmani, D. Marinov, and G. Agha. Evaluating ordering heuristics for dynamic partial-order reduction techniques. In *Proc. of the Fundamental Approaches to Software Engineering (FASE 2010)*, pages 308–322, Paphos, Cyprus, Mar. 2010.

[89] S. T. Lauterburg. *Systematic Testing for Actor Programs*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, Aug. 2011.

[90] Y. Y. Lee, S. Harwell, S. Khurshid, and D. Marinov. Temporal code completion and navigation. In *Proc. of the 35th ACM/IEEE International Conference on Software Engineering, New Ideas and Emerging Results (ICSE NIER 2013)*, pages 1181–1184, San Francisco, CA, May 2013.

[91] B. Li, Z. Sun, K. Mechitov, G. Hackmann, C. Lu, S. Dyke, G. Agha, and B. F. Spencer. Realistic case studies of wireless structural control. In *ICCPS*, pages 179–188, 2013.

[92] J. Li, T. Nagayama, K. Mechitov, and B. F. Spencer. Efficient campaign-type structural health monitoring using wireless smart sensors. In *Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems*, volume 8345, 2012.

[93] C. Liu, X. Yan, L. Fei, J. Han, and S. P. Midkiff. SOBER: statistical model-based bug localization. In *ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 286–295, New York, NY, USA, 2005. ACM Press.

[94] S. Lu, Z. Li, F. Qin, L. Tan, P. Zhou, and Y. Zhou. BugBench: A benchmark for evaluating bug detection tools. In *the Workshop on the Evaluation of Software Defect Detection Tools (BUGS 2005)*, 2005.

[95] D. Marinov, A. Andoni, D. Daniliuc, S. Khurshid, and M. Rinard. An evaluation of exhaustive testing for data structures. Technical Report MIT-LCS-TR-921, MIT Computer Science and Artificial Intelligence Lab, Cambridge, MA, Sept. 2003.

[96] D. Marinov and S. Khurshid. TestEra: A novel framework for testing Java programs. In *Proc. of the 16th IEEE Conference on Automated Software Engineering (ASE 2001)*, pages 22–31, San Diego, CA, Nov. 2001.

[97] D. Marinov, S. Khurshid, S. Bugrara, L. Zhang, and M. C. Rinard. Optimizations for compiling declarative models into boolean formulas. In *Proc. of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005)*, volume 3569 of *LNCS*, pages 187–202, St. Andrews, UK, June 2005.

[98] D. Marinov and R. O'Callahan. Object equality profiling. In *Proc. of the 18th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2003)*, pages 313–325, Anaheim, CA, Oct. 2003.

[99] D. Marinov and W. Schulte. Workshop on state-space exploration for automated testing (SSEAT 2008). In *Proc. of the International Symposium on Software Testing and Analysis (ISSTA 2008)*, pages 315–316, Seattle, WA, July 2008.

[100] A. Milicevic, D. Jackson, M. Gligoric, and D. Marinov. Model-based, event-driven programming paradigm for interactive web applications. In *Proc. of the Fourth Annual ACM International Conference on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH), Onward! Research Papers (Onward! 2013)*, pages 17–36, Indianapolis, IN, Oct. 2013.

[101] A. Milićević, S. Misailović, D. Marinov, and S. Khurshid. Korat: A tool for generating structurally complex test inputs. In *Proc. of the International Conference on Software Engineering, Demo Papers (ICSE Demo 2007)*, pages 771–774, Minneapolis, MN, May 2007.

[102] S. Misailović, A. Milicević, N. Petrovic, S. Khurshid, and D. Marinov. Parallel test generation and execution with Korat. In *Proc. of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2007)*, Dubrovnik, Croatia, Sept. 2007.

[103] S. Misailović, A. Milićević, S. Khurshid, and D. Marinov. Generating test inputs for fault-tree analyzers using imperative predicates. In *the Workshop on Advances and Innovations in Systems Testing (STEP 2007)*, Memphis, TN, May 2007.

[104] T. Nagayama, H.-J. Jung, B. F. Spencer, S. Jang, K. Mechitov, S. Cho, M. Ushita, C.-B. Yun, G. Agha, and Y. Fujino. International collaboration to develop a structural health monitoring system utilizing wireless smart sensor network and its deployment on a cable-stayed bridge. In *Proceedings of the 5th World Conference on Structural Control and Monitoring*, 2010.

