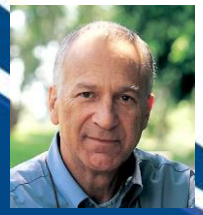
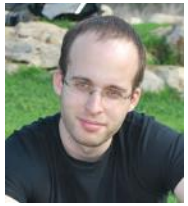
Towards Wise Computing:

An Initial Wise Development Environment for Behavioral Models

David Harel, Guy Katz, Rami Marelly, Assaf Marron

מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE

Talk Based on

D. Harel, G. Katz, R. Marelly and A. Marron,

“An Initial Wise Development Environment for Behavioral Models”, In Proc. of Modelsward, 2016.

Agenda

- The Wise Computing Vision
- An Initial Wise Development Framework

A Nagging Question

- Situation & Assumptions
  - Software development is very hard
    - including some theoretical results
  - And it is likely to remain so even when all tools from industry and academia are mature and available to all
  - But we are and will be able to create reliable complex systems

- Question:
  - So what is this “magic” that the humans will (still) do?

- Answers:
  - Hard work on routine tasks that were not automated yet
  - Complex tasks that we don’t know yet how to automate.

The "*Wise Computing*" Vision Summary

Enable IDEs to perform "all" SE tasks

that are presently considered uniquely human-centric

and require

sophistication / intelligence / wisdom.

Harel et al, *" Wise Computing: Towards Endowing System Development with True Wisdom ", arXiv,* 2015

The “Added Value” of Humans in SE

- Broad knowledge and experience – SE, domain, user’s “world”
- Unbounded access to formal artifacts and informal information
- Ability to discover emergent properties
- Association-driven action / initiative / proactivity
- Ability to prioritize, skip, or interrupt activities
- Dedication (e.g., thinking about something “all the time”)
- Creativity / Invention / Innovation / ”Thinking-outside-the-box”
- . . .

The “Added Value” of Humans in SE (cont.)

- . . .
- Applying external knowledge to a particular situation
- Judging emergent properties and behaviors as “good”, “bad”, etc.
- Visualizing hypothetical situations
- Work and communicate in multiple and orthogonal ways/abstractions
- Ability to ask questions
- Formalizing informal statements (= *“programming”*)
- Ability to repair/modify defective/lacking programs
- . . .

Wise Computing

- **Goal:**

  create an interactive , proactive and knowledgeable development framework (WDS)

  - **Accompanies all phases of development**
  - **Analyzes the model without being told**
  - **Autonomously raises concerns, makes suggestions, repairs**

- **The WDS becomes a member of the dev. team**

- **A long-term goal**

Related Work

- The Programmer's Apprentice:
  Rich, Waters, Shrobe et al, 70′s to early 90′s
  - Teach the computer how to model/program
  - Based on clichés

- "Towards a Programmer's Apprentice (Again)"
  Shrobe, Katz, Davis, 2015.
  - From NL requirements to code

- "A Long Way to Have Come and Still to Go" ,
  Cerf , CACM, 12/2014

- A variety of focused tools

Why Now?

- **A lot has changed**
  - **Stronger machines**
  - **Formal analysis tools**
  - **Powerful HCI and NL methods**
  - **New programming / modeling approaches: closer to how people describe behavior**
  - **…**

Recent Results

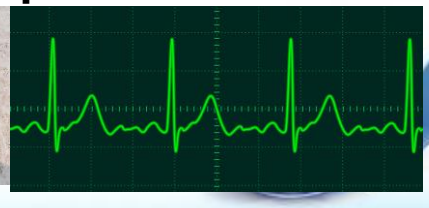
- An Initial Wise Development Suite (WDS)
- For Behavioral Models
  - Scenario-based programs in C++
  - System structure aligned with how people describe behavior
  - Amenable to incremental development and formal analysis

