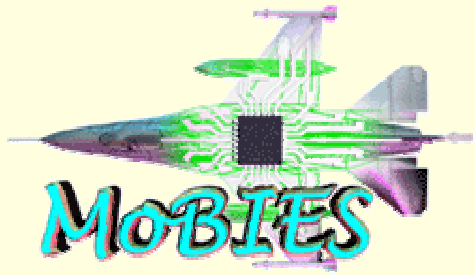
Keys to Success in Model-Based Design



Michael S Moore, Mark Brooks, Greg Willden
Southwest Research Institute
{msmoore,mbrooks,gwillden}@swri.org
Sandeep Neema
Institute for Software Integrated Systems
sandeep.k.neema@vanderbilt.edu

Work Sponsored By DARPA IXO
ARFL Rome, NY

Sponsor and Collaborators

- Thanks to sponsor:
  - DARPA IXO, AFRL, Rome, NY
- Thanks to collaborators:
  - Vanderbilt University ISIS
  - Teknowledge
  - BBN
  - Kestrel Institute
  - Carnegie Mellon University
  - UC Berkeley
  - Mathworks
  - Emeskay

Overview

- Challenges in computer-based systems
  - The Plethora of “Silver Bullets”
- What is *domain specificity*, and why is it important to success
- The DARPA MoBIES Software Defined Radio OEP
  - Tool-chain and process overview
  - Accomplishments and successes
  - Reasons for the successes
- Conclusions
  - MIC allows “domain specific” tools (Use accepted notations, formalisms, and semantics)
  - MIC tools are customizable for specific applications, platforms, and business models → and create success (better systems, cheaper)
  - Domain specificity creates buy-in (Bring the tool to the user, not the user to the tool)

Work Sponsored By DARPA IXO
ARFL Rome, NY

Challenges in Computer-Based Systems

- Complexity drives systems to be…
  - difficult to specify and understand
  - expensive to develop (integration and test)
  - expensive to maintain, evolve, and scale
  - difficult to design in a predictable and reproducible
- Agility of applications, technologies, and platforms drives…
  - design space explosion (so many options, so little time)
  - specialization
    - systems, hardware, software languages and design methods, platform development, testing, quality assurance, etc
    - … in addition to classic engineering and science disciplines
  - many notations / semantics
    - … but often no coherent over-arching notation or semantic

- Hero syndrome: few people really understand both the system and the implementation details… those people end up solving the integration problems on real programs – and "saving the day"

Work Sponsored By DARPA IXO
ARFL Rome, NY

The Plethora of "Silver Bullets"

- Many complex system software development approaches
  - Each has promising to rid the world of these problems…
- Languages (if we could all just agree on standard languages and document our code…)
  - Language, coding, and documentation standards
  - OO languages, 4GLs, scripting languages
- Processes (if designers would only better document, measure, and formalize what they do…)
  - SEI CMM / CMMI, ISO 9660, etc
- Runtime architectural approaches (if platforms could be abstracted and standardized…)
  - Platform abstraction / virtual machines
  - Middleware / layered design standards (e.g. POSIX, MPI, CORBA, etc)
  - Domain architectures (e.g. SCA, other proprietary product line architectures)
- Model-based design approaches (if we only had tools to model and generate software…)
  - UML design notation and code generation tools
  - MIC (domain specific modeling) and MDA (platform independent / platform specific modeling)