[105] T. Nagayama, P. Moinzadeh, K. Mechitov, M. Ushita, N. Makihata, M. Ieiri, G. Agha, B. F. Spencer, Y. Fujino, and J.-W. Seo. Reliable multi-hop communication for structural health monitoring. *Smart Structures and Systems*, 6(5):481–504, 2010.

[106] S. Negara, R. K. Karmani, and G. A. Agha. Inferring ownership transfer for efficient message passing. In *PPOPP*, pages 81–90, 2011.

[107] A. Nistor, Q. Luo, M. Pradel, T. R. Gross, and D. Marinov. Ballerina: Automatic generation and clustering of efficient random unit tests for multithreaded code. In *Proc. of the 34th ACM/IEEE International Conference on Software Engineering (ICSE 2012)*, pages 727–737, Zurich, Switzerland, June 2012.

[108] A. Nistor, D. Marinov, and J. Torrellas. Light64: Lightweight hardware support for race detection during systematic testing of parallel programs. In *Proc. of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2009)*, pages 541–552, New York City, NY, Dec. 2009.

[109] A. Nistor, L. Song, D. Marinov, and S. Lu. Toddler: Detecting performance problems via similar memory-access patterns. In *Proc. of the 35th ACM/IEEE International Conference on Software Engineering (ICSE 2013)*, pages 562–571, San Francisco, CA, May 2013.

[110] R. Panwar and G. Agha. A methodology for programming scalable architectures. *J. Parallel Distrib. Comput*, 22(3):479–487, 1994.

[111] R. Panwar, W. Kim, and G. Agha. Parallel implementations of irregular problems using high-level actor language. In *IPPS*, pages 857–862, 1996.

[112] Parasoft. Jtest version 5.1. Online manual, July 2004. `http://www.parasoft.com/`.

[113] ReAssert Home Page. `http://mir.cs.illinois.edu/reassert/`.

[114] ReEx Home Page. `http://mir.cs.illinois.edu/reex/`.

[115] S. Ren and G. Agha. Rtsynchronizer: Language support for real-time specifications in distributed systems. In *Workshop on Languages, Compilers, & Tools for Real-Time Systems*, pages 50–59, 1995.

[116] S. Ren, G. Agha, and M. Saito. A modular approach to programming distributed real-time systems. *J. Parallel Distrib. Comput*, 36(1):4–12, 1996.

[117] J. Rice, K. Mechitov, S.-H. Sim, T. Nagayama, S. Jang, R. Kim, B. F. Spencer, G. Agha, and Y. Fujino. Flexible smart sensor framework for autonomous structural health monitoring. *Smart Structures and Systems*, 6(5):423–438, 2010.

[118] J. Rice, K. Mechitov, S.-H. Sim, B. F. Spencer, and G. Agha. Enabling framework for structural health monitoring using smart sensors. *Structural Control and Health Monitoring*, 18(5):574–587, 2011.

[119] J. Rice, K. Mechitov, B. F. Spencer, and G. Agha. Autonomous smart sensor network for full-scale structural health monitoring. *Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems*, 7647(1), 2010.

[120] RTR Home Page. `http://mir.cs.illinois.edu/rtr/`.

[121] K. Sen and G. Agha. Automated systematic testing of open distributed programs. In L. Baresi and R. Heckel, editors, *FASE*, volume 3922 of *Lecture Notes in Computer Science*, pages 339–356. Springer, 2006.

[122] K. Sen, D. Marinov, and G. Agha. CUTE: A concolic unit testing engine for C. In *Proc. of the 5th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2005)*, pages 263–272, Lisbon, Portugal, Sept. 2005.

[123] K. Sen, G. Rosu, and G. Agha. Runtime safety analysis of multithreaded programs. In *ESEC / SIGSOFT FSE*, pages 337–346. ACM, 2003.

[124] K. Sen, G. Rosu, and G. Agha. Online efficient predictive safety analysis of multithreaded programs. In K. Jensen and A. Podelski, editors, *TACAS*, volume 2988 of *Lecture Notes in Computer Science*, pages 123–138. Springer, 2004.

[125] K. Sen, A. Vardhan, G. Agha, and G. Rosu. Efficient decentralized monitoring of safety in distributed systems. In A. Finkelstein, J. Estublier, and D. S. Rosenblum, editors, *ICSE*, pages 418–427. IEEE Computer Society, 2004.