Components of an initial WDS: The three sisters

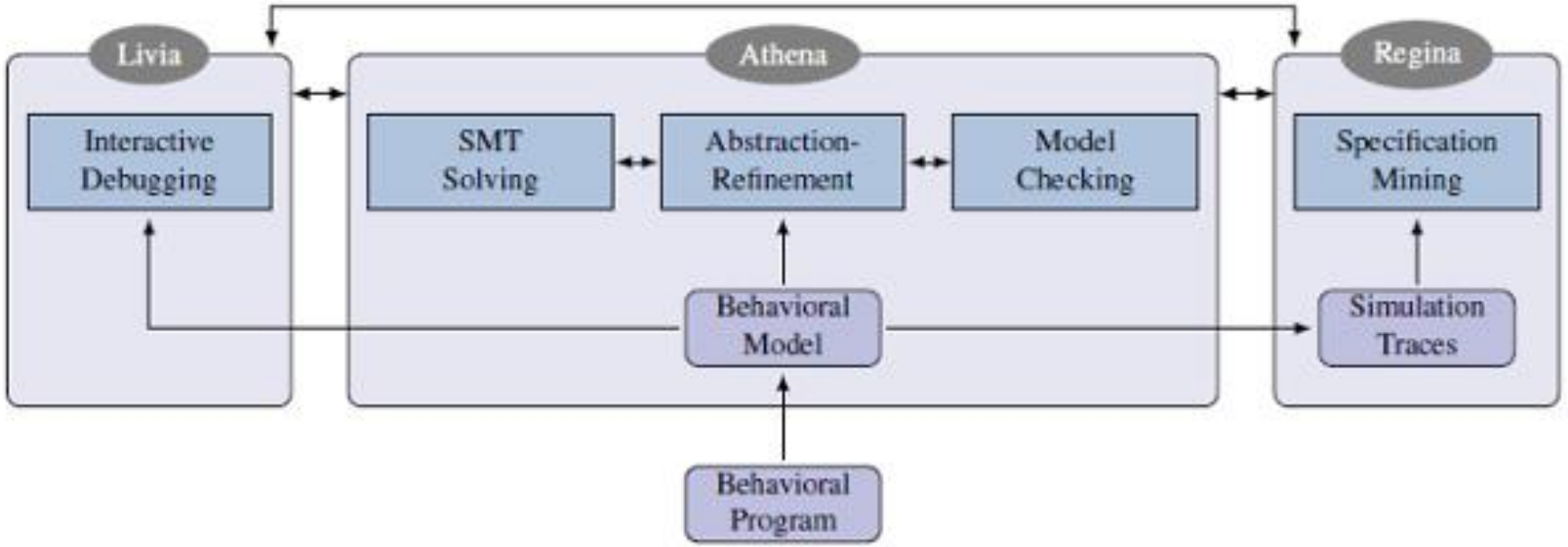
- **Athena:**
  - **Performs precise analysis of properties**
    - **General (e.g., deadlocks) and as suggested by Regina**
  - **Uses model-checking, SMT solving, abstraction-refinement…**

- **Regina:**
  - **Performs random/statistical simulations**
  - **Discovers emergent properties**
  - **Sends selected properties to Athena to validate**

- **Livia:**
  - **Performs live, run-time monitoring for desired properties**
  - **Run-time analysis, BMC**



