The Highs and lows of deploying formal methods in industry

AVOCS 2014

Guy H. Broadfoot
Guy.Broadfoot@Silverdt.eu

• Disclaimer: I have no connection with Verum since September 2013.
• All information in this presentation reflect my opinions. I do not speak on behalf of Verum
• Any consistency between these opinions and those of Verum is purely coincidental.
• I have no knowledge of Verum's business development or product development since September 2013
Background and History

Target Market and Market requirements

Technology and Product

Business Model

The Good, the Bad and the Ugly

>40 years in SW development. In mid 1990's conceived the idea of combining MDD (model driven development) and CBE (Component-based SW Engineering) with formal verification and code generation into a single integrated SW development platform.

I built proof of concept (PoC) prototype; tried it out on complex manufacturing machine being developed at Assembleon in Veldhoven. SW was controller kernel controlling 22 PCs connected via internal network Very successful; SW ran first time and was bug-free during its life.


Technology further developed with Philippa Hopcroft, then postdoc researcher at Oxford, later head of research and IP at Verum. Support from Bill Roscoe, Michael Goldsmith and others at Oxford in addressing CSP theory issues and adapting FDR2
Market Requirements

- Target Market
  - Manufacturers of SW intensive products
  - Complex, event driven, reactive, long running
  - Not safety critical
- Must scale to very large systems
- Accommodate legacy code
- Must breakeven on the first project

- The way industry sees SW development tools – what does industry want from formal methods?
  - You cannot sell FM to industry – you have to sell real business value
  - Industry doesn't want FM and has no interest in them. Industry wants SW which is:
    - Cheaper
    - Quicker to make
    - “Better” quality
- Be clear about your target market/customers. Failure to do this is most common reason why start-ups fail. Do your market research before you spend your money.
  - Verum decided to exclude safety critical due to lack of people with domain knowledge and its complex regulatory environment. Non-safety critical is much larger addressable market. Probably this was a mistake.
- Scale: Very large systems
  - Legacy e.g., ASML = 30M lines of code. At 5locs per hour = 6M hours = 750K mds = 3,750 mys = 450M euros if every LOC only written once!
  - If you don’t breakeven on first project there won’t be a second project.
Market Requirements

- Models must be accessible to all stakeholders
  - No fancy notation
  - Teachable in days
  - Must be clear what questions are being asked
  - Must be clear what the answers mean
- Ready to run “Out of the box”
  - Integrated, push-button solution
  - Interoperability with other tool chains

- Models have to be clear to people who are not SW engineers and who have no technical skills or deep (any?) mathematical skills
- Product must be an integrated end-to-end solution. No-one wants to mess about with a box of bits and pieces.
- Interoperability with other SW environments and tools essential for deploying in industry. Also r/t platform independence.
What is ASD?

- **ASD is Software Design Automation technology using fully integrated formal verification methods**

- **ASD: Suite is a software engineering platform for:**
  - constructing complete and correct industrial scale systems from components *mathematically verified during design*

- **ASD provides:**
  - fully automated formal verification of specifications and designs
  - fully automatic code generation (C++, C#, MISRA C, Java)
  - easy integration into existing software development teams

- **ASD guarantees:**
  - semantic equivalence between specifications, designs, formal models and runtime behaviour of generated code

- The standard pitch and it’s pretty well all true – or was when I last had anything to do with it. Can’t speak for now.
There is no business case for formal verification unless executable code is generated from exactly the same model being verified.

Model has to serve two purposes, formal verification and code generation for runtime execution - really difficult to resolve conflict between:

- Too much information for FV modeling (not sufficiently abstract);
- Too little information for code generation (not sufficiently concrete)

No-one wants verified models – they want defect free, executable code. Period.

Code generation from (for example UML) makes no sense. If you don’t know models are correct, how do you know generated code is correct?

Test and debug at code level, not model level. Have to keep models in sync.

ASD code generated from verified models. The models are the “program source”, the generated source code is disposable. Errors reported in model terms at model level of abstraction. Models are “debugged” and corrected; source code regenerated when needed.

Huge productivity saving; most “debugging” like this done local to the developer, no giant builds etc.

Generated code reaches testing with 10 times fewer defects (TU/e report on PHC) than conventional methods.
What is ASD?

- Sequence-based Specification
  - basis of ASD Modelling Language
  - Regular expressions & Mealy machines
  - No formalism visible in models

- Eliminates:
  - Concurrency problems
  - Deadlocks
  - Live locks
  - Race conditions
  - Illegal behaviour
  - Interface mismatches
  - Unhandled errors
  - Incomplete behaviour

- Communicating Sequential Processes
  - Hoare, Roscoe (Oxford university)
  - Process Algebra + Finite State Model checking

- Source code in MISRA compliant C, C++, C# and JAVA
- JAVA code serialisable and mobile (for telecoms domain)
Two type of models – same notation
- IMs are embodiment of CBE and Parnas’ design by contract
- Behavior so Semantics as well as syntax.
- Used to generate property sets for verification plus define Implementation’s executable environment.
- Crucial to scalability and FV
- DMs embody component’s design. DMs are verified; code generated from DMs (interface declarations etc. from IMs)
- Emphasize dual role of IMs.
Typical organization of SW intense products