[126] K. Sen, M. Viswanathan, and G. Agha. Statistical model checking of black-box probabilistic systems. In *CAV*, pages 202–215, 2004.

[127] K. Sen, M. Viswanathan, and G. Agha. On statistical model checking of stochastic systems. In *CAV*, pages 266–280, 2005.

[128] Setac Home Page. http://mir.cs.illinois.edu/setac/.

[129] R. Sharma. Guidelines for coverage-based comparisons of non-adequate test suites. Master's thesis, University of Illinois at Urbana-Champaign, Urbana, IL, Dec. 2013.

[130] R. Sharma, M. Gligoric, A. Arcuri, G. Fraser, and D. Marinov. Testing container classes: Random or systematic? In *Proc. of the Fundamental Approaches to Software Engineering (FASE 2011)*, pages 262–277, Saarbrucken, Germany, Mar. 2011.

[131] R. Sharma, M. Gligoric, V. Jagannath, and D. Marinov. A comparison of constraint-based and sequence-based generation of complex input data structures. In *the 2nd Workshop on Constraints in Software Testing, Verification and Analysis (CSTVA 2010)*, pages 337–342, Paris, France, Apr. 2010.

[132] J. Siddiqui, D. Marinov, and S. Khurshid. Optimizing a structural constraint solver for efficient software checking. In *Proc. of the 24th IEEE/ACM Conference on Automated Software Engineering (ASE 2009)*, Auckland, New Zealand, Nov. 2009. (Short paper.).

[133] J. H. Siddiqui, D. Marinov, and S. Khurshid. Lightweight data-flow analysis for execution-driven constraint solving. In *Proc. of the Fifth IEEE International Conference on Software Testing, Verification and Validation (ICST 2012)*, pages 91–100, Montreal, Canada, Apr. 2012.

[134] A. Sobeih, M. d'Amorim, M. Viswanathan, D. Marinov, and J. C. Hou. Assertion checking in J-Sim simulation models of network protocols. *Simulation: Transactions of The Society for Modeling and Simulation International*, 86(11):651–673, Nov. 2010.

[135] A. Sobeih, M. Viswanathan, D. Marinov, and J. Hou. Finding bugs in network protocols using simulation code and protocol-specific heuristics. In *Proc. of the 7th International Conference on Formal Engineering Methods (ICFEM 2005)*, volume 3785 of *LNCS*, pages 235–250, Manchester, UK, Nov. 2005.

[136] K. Stobie. Advanced modeling, model based test generation, and Abstract state machine Language (AsmL). Seattle Area Software Quality Assurance Group, http://www.sasqag.org/pastmeetings/asml.ppt, Jan. 2003.

[137] D. C. Sturman and G. Agha. A protocol description language for customizing semantics. In *SRDS*, pages 148–157, 1994.

[138] K. Sullivan, J. Yang, D. Coppit, S. Khurshid, and D. Jackson. Software assurance by bounded exhaustive testing. In *Proc. of the International Symposium on Software Testing and Analysis (ISSTA)*, 2004.

[139] L. Tan, X. C. Zhang, X. Ma, W. Xiong, and Y. Zhou. AutoISES: Automatically inferring security specifications and detecting violations. In *Proc. of the 17th USENIX Security Symposium*, July-August 2008.

[140] S. H. Tan. @tComment: Testing Javadoc comments to detect comment-code inconsistencies. Master's thesis, University of Illinois at Urbana-Champaign, Urbana, IL, May 2012.

[141] S. H. Tan, D. Marinov, L. Tan, and G. T. Leavens. @tComment: Testing Javadoc comments to detect comment-code inconsistencies. In *Proc. of the Fifth IEEE International Conference on Software Testing, Verification and Validation (ICST 2012)*, pages 260–269, Montreal, Canada, Apr. 2012.

[142] S. Tasharofi, R. K. Karmani, S. Lauterburg, A. Legay, D. Marinov, and G. Agha. TransDPOR: A novel dynamic partial-order reduction for testing actor programs. In *Proc. of the joint international conference Formal Methods for Open Object-Based Distributed Systems and Formal Techniques for Networked and Distributed Systems (FMOODS & FORTE 2012)*, volume 7273 of *LNCS*, pages 219–234, Stockholm, Sweden, June 2012.

