Methods Working Group
October 5-6 2006

Moderator: Azer Bestavros
Scribe: Cesar Munoz

Participants Thursday:
- Martha Matzke [MM]
- Allen Goldberg [AG]
- Reinhard Wilhelm [RW]
- Lui Sha [LS]
- Azer Bestavros [AB]
- Matthew Dwyer [MD]
- David von Oheimb [DO]
- Calton Pu [CP]
- Cesar Munoz [CM]
- Paul Miner [PM]

Participant's Interests:

[MM] Representing the National Coordination Office.

[AG] Recent project with Airbus. Run time monitoring, software fault protection, model-based development. Furthermore, code generators, notations that capture high level requirements, analysis tools that capture domain specific domains, static analysis tools.

[RW] Static program analysis, real time verification, redesign of systems to be more time-predictable.

[LS] : Dependable systems.

[AB] Formal verification, extended automata models, verification methods.

[MD] Recent DARPA project. Code analysis, systems understanding, model checking, static analysis, limits of these methods. How these methods can be combined. What can be done when these methods reach their limits. How the information produced by these tools can be used (may be in a different technique).

[DO] Formal methods, application in security areas. Current project with Boeing, lots of techniques to explore different properties.

[CP] Building systems that produce software with good properties, compilation tools, code generation.

[CM] Theorem proving, application of formal methods to Air Traffic Management systems.

[PM] Theorem proving, fault tolerance, Byzantine error modeling, hardware realization.

Discussion Thursday:

[RW] Details on the cooperation with Airbus: development of tools used for certification. Example of

successful project, Airbus people very happy with the results. For that reason, he doesn't share the pessimistic view of formal methods in industry.

[AB] Focus the discussion on lessons learned. Issue for discussion: scalability.

[DO] Developers need tools that they can use. Reasonable effort to use these tools.

[AG] Two kinds of methods: General methods that provide minimum level of specification and specific methods for particular kinds of analyses. Trade-off on how much extra effort are developers able to put.

[AB] To summarize: pessimistic view is due to the developer's impression that formal methods require a parallel track development. It's important to seamlessly integrate formal methodologies to the development process.

[AG] The cost of formal methods tend to be overestimate. If the cost is very significant, they are not used.

[RW] For the research group the Airbus project was a extremely costly process. However, for Airbus the cost was not very high. They have formal specifications in Scade that simplified the task. Developers expect a push button tool, otherwise is very hard to convince them.

[PM] Need for formal notations that are familiar to the people that need them. Model checking is an example. For that reason they are accepted. On the other hand, general purpose theorem provers don't get that traction. Theorem provers are very expressive. There are many representations of the same specification.

[DO] We need good models. A good model is one that is easy to validate. It's not always possible to have a good model.

[PM] Important to align the model with the problem framework.

[AB] Does this explain the “failure” of theorem proving?

[PM] Useful models are close to the problem domain. Same thing for computational models.

[RW] Airbus project revealed a strange effect not discussed in the literature of static analysis. [Explanation of a problem related to cache memory and unroll of loops]. This is an example that general static analysis techniques don't always work on specific problems.

[MD] Models are useful when they are domain specific. In the context of model checking, existing partial order reduction don't work for Java. [Explanation of a problem related to the use of the heap in Java]. Important to understand the domain to customize the methods. To be cost effective, methods have to be customized to target the domain.

[AB] Methods must be informed of the domain.

[LC] General model are useless. Methods have to capture the domain. This is really crucial.

[AG] Airbus seems to be in better situation than the US. They use Scade, which generates code. Why

they are there and the US not. Guess: US main interest is in tools with limited code generation capabilities.

[LC] Model integration is quite important. Airbus current situation isn't that good. Airbus and their partners use slightly different tools that do not integrate easily.

[AG] Does this mean that applications in the US they are more integrated in the system level ?

[LC] Studies show that software hazard problems in Boeing are almost the same as in Airbus.

[RW] Airbus project originated in a department whose mission is to search for new methods and tools. They asked Patrick Cousot and he suggested solutions.

[MD] Remark that Cousot didn't have a tool. Polyspace was already in the market but generated more than 300 false alarms.

[LC] [List of projects where he has participated]

[AB] What are the lessons learned ?

[LC] You need very precise high quality models. Otherwise it's not going to work.

[AB] Garbage in- garbage out.

[LC] Completeness of a model is extremely hard to prove. Anther lesson: complexity control. If you cannot reduce the complexity, you cannot model the problem correctly. Notion of interactive complexity. If you allow for arbitrary interaction, you cannot show anything. Complexity control is important. Another lesson: stability control. Stability of hardware and software is essential.

[AB] Lesson learned: Abstract methods don't work.

[CM] Problem is not abstract methods per se but methods that cannot be easility tailored to specific domain problems.

[DO] You can lose precision if the right model isn't used.