Key Point: the green bit is the SW for which testing is least effective due to complexity and non-det behavior when viewed as black box.
• Example product built with ASD for Philips Digital Pathology by CCM, ProDrive, Verum and others
• 12 moths available from scratch to end date.
• Drop-dead end date due to exhibition
• Fully functional, reliably working SW delivered on time and integrated with HW in days rather than the usual months
• Many suppliers; ASD IMs were used to define interfaces which were also interfaces between suppliers. Big reduction in integration problems; practically none.
• ASD used to make simulator for GUI supplier to test with. Know to be accurate simulator because it was verified against the IM used to define its externally visible behavior. Same IM as used to verify real implementation.
• The need to interoperate. There is no universal tool that does everything
Scalability crucial to realistic system development
ASD scales using compositional verification
Note: even if model checkers and theorem proving tools could handle giant systems in a single verification, it still wouldn’t work due to:
  • You never have all the designs; some belong to TPs.
  • You can’t structure the work of development and verification this way; verifying entire system and fixing errors much too difficult
  • Model checking state space expands exponentially with respect to model complexity
This is how ASD does it. Do architecture partitioning as per normal; then add ASD IMs as a means of adding precision (not detail) to architecture
Then develop implementations and FV as you go along
Cross-hatched interfaces are on the boundary between the formal and informal worlds.
• Reapply partitioning to make components simple enough for human to oversee and understand. Use usual goals of abstractions etc.
• Add ASD IMs for the internal interfaces.
• Verify component by component
• Repeat
How ASD Scales

Wafer Fab

Controller

Inspection Unit

Stage

Robot

Input Loadport

Output Loadport

Input Wafer Track

Stage Axis

Output Wafer Track

AVOCS 2014 – Guy H. Broadfoot
• Compositional verification relies on CSP’s compositionality – see Roscoe for more information.
• This is just an overview of principles.
• In practice, its much harder than it looks due to practical issues such as minimising model checking time.
• Structure of generated code.

• From top to bottom:
  • Generated Code
    Code that is specific to a given ASD model
  • ASD Runtime
    Code that is common to all ASD components BECAUSE they are ASD components. To preserve compositionality of non-compositional properties (e.g. deadlock freedom) components must be composed to form systems according to certain rules. This common ASD code encapsulates those rules. Delivered as source code, one for each target language.
  • O/S and Runtime Platform
    Depends on target language. E.g. dot net runtime, JAVA VM. For C++ its using Boost; for C there is an OSAL layer; Verum delivers a standard POSIX compliant version. Users make their own in a few days. Connects ASD code to underlying O/S.
ASD Execution Models

- **Multi-threaded Execution Model**
  - API calls executed under calling Client’s thread
  - Asynchronous event notifications from used components executed under component’s own internal thread
  - Multi-threaded if component implements asynchronous events
  - Can require as many as one thread per component instance

- **Single-threaded Execution Model**
  - Intended for components used within an application server framework or synchronous/resource-constrained environment
  - No internal component thread
  - Both API calls and event notifications are processed under application server’s thread
  - Components can be serialized and mobile (Java only)

- Formal verification is all about behavior so there has to be a definition of runtime behavior regarding threading, concurrency etc.
Business Model

- **SaaS – Software as a Service**
- **Cloud-based on hosted private cloud**
- **Client/Server Architecture**
  - Models created and edited on local PC
  - Code generation and model checking done on Server
- **Charging based on connection time**
- **Floating licenses**
- **Pay per use**

• This is how Verum attempted to sell ASD. No idea if it’s still their approach.
• Model editor very important because verification depends on well formed models being presented to it. So editor ensures ASD models are well formed and complete. Model checking then answers the correctness questions. Most well-formedness properties are hard or impossible to verify in CSP.
• Floating licenses: as much use as needed for limited number of concurrent users.
• Pay per Use: limited amount of use for unlimited number of concurrent users.
The Good, the Bad and the Ugly

- **Formal Verification has real repeatable benefits**
  - ASML: LOPW project saved ± 60 man years
  - FEI: Cost reduced by 75%
  - Ericsson: Costs reduced by 50%
  - PHC: Defects reduced by 90%, time by 66%

- **Formal Verification can be introduced to industry**
  - Verum spent ± €10M over 10 years breaking through into mainstream market

- **Resistance to change is massive**
  - Benefits insufficient to overcome resistance
  - Business case carries no weight with technical staff
  - Perceived as a threat
  - Users new to MDD & CBE found abstraction too hard

- FV has demonstrated real, repeatable and significant business value
  - 60 MY at ASML is about 7.2M Euros – ASML has confirmed this face to face to me
  - Ericsson comparison is between costs in Rijen NL and India, China.
  - Published by Ericsson in Dutch FD article
  - PHC based on independent report from TU/e. PHC elapsed time reduced by

- It is possible to introduce FM into industry if done right
- Resistance to change much greater than I ever imagined.
  - Even these business benefits did not persuade management to overcome opposition from programmers
  - Many programmers saw ASD as a threat. ASD means a newly graduated engineer without extensive programming knowledge but with good education in computer science and design could complete with a 10 year experience C+ + GURU and complete well.
  - Programmers new to MDD and CBE found modeling level of abstraction too difficult and anyway, didn’t want to stop programming
The Good, the Bad and the Ugly

- **MDD is disruptive**
  - Causes change management issues
  - Very divisive
- **Technology & product need continuous development**
  - Expressiveness, runtime execution models
  - More verifiable properties
  - Continuing development costs
- **Small Vendors lack credibility**
  - Adopting MDD is a strategic decision
  - Vendor lock-in is real
  - Market expects SW development tools are cheap

- Introducing any MDD platform (not just ASD) is disruptive
  - At PHC, for example, ASD caused major divisions and arguments between technical staff
- It's never finished
- Small Vendors lack credibility – this is a killer
  - Adopting an MDD like ASD means creating a strategic dependency on the supplier. Small companies with flakey finances are not credible
  - However hard we might try to circumvent this, Vendor lock-in is both real and unavoidable. After some time, the customer’s IP is locked up in ASD models. Switching costs are very high for the customer. Small companies with flakey finances are not credible suppliers in this situation.
  - Market price expectations set by Visual Studio etc.; cheap in other words. Investors know this.
Thank you

Questions?