[143] P. Thati, R. Ziaei, and G. Agha. A theory of may testing for actors. In B. Jacobs and A. Rensink, editors, *FMOODS*, volume 209 of *IFIP Conference Proceedings*, pages 147–162. Kluwer, 2002.

[144] P. Thati, R. Ziaei, and G. Agha. A theory of may testing for asynchronous calculi with locality and no name matching. In *AMAST*, pages 223–238, 2002.

[145] Toddler home page. http://mir.cs.illinois.edu/toddler/.

[146] UDITA home page. http://mir.cs.illinois.edu/udita/.

[147] A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. Actively learning to verify safety for FIFO automata. In K. Lodaya and M. Mahajan, editors, *FSTTCS*, volume 3328 of *Lecture Notes in Computer Science*, pages 494–505. Springer, 2004.

[148] A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. Learning to verify safety properties. In *ICFEM*, pages 274–289, 2004.

[149] C. A. Varela and G. Agha. Programming dynamically reconfigurable open systems with salsa. *SIGPLAN Notices*, 36(12):20–34, 2001.

[150] M. Veanes, C. Campbell, W. Schulte, and N. Tillmann. Online testing with model programs. In *Proceedings of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 273–282, New York, NY, 2005. ACM Press.

[151] N. Venkatasubramanian, G. Agha, and C. L. Talcott. Scalable distributed garbage collection for systems of active objects. In Y. Bekkers and J. Cohen, editors, *IWMM*, volume 637 of *Lecture Notes in Computer Science*, pages 134–147. Springer, 1992.

[152] N. Venkatasubramanian, C. L. Talcott, and G. Agha. A formal model for reasoning about adaptive qos-enabled middleware. *ACM Trans. Softw. Eng. Methodol*, 13(1):86–147, 2004.

[153] W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda. Model checking programs. *Automated Software Engineering*, 10(2):203–232, April 2003.

[154] X. Xiao, T. Xie, N. Tillmann, and J. de Halleux. Precise identification of problems for structural test generation. In *Proceedings of the International Conference on Software Engineering*, pages 611–620, 2011.

[155] T. Xie, D. Marinov, and D. Notkin. Rostra: A framework for detecting redundant object-oriented unit tests. In *Proc. of the 19th IEEE International Conference on Automated Software Engineering*, Sept. 2004.

[156] T. Xie, D. Marinov, W. Schulte, and D. Notkin. Symstra: A framework for generating object-oriented unit tests using symbolic execution. In *Proc. of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2005)*, pages 365–381, Edinburgh, UK, Apr. 2005.

[157] T. Xie, J. Zhao, D. Marinov, and D. Notkin. Detecting redundant unit tests for AspectJ programs. In *Proc. of the 17th IEEE International Symposium on Software Reliability Engineering (ISSRE 2006)*, pages 179–190, Raleigh, NC, Nov. 2006.

[158] D. Yuan, H. Mai, W. Xiong, L. Tan, Y. Zhou, and S. Pasupathy. SherLog: Error diagnosis by connecting clues from run-time logs. In *ASPLOS 2010*, pages 143–154, Mar. 2010.

[159] L. Zhang, M. Gligoric, D. Marinov, and S. Khurshid. Operator-based and random mutant selection: Better together. In *International Conference on Automated Software Engineering*, pages 92–102, 2013.

[160] L. Zhang, D. Marinov, L. Zhang, and S. Khurshid. Regression mutation testing. In *Proc. of the ACM International Symposium on Software Testing and Analysis (ISSTA 2012)*, pages 331–341, Minneapolis, MN, July 2012.

[161] L. Zhang, D. Marinov, and S. Khurshid. Faster mutation testing inspired by test prioritization and reduction. In *Proc. of the ACM International Symposium on Software Testing and Analysis (ISSTA 2013)*, pages 235–245, Lugano, Switzerland, July 2013.

[162] L. Zhang, D. Marinov, L. Zhang, and S. Khurshid. An empirical study of JUnit test-suite reduction. In *Proc. of the 22nd IEEE International Symposium on Software Reliability Engineering (ISSRE 2011)*, pages 170–179, Hiroshima, Japan, Nov. 2011.

[163] R. Ziaei and G. Agha. Synchnet: A petri net based coordination language for distributed objects. In *GPCE*, pages 324–343, 2003.