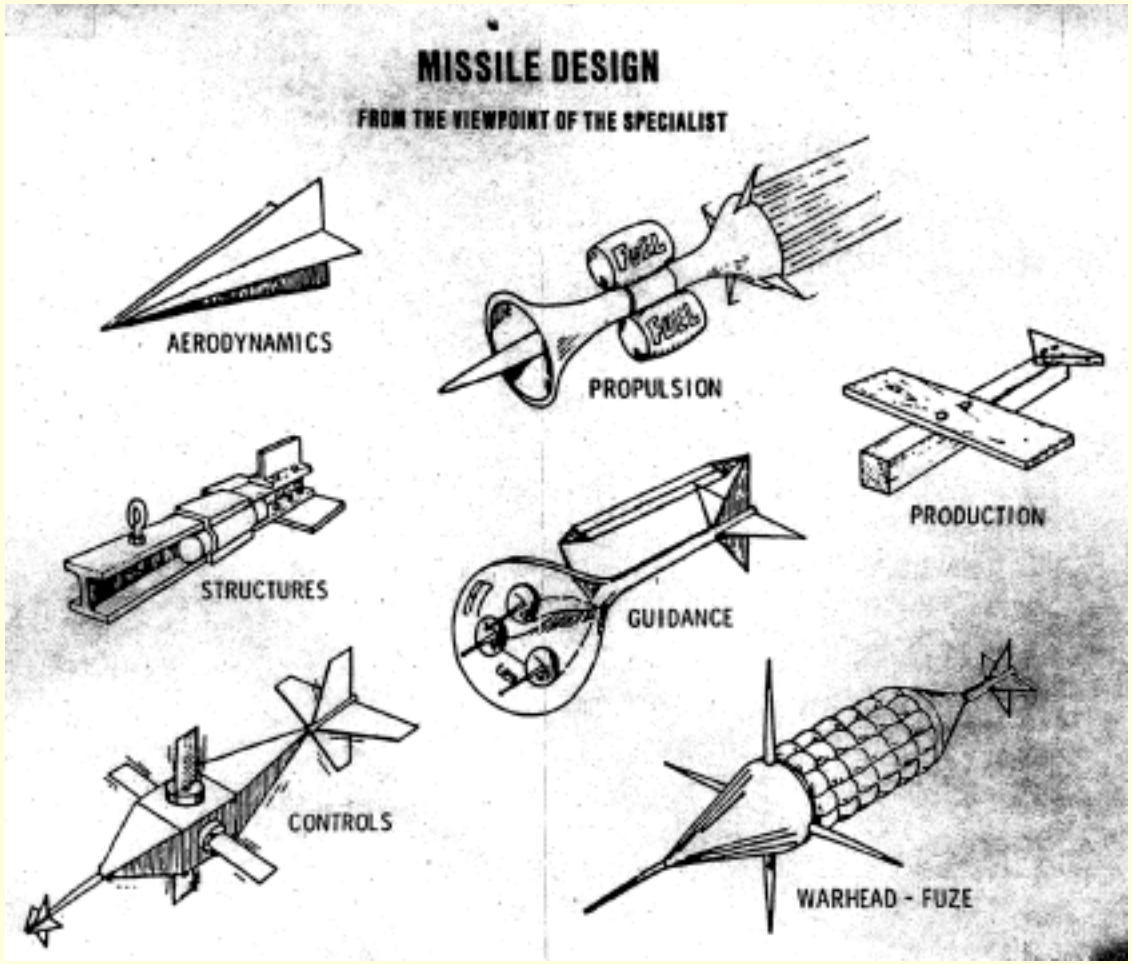
Each of these approaches could prove important to a potential solution to building better systems cheaper, but none is a "silver bullet"

Current Trends: Toward a Solution?

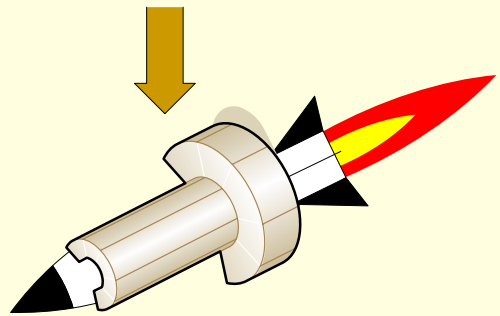
- Software design notations (UML 2.0)
- Middleware / domain architectures
  - OO middleware (CORBA) + Component models
  - Example: JTRS / SCA for Software Defined Radio
- Software-centric design methodologies…
  - … treat the software as the primary entity (software perspective)
  - … overlook domain knowledge (notations, semantics, and practices)
  - … subjugate the application/functional purpose (aka “business logic”)

Software design notations and domain architectures are important, and useful for software experts, but can alienate “domain experts” who design systems

Viewpoints / Engineering Disciplines



And, finally, a missile in the the view of the software expert…



Software

Work Sponsored By DARPA IXO
ARFL Rome, NY

Domain Specific Modeling

- Specify systems in terms of the accepted design notations
- Build domain-specific system generators

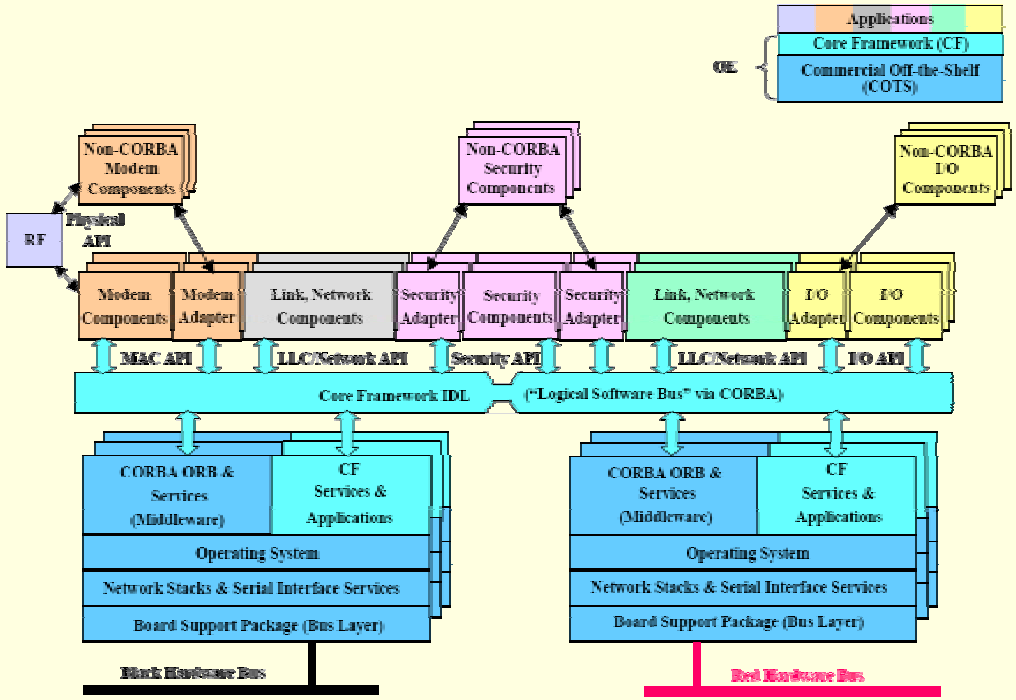
- Examples of domain specific modeling
  - Discrete manufacturing : processes, queues, control and data interfaces
  - Electric utilities : electrical elements (wires, switches, etc)
  - Controls : control flow diagrams, state transition diagrams, control laws
  - Signal processing : signal flow diagrams, difference equations, transfer functions
  - Software defined radio : signal processing, specialized for radio (antennas, PAs, RF interfaces, etc)
  - Signal classification : signal processing, specialized with classification rules, data interfaces, etc