[CP] Even in the system software, specific languages ... [scribe lost track]

[MD] A good model has an effect on the system.

[AB] To summarize, the time to do modeling is when the system is being developed. Modeling is not an after thought.

[AB] How about technologies ? Typing ? Scalability comes from programming languages that force developers to use types.

[LC] Complexity reduction is very important.

[AB] Networking idea: the hourglass model. Low layers assume IP and applications build on top of IP.

Java and the virtual machine the “IP” intermediate representation.

[PM] Remark that programming language type systems don't support real time guarantees.

[AB] Maybe there is an alternative technology.

[LC] Let's people do what they want. Complexity reduction is important.

[AB] Complexity reduction is the hourglass metaphor. Other techniques?

[LC] Stability.

[AB] Stability is safety net.

[PM] Other lesson: Composable methods. You cannot achieve composability unless your architecture support it. Architecture has to be designed to be composable.

[LC] Design for compositionality.

[AB] There are properties that don't compose.

[DO] Timing schemes don't work.

[AB] On any application or in your example ?

[AG] Code generators don't have to generate all the code, only the hard code. If you trust this code, then the rest can be localized.

[LC] Java and C add a lot of complexity but provide low value. Lesson learned: don't use Java. Java generates high complexity but give little value.

[AB] Future challenges ?

[CP] More systematic approach to dependency management. We don't have tools for dependency tracking.

[LC] Dependency relations create a lot of problems.

[CP] You have to 1) identify dependencies, 2) make them explicit, and 3) apply tools.

[AG] Non-dependency rather than dependency analysis.

[AB] Time is over. What is the plan for tomorrow ?

Participants Friday:

- Martha Matzke [MM]
- Allen Goldberg [AG]
- Reinhard Wilhelm [RW]

- Lui Sha [LS]
- Azer Bestavros [AB]
- Matthew Dwyer [MD]
- Calton Pu [CP]
- Cesar Munoz [CM]
- Paul Miner [PM]
- Paul Jones [PJ]
- Bill Spees [BS]
- John Rushby [JR]

Discussion Friday:

[AB] Afterthoughts ?

[CP] Liu made a point yesterday about reduction of state space. This reduction is needed because of tools limitations.

[RW] Abstraction is a technique for state reduction. Abstraction is needed to reduce complexity while keeping properties.

[CP] Interference adds complexity.

[AB] [Showing slides from yesterday discussion]. Participants backgrounds [see slides]

[LS] Cross-domain analysis is a mess. Design what you can analyze.

[AB] Very strong statement.

[LS] This is a goal.

[AB] Is that a lesson ?

[RW] In the hour-glass analogy, resources should be part of the interface.

[AB] The intermediate level should not change frequently.

[MD] Wonder if people using these models are willing to tolerate huge variations on top and bottom levels compared to mid-level.

[LS] Another aspect of complexity reduction: keep model simple, but don't oversimplify. Interactions involving critical components mus be simple.

(discussion on examples of critical components interfering)

[AB] Don't forget education.

[CP] One challenge: How to reconcile everything we know when building a commercial operating system.

[JR] A market place is needed. Make certifiable systems COTS.

[LS] Safety means that bad thing cannot happen. Stability is related to degradation of operation.

[PM] Distinction between mechanism and policies. Is the policy that we are implementing the right one.

[LS] We need methods that guarantee system stability.

[AB] What kind of methods ?

[JR] Verification methods and certification methods. Certification methods should cover the verification that systems degrade well.

[LS] Systems must be verifiable and able to handle error conditions. Discussion on FAA case.

[JR] Examples of redundancy and fault tolerance with complex interactions.

[LS] Need methods to maintain system safety and system stability.

[AB] Summary: Develop systems that tolerate bug interactions.

[LS] Systems that tolerate residual bugs.

[AB] All bugs are residual.

[JR] We should be able to anser the question of what interactions exist in the presence of failures. Some errors cannot be tolerated.

[John Hansman] Some errors can be managed. Distinguish between manageable residual bugs, which should be removed, and unmanageable bugs, for which strategies to mitigate their effects should be developed.

[JR] It's important the the strategies should not made the problem worse.

[PM] Failures should not break the interface. Failures that create interfaces are unmanageable.

[LS] The formal methods community has a very poor understanding of the issue.

[AB] Is that interface or interference ?

[AB] Integrating theories: timing, concurrency. How do we integrate them within a single framework.

[MD] Comment on education. Demand better high education is a good idea but we are a relatively small community.

[AB] Medical systems are less well understood. No conception of what happen with all the interactions within the human body. Avionics systems are trivial compared to medical systems.

[LS] We need a license for software critical software developers.

[PM] Economic problem here. In aerospace we don't have the demand. How other industries manage this? Intel and Microsoft invest a lot on education and universities. Idea: solicit an industry consortium on several domains: Medical, Automotive, ...

[CP] Maybe high-confidence system is the big umbrella.