List of Personnel

1. Gul Agha; University of Illinois at Urbana-Champaign; PI
2. Darko Marinov; University of Illinois at Urbana-Champaign; co-PI
3. Daniel Jackson; Massachusetts Institute of Technology; PI

List of Collaborators

(in Past at Least 48 Months, outside of UIUC and MIT)

1. Nazareno Aguirre; UNRC, Argentina
2. Amr Ahmed; CMU
3. Amin Alipour; Oregon State University
4. Kumar Apurva; IBM, India
5. Andrea Arcuri; Simula Lab, Norway
6. Mark Astley; Two Sigma Investments
7. Paulo Barros; UFPE, Brazil
8. Don Batory; UT Austin
9. Farnaz Behrang; Auburn University
10. Valeria Bengolea; UNRC, Argentina
11. Tom Brown; Google
12. Christian Callsen; Sun Microsystems
13. Rohit Chadha; INRIA-Saclay
14. Po-Hao Chang; RiverGlass Inc.
15. Liping Chen; Microsoft Corporation
16. Alcino Cunha; U. Minho, Portugal
17. Marcelo d'Amorim; UFPE, Brazil
18. Christo Devaraj; Google
19. Danny Dig; Oregon State University
20. Bill Donkervoet; Intel Corporation
21. Mirco Dotta; EPFL, Switzerland
22. Gordon Fraser; University of Sheffield, UK
23. Marcelo F. Frias; ITBA, Argentina
24. Svend Frolund; Software (Denmark)
25. Juan Pablo Galeotti; Universidad De Buenos Aires, Argentina
26. Mark Greenstreet; University of British Colombia

27. Michael Greenwald; Stanford University
28. Alex Groce; Oregon State University
29. Tihomir Gvero; EPFL, Switzerland
30. Thomas R. Gross; ETHZ, Switzerland
31. Munawar Hafiz; Auburn University
32. MyungJoo Ham; Samsung Research
33. Jianye Hao; SUTD, Singapore
34. Sam Harwell; UT Austin
35. John Holland; University of Michigan
36. Masataka Ieiri; University of Tokyo
37. Michael Jackson; independent consultant, London
38. Vilas Jagannath; Optiver, LLC
39. Nadeem Jamali; University of Saskatchewan
40. Myeong Jang; Samsung Research
41. Shinae Jang; Univ. of Connecticut
42. Hongki Jo; Univ. of Arizona
43. Sun Jun; SUTD, Singapore
44. Randy Katz; University of California at Berkeley
45. Rick Kazman; Univ. of Hawaii at Manoa
46. Assaf Kfoury; Boston University, Boston
47. Shadi Abdul Khalek; UT Austin
48. Sanjeev Khanna; University of Pennsylvania
49. Sarfraz Khurshid; UT Austin
50. Chang Hwan Peter Kim; UT Austin
51. WooYoung Kim; Intel Corporation
52. Mathew Kirn; Microsoft
53. Vijay Korthikanti; Synopsys
54. Viktor Kuncak; EPFL, Switzerland

55. Youngmindvisee Kwon; Microsoft Corporation
56. Timo Latvala; Nokia
57. Steven Lauterburg; Salisbury University
58. Gary T. Leavens; University of Central Florida
59. Axel Legay; INRIA, France
60. Shan Lu; University of Wisconsin–Madison
61. Sam Midkiff; Purdue University
62. Lynette Millett; National Academies, Washington DC
63. Mehdi Mirzaaghaei; University of Lugano, Switzerland
64. Peter C. Mehlitz; NASA Ames
65. Andrea Mocci; Politecnico di Milano, Italy
66. Sherin Moussa; Ain Shams University
67. Ushita Mitsushi; University of Tokyo
68. Tomonori Nagayama; University of Tokyo
69. Mehwishr Nagda; Los Alamos National Lab
70. Makihata Noritoshi; University of Tokyo
71. Jeffrey Overbey; Auburn University
72. John Palmer; IBM Almaden Research Center
73. Karl Palmskog; Stockholm University
74. Rajendra Panwar; Fenwick and West
75. Corina Pasareanu; NASA Ames and CMU-Silicon Valley
76. Anna Patterson; Venture Capitalist
77. Jean-Francois Perrot; University Pierre et Marie Curie
78. Mauro Pezzè; University of Lugano, Switzerland
79. Michael Pradel; ETHZ, Switzerland
80. Derek Rayside; U. Waterloo, Canada
81. Reza Razavi; University of Luxembourg
82. Shangping Ren; Illinois Institute of Technology