- Modeling in UML using domain specific architectural patterns does not meet this definition

Work Sponsored By DARPA IXO
ARFL Rome, NY

Domain Specific Modeling

- Challenge problem
  - Find the radio in this drawing…
  - This is the definition of a DoD radio (JTRS SCA 2.2)
- If radio engineers cannot relate to this architecture, they will not buy in
- This architecture may represent a good software solution, but an interface recognizable by radio experts would help the transition



Work Sponsored By DARPA IXO
ARFL Rome, NY

SDR OEP Top-Level Goals

- Applying MoBIES MIC technologies to the Signal Exploitation domain
  - Signal Exploitation (SE) is a sub-domain of Software Radio
  - A Signal Analyzer (SA) is the part of a SE system that classifies signals
- Our goals are to Optimize the …
  - Development efficiency : time, cost, first time quality
    - Find bugs earlier in design cycle
    - Generate code, validate functional performance
  - Functional performance : misses, hits, false alarms
    - Automatically discover 'best' values of key parameters
  - Computational performance : latency, throughput
    - Generate high performance code to utilize platform
    - Automatically map computations to resources

… of Signal Analysis systems

Work Sponsored By DARPA IXO
ARFL Rome, NY

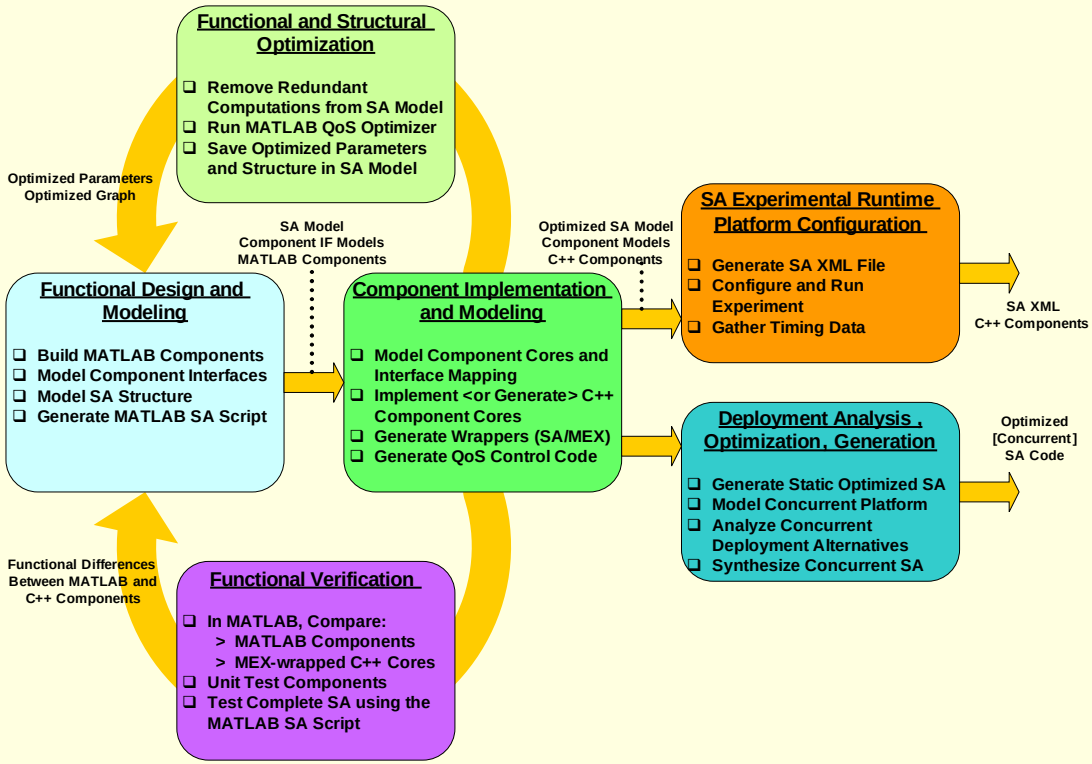
Signal Analyzer Development

- The incumbent design notation is MATLAB
  - Does not lend itself to deployment-related analysis or optimization
- Problems are identified late in the development cycle
  - e.g. functional differences between design and implementation
- Functional parameters is not chosen systematically (rules-of-thumb)
- Achieving the required performance is difficult
  - One-of development, deployment onto platform (using black magic)
- Design cycles are too long (conditions change – e.g. new waveforms)
  - Systems should be end-user configurable (non software experts)
- Systems must adapt to environment, state, and specific circumstances

Work Sponsored By DARPA IXO
ARFL Rome, NY

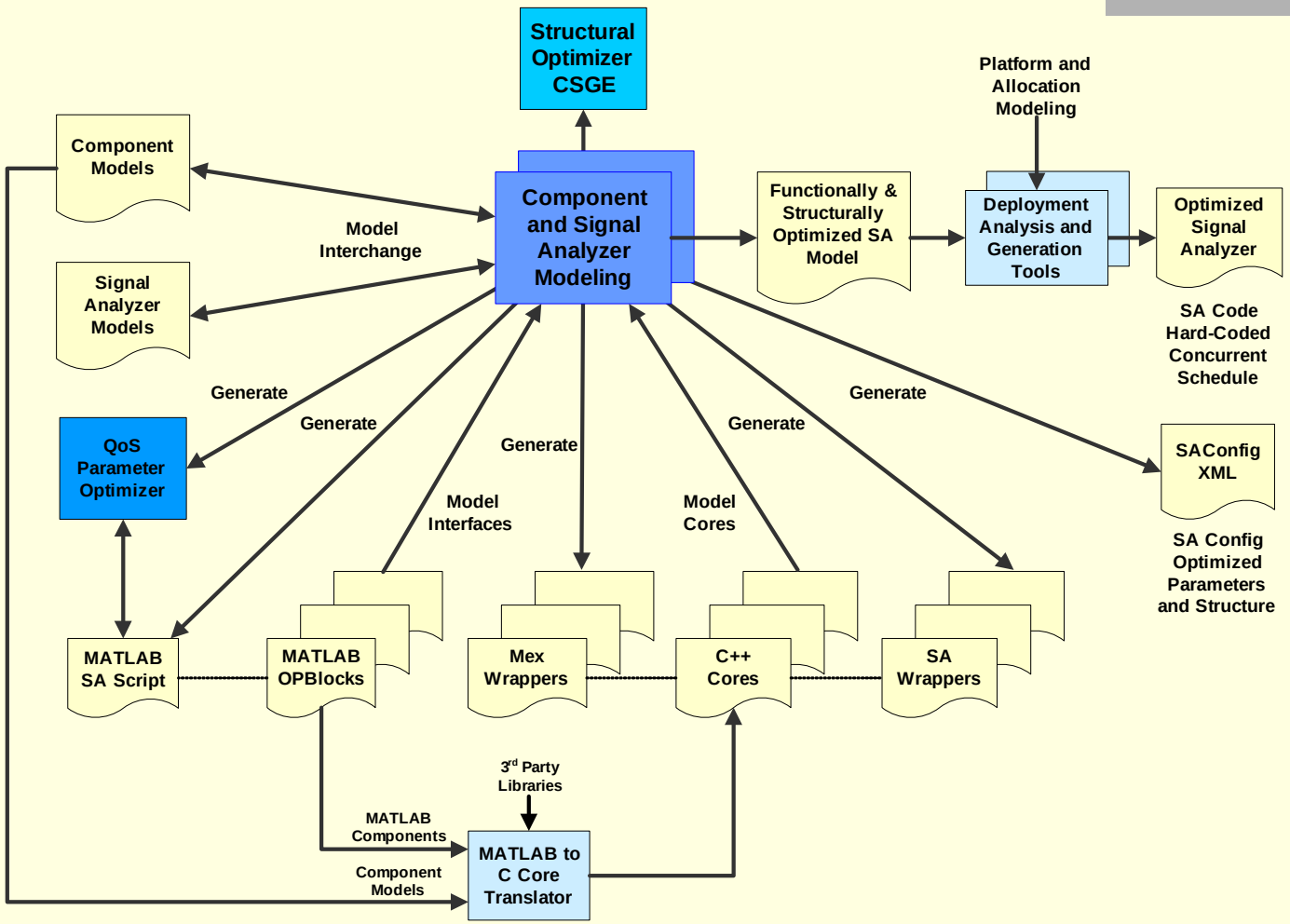
SA Development Process

- SA specific development process
  - Functional design in MATLAB
  - Component code generation
    - Cores / wrappers model
  - Classification engine code gen
  - Functional verification (before integration)
  - Compose large systems, optimize, and deploy onto parallel hardware seamlessly
- Requires customized tools aware of SA application domain
  - Classification rules
  - Functional performance
  - Structural optimization
  - Underlying platform requirements