An Initial WDS



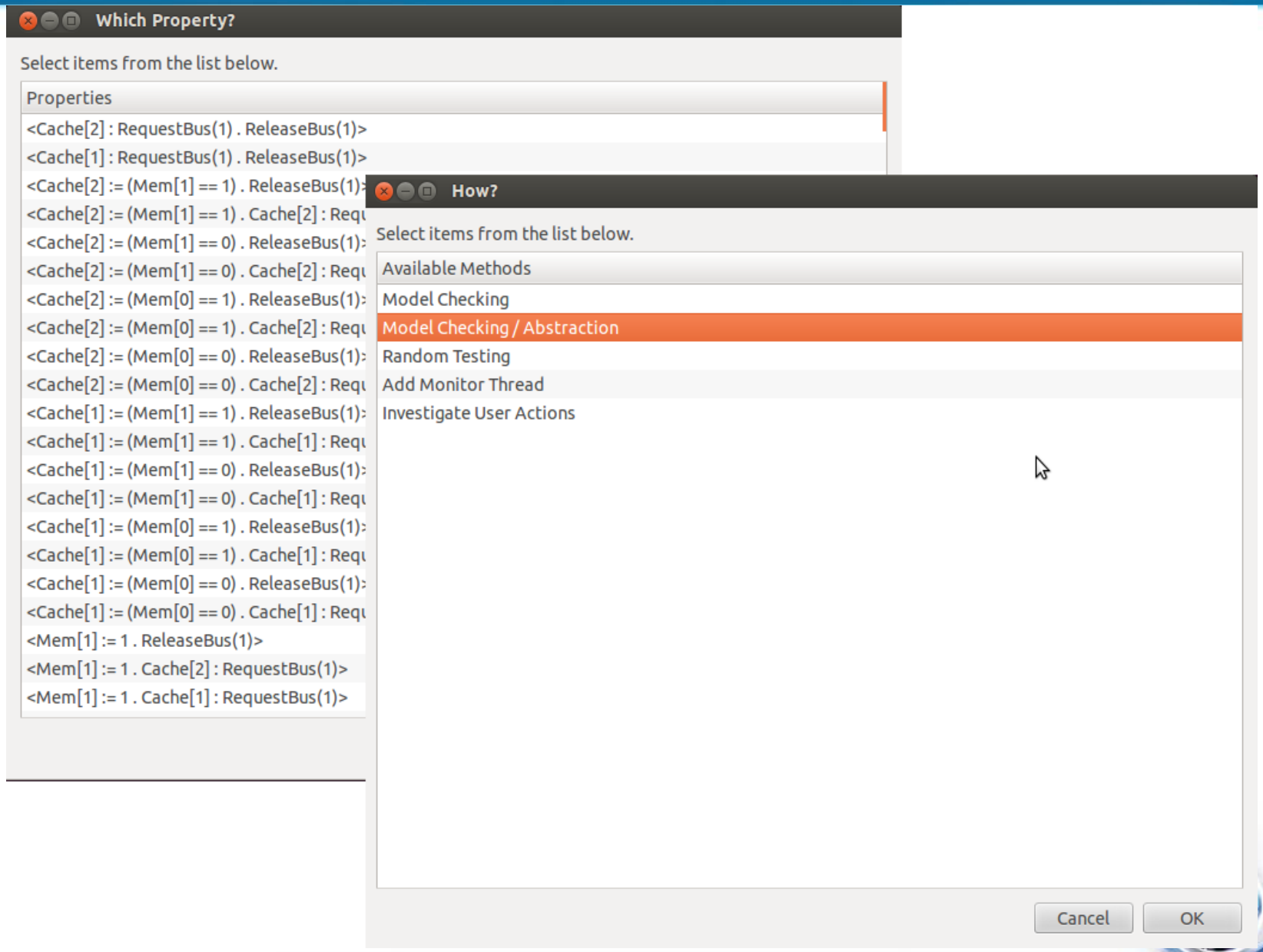
Development Workflow

- Constant analysis: interrupted and restarted with each compilation
  - Extract scenario models from code
  - Abstract the details and focus on inter-scenario comm.
  - Partition into modules – using patterns and “shared” events
  - Specification mining:
    - Run statiscal simulations on abstract model
    - Discover emergent properties: Bugs/Goals/Other?
  - Prioritize: heuristics / representatives / manual
  - Prove properties: MC, SMT, Abstraction-refinement, statistical testing
  - Throughout: interactively present results, explore deeper, repair

```
#ifndef __CacheOneReadHasBit_h__
#define __CacheOneReadHasBit_h__

#include "BThread.h"
#include "EventCodes.h"

class CacheOneReadHasBit : public BThread
{
public:
    CacheOneReadHasBit() : _getBitZero( *this, false )
    {
        _cacheOneReadEvents.insert( EventCodes::CACHE_ONE_EMPTY );
        _cacheOneReadEvents.insert( EventCodes::CACHE_ONE_BIT_0_IS_0 );
        _cacheOneReadEvents.insert( EventCodes::CACHE_ONE_BIT_0_IS_1 );
        _cacheOneReadEvents.insert(
        _cacheOneReadEvents.insert(
    }

    void entryPoint()
    {
        while ( true )
        {
            waitForRequests();

            if ( bitIsStored() )
            {
                reset();
                replyToRequest();
            }
            else
            {
                reset();
                waitForReply();
            }
        }
--:**-  CacheOneReadHasBit.h    Top
Beginning of buffer
```