83. Mark Reynolds; Boston University, Boston
84. Jennifer Rice; University of Florida
85. Gregg Rothermel; University of Nebraska-Lincoln
86. Mooly Sagiv; Tel Aviv University
87. Koushik Sen; Univ. of California at Berkeley
88. Ju-Won Seo; KAIST
89. Junaid Siddiqui; UT Austin and LUMS School of Science and Engineering, Pakistan
90. Sung-Han Sim; KAIST
91. Linhai Song; University of Wisconsin–Madison
92. Sabrina Souto; UFPE, Brazil
93. Daniel Sturman; Google
94. Sameer Sundresh; Everpix
95. Carloyn Talcott; SRI
96. Lin Tan; University of Waterloo, Canada
97. Shin Hwei Tan; National University of Singapore, Singapore
98. Prasanna Thati; Goldman Sachs
99. Martyn Thomas; independent consultant, Bath, UK
100. Emina Torlak; UC Berkeley, CA
101. Predrag Tosic; Univ. of Houston
102. Thesis Uttamchandani; IBM Research
103. Abhay Vardhan; Google
104. Carlos Varela; Rensselear Polytechnic Institute
105. Nalini Venkatasubramania; University of California at Irvine
106. Guowei Yang; UT Austin
107. Masatoshi Yoshida; Toshiba, Japan
108. Fujino Yozo; University of Tokyo
109. Chaoqiang Zhang; Oregon State University
110. Lingming Zhang; UT Austin
111. Lu Zhang; Peking University, China
112. Mahmood Ziaei; Google

4 Data Management Plan

All three PIs have a substantial track record in disseminating the results of their work through presentations, publications, open-source code, and educational activities. The results of this project will be likewise broadly disseminated.

4.1 Type of Data and Other Research Products

We anticipate that the proposed project will generate the following types of data:

1. *Research articles and technical reports* representing the research results and discoveries (such as programming language specifications, semantics, parallel algorithms, compilation techniques, formal methods, and software) that are generated as a result of scientific activities by the PIs and their students. (See Section 5 for a more precise list of specific activities.)

2. *Educational materials* such as lectures slides, course materials, and video recordings of presentations of the results of the scientific and engineering research conducted in the project.

3. *Software and experimental results* generated in the process of implementing and evaluating the techniques and tools developed.

4.2 Access and Sharing Policies

All final research results will be made publicly accessible for other researchers. As we have done in the past, we anticipate making a version of the significant software developed under the project to be available as open source to other researchers. (Section 5.3 describes in more detail several software projects that the PIs publicly released into open source.)

4.3 Policies and Provisions for Re-Use, Re-Distribution, and the Production of Derivatives

Research products, educational materials, and data collections generated by activities of this project will be provided free for use and re-use by research and academic communities, given appropriate acknowledgment and waiver of liability.

4.4 Archiving and Dissemination

Research articles will be submitted to archival publications. Significant data will be maintained in a repository for a reasonable period of time of at least up to three years following the end of the project.

5 Collaboration Plan

An important strength of our proposal is that our team combines extensive and complementary expertise in parallel programming (Agha), modeling languages (Jackson), and software testing (Marinov). (See Section 1.2 for more about our background.) This combination enables us to collectively address problems that we could not address individually.

5.1 Research Tasks with Roles of Project Participants

Table 1 lists the activities that form our statement of work and shows the timeline of our proposed research. We are structuring our work around three thrusts: Input Language (Section 2.1), Execution Platform (Section 2.2), and Development Toolset (Section 2.3). For each of the three thrusts, we tabulate the main tasks and the years in which we will work on these tasks. Each thrust will be co-led by two PIs: Input Language by Jackson and Agha, Execution Platform by Agha and Marinov, and Development Toolset by Marinov and Jackson. Each thrust will involve students from the two PIs closely collaborating, with all the students from all three PIs working together on building shared infrastructure and evaluating our results. Jackson and Marinov have already had a pair of their students (from MIT and UIUC) working together, so far producing a joint paper [100] and releasing open-source code [1]. Agha and Marinov have co-supervised one MS [67] and one PhD dissertation [59] and have jointly worked with eight more students on six papers [62, 86–88, 122, 142].