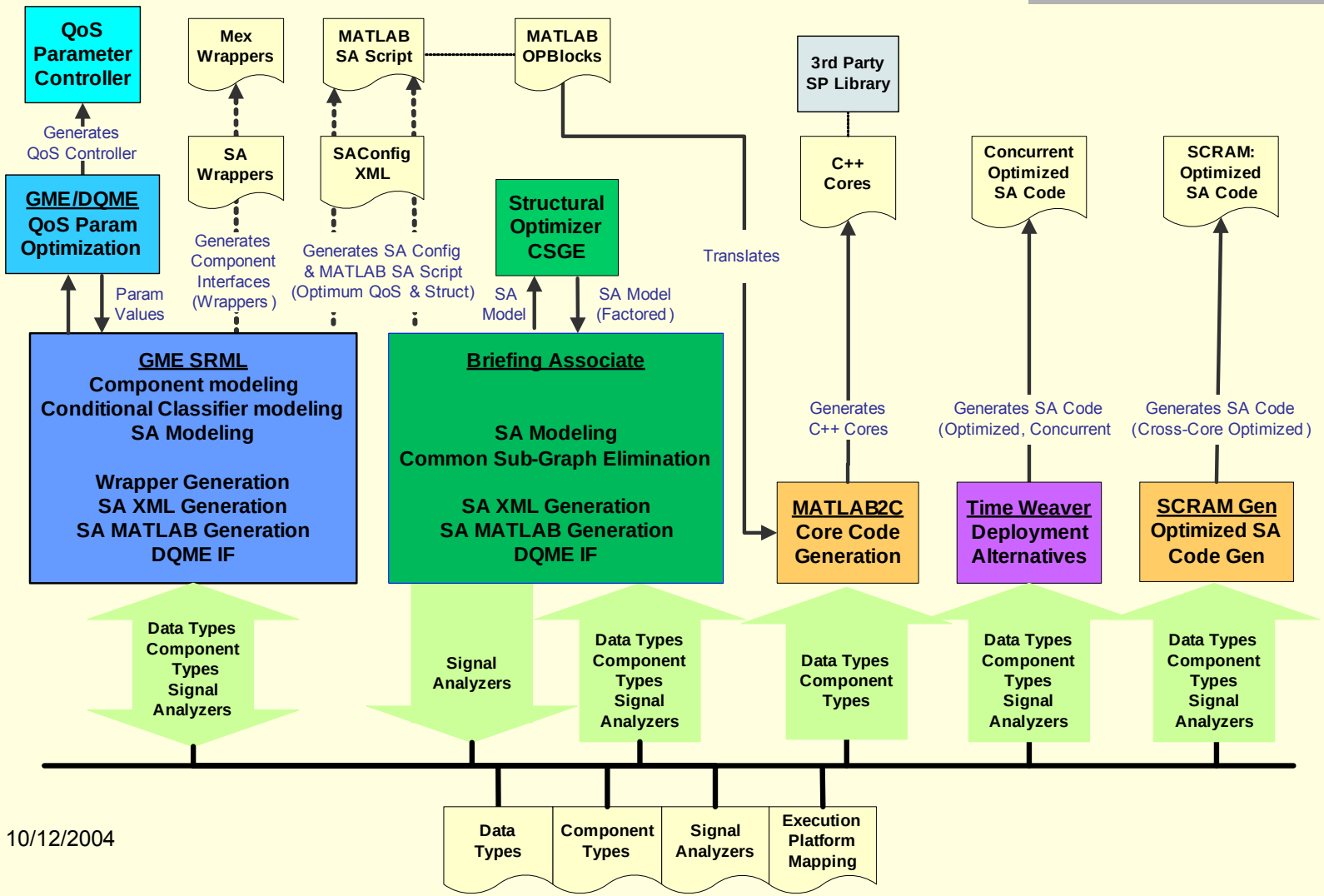
Work Sponsored By DARPA IXO
ARFL Rome, NY

SA Tool-Chain Concept



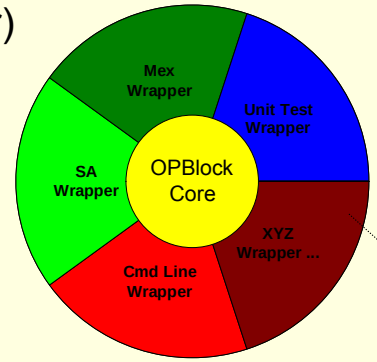
Work Sponsored By DARPA IXO
ARFL Rome, NY