Wise Framework Output

```
13:20:18        Selected property to check: ReleaseBus(1) <--> Cache[1] : RequestBus(1)
13:23:56        Model-checking based proof that property ReleaseBus(1) <--> Cache[1] : RequestBus(1) does not hold
13:23:56                Pc[1] : Write(Mem[1], 0)
13:23:56                Cache[1] == ?
13:23:56                pc[2] : Write(Mem[1], 0)
13:23:56                Cache[2] == ?
13:23:56                Cache[1] : RequestBus(1)
13:23:56                Mem[1] == 0
13:23:56                Cache[1] := (Mem[1] == 0)
13:23:56                ReleaseBus(1)
13:23:56                Cache[2] : RequestBus(1)
13:23:56                Mem[1] == 0
13:23:57                Cache[2] := (Mem[1] == 0)
13:23:57                ReleaseBus(1)
13:23:57                pc[2] : Success
13:23:57                pc[2] : Write(Mem[1], 0)
13:23:57                Cache[2] == (Mem[1] == 0)
13:23:57                pc[2] : Success
13:23:57                Pc[1] : Success
13:23:57                Pc[1] : Write(Mem[1], 0)
13:23:57                Cache[1] == (Mem[1] == 0)
13:23:57                pc[2] : Write(Mem[1], 0)
13:23:57                Cache[2] == (Mem[1] == 0)
13:23:57                pc[2] : Success
13:23:57                Pc[1] : Success
13:23:57                Pc[1] : Write(Mem[1], 0)
13:23:57                Cache[1] == (Mem[1] == 0)
13:23:57                pc[2] : Write(Mem[1], 0)
13:23:57                Cache[2] == (Mem[1] == 0)
13:23:57        Duration of model checking: 00:03:34
13:23:57
13:24:19        Selected property to check: Cache[2] : RequestBus(1) --> ReleaseBus(1)
13:26:46        Property Cache[2] : RequestBus(1) --> ReleaseBus(1) holds!
13:26:46        Duration of model checking: 00:02:23
13:26:46
```


Which Property?
Select items from the list below.
Properties
<Cache[2] : RequestBus(1) . ReleaseBus(1)>
<Cache[1] : RequestBus(1) . ReleaseBus(1)>
<Cache[2] := (Mem[1] == 1) . ReleaseBus(1)>
<Cache[2] := (Mem[1] == 1) . Cache[2] : Requ
<Cache[2] := (Mem[1] == 0) . ReleaseBus(1)>
<Cache[2] := (Mem[1] == 0) . Cache[2] : Requ
<Cache[2] := (Mem[0] == 1) . ReleaseBus(1)>
<Cache[2] := (Mem[0] == 1) . Cache[2] : Requ
<Cache[2] := (Mem[0] == 0) . ReleaseBus(1)>
<Cache[2] := (Mem[0] == 0) . Cache[2] : Requ
<Cache[1] := (Mem[1] == 1) . ReleaseBus(1)>
<Cache[1] := (Mem[1] == 1) . Cache[1] : Requ
<Cache[1] := (Mem[1] == 0) . ReleaseBus(1)>
<Cache[1] := (Mem[1] == 0) . Cache[1] : Requ
<Cache[1] := (Mem[0] == 1) . ReleaseBus(1)>
<Cache[1] := (Mem[0] == 1) . Cache[1] : Requ
<Cache[1] := (Mem[0] == 0) . ReleaseBus(1)>
<Cache[1] := (Mem[0] == 0) . Cache[1] : Requ
<Mem[1] := 1 . ReleaseBus(1)>
<Mem[1] := 1 . Cache[2] : RequestBus(1)>
<Mem[1] := 1 . Cache[1] : RequestBus(1)>
How?
Select items from the list below.
Available Methods
Model Checking
Model Checking / Abstraction
Random Testing
Add Monitor Thread
Investigate User Actions
Cancel
OK

A Case Study

- Modeled a cache coherence protocol
- Used the wise framework throughout development
- Wanted to answer:
  - Is the idea scalable?
  - Can we reach interesting conclusions on a larger program?
  - Is this type of development enjoyable?
- In our experience: yes, yes and a subjective yes
- Video available online (link in paper)

Example: An Emergent Property

- **"Cache 3 cannot acquire bus 2 twice without first releasing it"**
  - **Prioritized because it is a mutual exclusion property**
- **Framework abstracts away code relating to bus 1**
- **Accelerated verification due to abstraction:**
  - 21,000 **states instead of** 972,233
  - 30 **seconds instead of** 1620
- **Process completely autonomous**

Case-Study Conclusions

Showed useful proactive analysis as part of development flow

General Challenges in Wise Computing

- **Modeling and representation**
- **Analysis and synthesis techniques**
- **Human-computer discourse**

Thank You!