This project will ideally support three student RAs each year. All the students will jointly work on the shared components for the entire project, and each student will be the primary lead for one of the three thrusts. The students who already helped with some preliminary results for this proposal have expertise in these thrusts, Aleksandar Milicevic with modeling languages for the Input Language [100], Peter Dinges on parallel programming for the Execution Platform [20, 36, 37], and Milos Gligoric with the Development Toolset [49, 51].

	Thrusts		
Year	Input Language (leads: Jackson and Agha)	Execution Platform (leads: Agha and Marinov)	Development Toolset (leads: Marinov and Jackson)
1	Define the user-level language (syntax and semantics)	Develop a simple automatic translator from the user-level language to the execution engine	Develop techniques for testing at the execution level (handling events from the user-level language and constraints from the actor language)
2	Develop extensive libraries for the user-level language; provide translation mechanisms for policies and pragmas	Develop an efficient execution engine framework on top of an existing actor library (Scala)	Develop techniques for model checking the translated actor programs
3	Improve the design of the user-level language based on initial experiences with case studies	Improve the execution engine to support efficient runtime monitoring of distributed properties, optimizations for scalable execution, and integration of constraints	Develop tools to map the results of testing and model checking from the execution level back to the input level

Table 1: Research timeline for the main tasks; *evaluation*, including development of several applications in the proposed paradigm, is cross-cutting through all the years and thrusts

5.2 Coordination Plan

The teams from UIUC (PIs Agha and Marinov) and MIT (PI Jackson) will co-operate closely. PIs Agha and Marinov have been collaborating for years, publishing several joint papers [62, 86–88, 122, 142] and co-supervising student theses [59, 67]. PIs Jackson and Marinov collaborated both while Marinov was a student at MIT (even though not supervised directly by Jackson) [75,76] and more recently while at different institutions [100].

We will hold regular teleconferences, over systems such as Skype, to discuss project progress between UIUC and MIT. Indeed, Skype already greatly helped us and our students to produce recent joint results [1, 100] that influenced the ideas for this proposal. We also plan to make several trips per year to have in-person meetings (at UIUC, MIT, or conferences) to keep us informed of one another's ideas, progress, and to coordinate software integration. Perspectives from different institutions will foster a mix of ideas that could potentially lead to unanticipated but significant contributions. Our budget includes appropriate items for travel.

5.3 Empirical Evaluation and Collaborative Development

Several activities are not explicitly shown in Table 1 but will be ongoing throughout the project duration, primarily *evaluation* of the developed techniques and tools. We plan to validate our proposed work by implementing, testing, and releasing novel apps using our proposed paradigm. Our students have already released one jointly developed mobile game [1], and we have released several other software projects (as listed in our biosketches).

This is a software-intensive project. Our software deliverables will include implementations of our language tools, translators, efficient execution engine, and testing tools. The software will be developed jointly by the students from UIUC and MIT. Our ongoing joint work has already used private software repositories (e.g., an SVN repo for this text). We plan to develop software for this project collaboratively by using one repository that we will make open to the public, most likely on Github. All the software we develop through the direct support of this grant will be open-source, released under a license to be decided (MIT or UIUC/NCSA). In the past, all our three groups have released several software systems under these licenses.

Over the years, Agha's group has released a number of programming language definitions, compilers and frameworks, as well as tools for simulation, verification and testing. These include *ActorFoundry*, Java framework for actor programming; *SALSA*, a general-purpose actor-oriented programming language especially designed to facilitate the development of dynamically reconfigurable open distributed applications; *SOTER*, an extensible static analysis and transformation tool that facilitates safe yet efficient message passing in actor programs; *Atomic Set Inferencer*, a tool that helps programmers convert multi-threaded Java programs from lock-based synchronization to atomic sets; *iLTL Checker*, a model checker for Multiple Discrete Time Markov Chains, and *jCUTE*, a unit test generator for multi-threaded Java programs (including those written in actors) using concolic execution. These software tools are available at `http://osl.cs.illinois.edu/software/index.html`. Marinov's group has released several software systems [13, 18, 56, 83, 113, 114, 128, 146] and has also contributed to widely-used open-source projects such as NASA's Java PathFinder [70, 153]. Likewise, all the research products of Jackson's group (including the Alloy Analyzer, Kodkod model finder, Forge program verifier, Moolloy optimization tool and Squander framework) have been publicly released, under the MIT license.