Integrated SA Tool-Chain

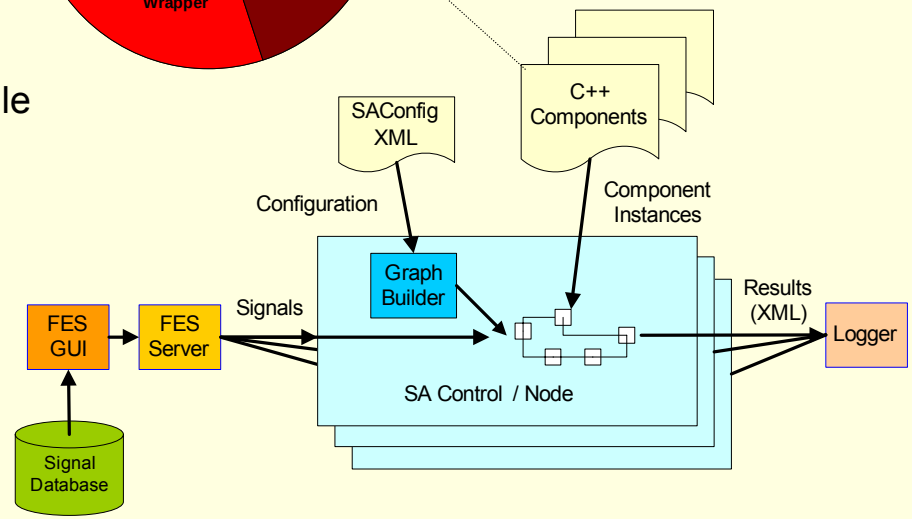


SwRI Signal Analyzer Testbed Platform

- Execution infrastructure
  - Signal Database
  - Front-End Simulator (FES / FES Server)
- SA Control
  - Receives commands
  - Configures SAs
  - Assigns signals to SAs
- Graph Builder
  - Creates the SA from the SAConfig XML file at configuration time
- Virtual Machine style approach
  - In-field configurable without a compiler
  - Not optimal performance



OPBlocks (Components) Functional and platform-specific interface code separated to support alternative deployment



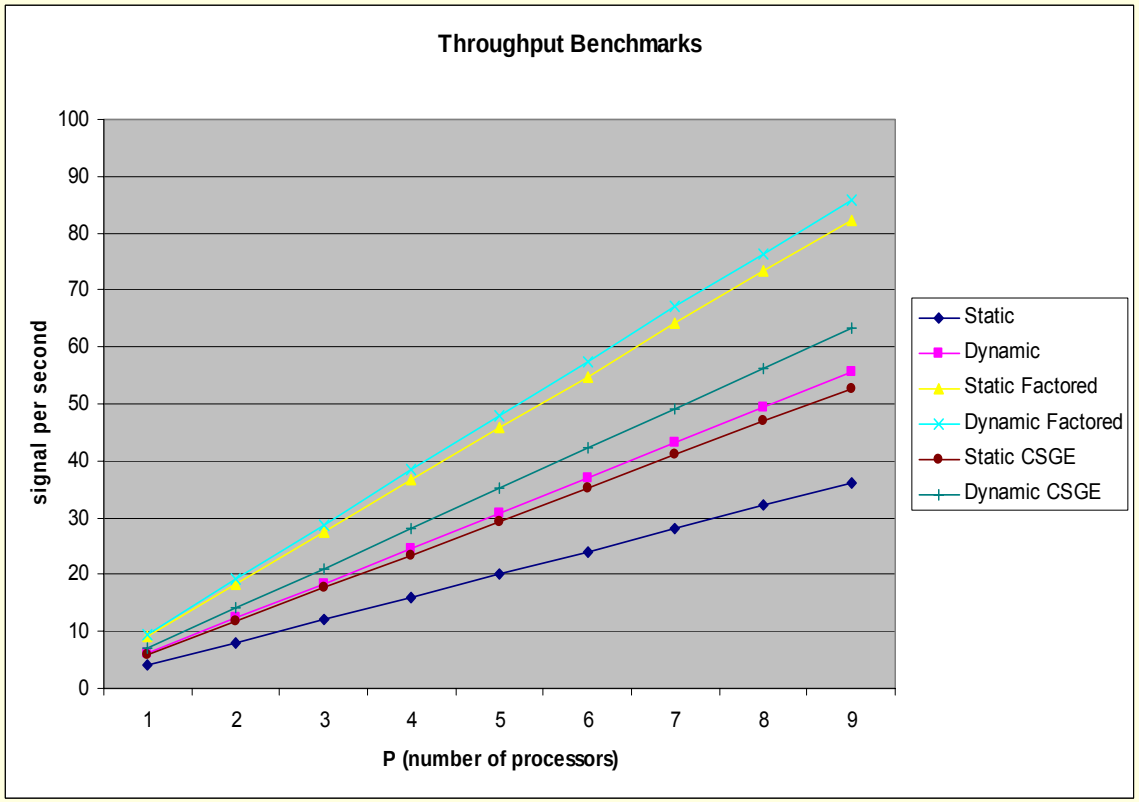
Work Sponsored By DARPA IXO
ARFL Rome, NY

SwRI Signal Analyzer Testbed

- Host node
  - 3 GHz Pentium 4, 1 MByte cache
  - Debian Linux (Knoppix install)
  - File system / NFS exported
- 9 compute nodes
  - 2.8 GHz Xeon, 1 GByte Ram
  - Debian Linux (Knoppix CD boot)
  - Diskless / mount host file system
- Network
  - 1 GBit IP switch (10 ports)
- Software / runtime architecture
  - SA data interface runs on host
  - SA Controls run on compute nodes
  - CORBA middleware
    - ACE/TAO 5.31/1.31

Work Sponsored By DARPA IXO
ARFL Rome, NY

Data Parallel Timing Study



- 86 signals/sec! (sustained)
- 21.3 speedup
  - vs P=1 benchmark
  - Hand-coded version not benchmarked on test-bed
- Very low overhead
  - > 95% of time in cores
- Single Processor performance is probably very close to hand-coded version
- Estimated bursting capacity well past 300 signals/sec!

Work Sponsored By DARPA IXO
ARFL Rome, NY

Software Radio OEP Successes

- Results
  - Integrated MIC SigInt development tool-chain
  - Quantitative measures of success in
    - Development efficiency ↓ cost, ↓ time
    - Functional performance ↓ cost, ↑ capability
    - Computational performance ↓ cost, ↑ capability
- Development efficiency
  - Huge improvement of SA development (%75-%80 of code generated)
- Functional performance
  - Significant performance improvement (difficult to quantify)
- Computational performance
  - 21x throughput improvement
  - 4.6x latency improvement (projected)
- Domain experts buy into it (they helped define the interface)

Work Sponsored By DARPA IXO
ARFL Rome, NY

Reasons For The Success

- Platform specific component info → component code generation
- Classification rule modeling → classification code generation
- Custom graph semantics → structural optimization
- Semantic mapping of graph to platform → system generation
- Domain experts defining the language → tool acceptance

Domain specificity of the models and tools was the key enabler for the success

Conclusions

- Software architecture itself is not the silver bullet
  - (nor are process, or case tools)
- MIC allows use of accepted domain notations, formalisms, and semantics (MDA should also)
- Domain knowledge is not “Business Logic” – but the entire reason for the system’s being
- Domain specificity promotes:
  - Customization of generators for application space
  - Reduction of development and life-cycle costs
  - Acceptance by domain experts

Work Sponsored By DARPA IXO
ARFL Rome, NY

Backup Slides

Work Sponsored By DARPA IXO
ARFL Rome, NY

Signal Exploitation Domain and Signal Analyzers

- Signal Exploitation Domain (sub-domain of Software Radio)
  - Use all available RF information to recover a transmitted signal in the presence of noise and other degraded conditions
- A Signal Analyzer (SA) is the part of a SE system that classifies signals
  - Identifies the type of waveform, band, and parameters
  - Complex combinations of common and custom 1-D DSP algorithms
- Relative size and nature of SA systems
  - Medium scale concurrent hardware with highly specialized analog front-ends
  - Commercial Real-Time Operating Systems, CORBA middleware
  - 1000s of SW components, medium granularity (e.g. Power Spectral Density)
- Environment
  - Noise : Gaussian noise, multi-path, impulse noise, co-channel interference
  - Persistent computations : large number of signals over time, bursts of activity
  - Soft real-time : throughput is paramount, latency less important
  - Overloaded : always more signals to analyze than there are available resources
  - Adaptive : behavior during overload is driven environment

Work Sponsored By DARPA IXO
ARFL Rome, NY

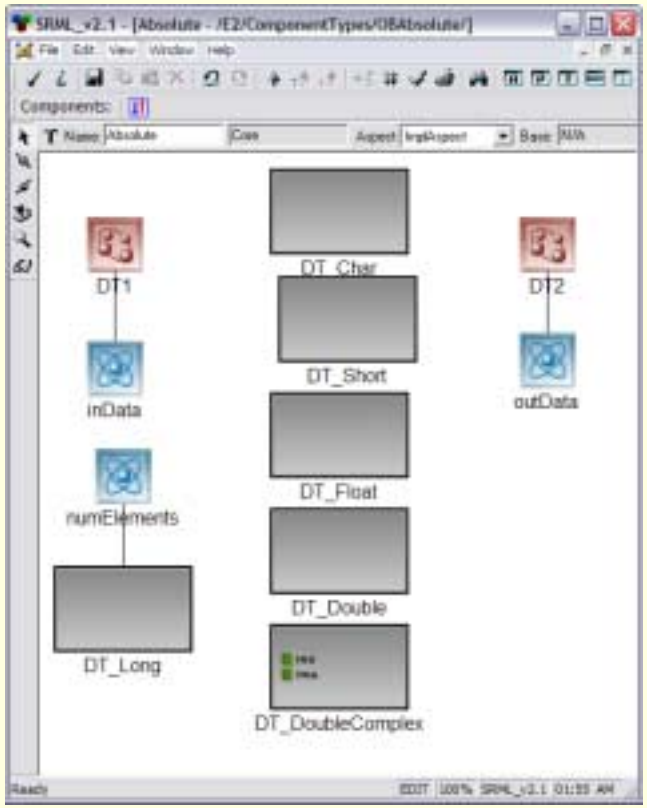
Applying MIC to Signal Analyzer Development

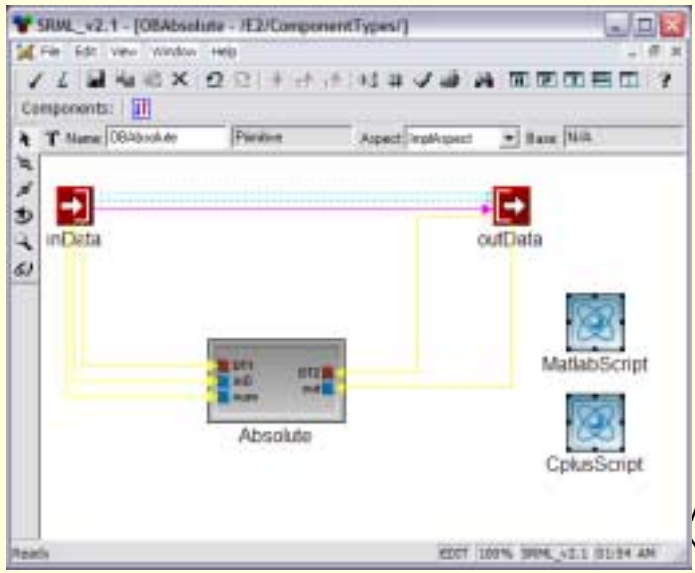
- Provide a domain-specific modeling language
- Provide tools for functional verification of components
- Utilize QoS adaptation technology to optimize functional parameters
- Automate the deployment and runtime optimization
  - Analysis tools for deployment alternatives (concurrency, mapping)
  - Optimized code generators to generate concurrent SA code
- Reduce design cycles by generating code from models
  - Generate the composition code (‘glue code’)
  - Generate component code (interfaces, and partial functional cores)
- Model the adaptive SA behavior, and relationships to the environment
  - Generate the code to implement the adaptive behavior

Work Sponsored By DARPA IXO
ARFL Rome, NY

Component Modeling

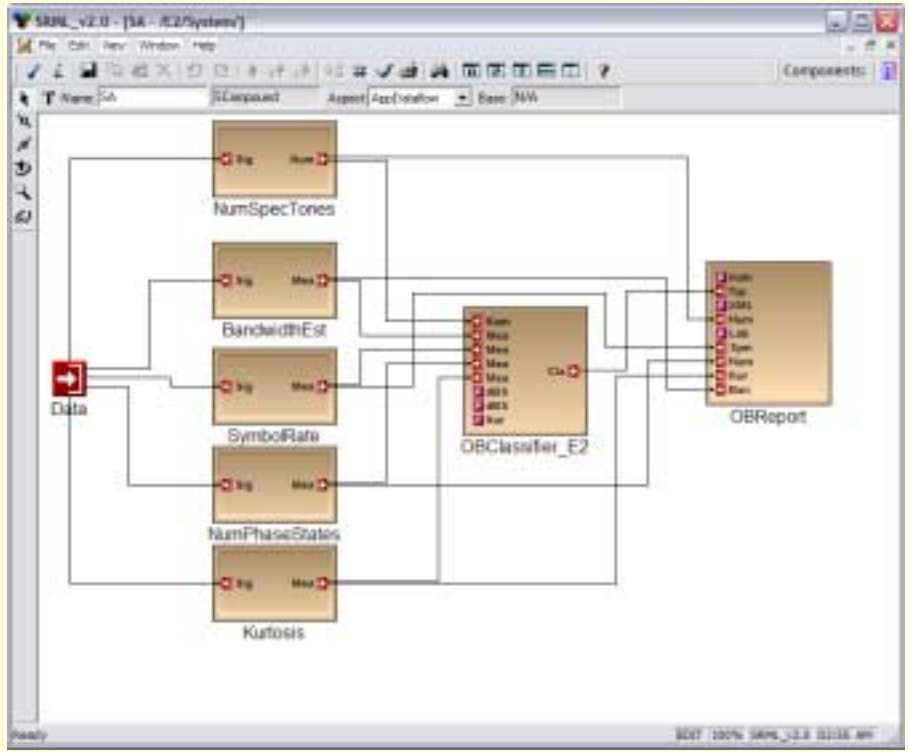
- Component type models
  - C++ Core interface
  - Component platform interface
  - Mapping between core and platform
  - Data type and data size semantics
    - Rules by which component polymorphism can be resolved





Signal Analyzer Modeling

- Signal Analyzer Composition
  - Hierarchical dataflow
- Features
  - Leaves are instances of component types
- Classifiers
  - Logical relationships between features and outcomes
- Semantics (MoC) applied
  - Static (SDF) semantic applied to features
  - Dynamic demand-driven scheduling semantic applied at top level SA



Work Sponsored By DARPA IXO
ARFL Rome, NY