MAUDE:

A Wide-Spectrum Formal Language

for Secure Active Networks

José Meseguer
SRI

Carolyn Talcott
Stanford University

Active Nets Briefing

March 18, 1999

Problem Statement

The present situation in developing new software and communication technology:

- Nobody knows ahead of time what the new technology is (what is an AN?)

- Current software technology is very weak at modeling and analysis: typically using informal specifications and pseudo-code, sometimes simulations, and then implementation. This "build first, learn later" paradigm is:
  - too expensive
  - too late

- System integration can bee very hard, particularly when integrating subsystems developed by different teams (Tower of Babel)

The Situation for Active Nets

All of these problems arise in Active Nets because:

- AN is a new technology
- there are many teams/projects to be integrated
- conventional system development methods (e.g. pseudo-code $\longrightarrow$ implementation) are inadequate and do not scale up
- there are no methods to clearly communicate the designs and requirements of different components
- there are important high-confidence requirements

Paradigm Shift

A promising approach to alleviating these problems is a paradigm shift in system development that uses formal modeling systematically and more aggressively from the earliest design phases, spanning the horizontal and vertical dimensions of system interoperation.

What can Maude do for Active Nets?

Maude can be used to build executable formal models of Active Nets artifacts/subsystems very quickly, at different levels of abstraction, and amenable to a wide range of formal analyses.

In this way, Maude can support:

- horizontal and vertical interoperation and evolution
- communication of designs and architectures across teams
- formal analysis and high assurance for critical subsystems
- rapid prototyping and code generation

Encouraging Experience

We have been collaborating with other teams in the Active Nets program and other researchers to realized this paradigm shift in practice, focusing on several areas

- Reliable Broadcast Protocol Design (UCSC (team 4))
- Active Net algorithms (UPenn (team 6))
- Cryptographic Protocol Analysis (team 6)
- Semantics of CAPSL (J. Millen and G. Denker (SRI))
- Composable Communication Services (team 4)
- Interoperation of Heterogeneous Software Architectures (EDCS)

Expanding the Horizons

Other areas seem also ripe for applying these methods. For example:

- Middleware

- Active Net architectures and standards

What Matters

Mathematical modeling, including symbolic simulation, of the different aspects and stages of system development that support formal analysis.

Correctness is only one part of the analysis, and is only meaningful after adequate models have been built. Full proofs are only needed for critical aspects.

Maude supports a wide spectrum of analysis capabilities:

- Executable specification and testing of specifications
- Model checking and search
- Symbolic simulation
- Theorem proving

Vertical Integration

Maude supports specification spanning a wide variety of levels of abstraction:

- end-to-end requirements
- architecture
- formal specification
- prototypes
- system implementation

Horizontal Integration

Maude supports heterogeneous composition and interoperation:

- components and their composition
- protocols and composable services
- middleware
- monitoring and management
- dynamic evolution and adaptation

How is this done?

Maude is based on a metalogic (rewriting logic) with great breadth and flexibility to model and relate different system aspects. Maude specifications are efficiently executable. Furthermore, these specifications may be partial or incomplete.

Key Maude feature include:

- Very high performance
- Intrinsically distributed and object-oriented
- Reflection
- Extensible module calculus

Maude's Performance

Maude combines the flexibility and traceability of an interpreter with the hight performance of advanced semi-compilation techniques:

- Fastest equational rewriting engine in the world (800k rewrites per second and 120k AC rewrites per second on Pentium II for realistic applications) beating the *compilers* of its closest competitors (ELAN and CafeOBJ)

- 50–100,000 times speedup over previous generation specification languages like OBJ3

- Easy to get another order-of-magnitude speedup through compilation (8M rewrites per second) making it a very attractive implementation language.

Intrinsically Distributed and Object-Oriented

Maude has great flexibility for specifying and executing concurrent and distributed models and architectures:

- supports concurrent and distributed objects in a natural way

- systems can be synchronous, asynchronous, or hybrids, and can communicate through point-to-point, multicast, broadcast, or combinations

- supports structuring of distributed systems through domains, firewalls, dynamic distributed services, etc.

- supports reflective capabilities such a monitoring, management, dynamic adaptation, and composable services

Maude's Reflection

Maude's rewriting logic is *reflective*. That is, rewriting logic can be formalized in itself. Maude efficiently supports this reflection and can manipulate the formalizations themselves and their relations:

- specifications and models are first-class objects for transformation and composition

- formalizations can be related by mappings:

  – partial evaluation and abstraction

  – mappings spanning the vertical integration

  – mappings between formalisms

- Maude can be used as a meta-tool to generate other tools such as formal analysis tools and domain specific languages

Extensible Module Calculus

By reflection, specifications are first-class objects that can be computed with. They can be *transformed* and *composed* through user-definable module operations. This extensible module calculus supports powerful horizontal system structuring capabilities.

Such an extensible module calculus (Full Maude) has been defined in Maude using reflection. It supports:

- parameterized modules
- views
- module expressions
- object-oriented modules

Progress Report

- Reliable Broadcast Protocol
- Formal Analysis of PLAN Algorithms
- Cryptographic Protocol Analysis
- Common Authentication Protocol Specification Language (CAPSL)
- Composable Distributed Systems
- Semantic Interoperation of Software Architectures
- Possible Applications within Team 4

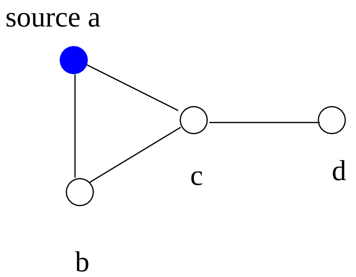
Reliable Broadcast Protocol

Collaboration with
J.J. Garcia-Luna-Aceves, J. Raju, B. Smith (UCSC)
G. Denker, P. Csaba Ölveczky (SRI)

- early design pseudocode
- challenging new protocol: dynamic & decentralized

static case:

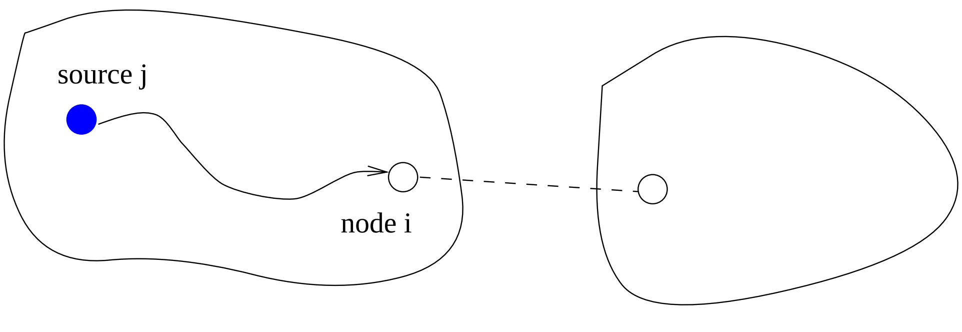
no change in the network due to failure or mobility
given net topology
connected component: all direct and indirect neighbors
goals: (1) nodes in the connected component of the source receive information within finite time
(2) the source node is notified about it



Reliable Broadcast Protocol for Dynamic Topologies

- extension: multiple, concurrent messages and source nodes
- extension: links may fail, new links come up
- goals:

  for all nodes and latest messages the following holds
  (1) nodes in the connected component of the source receive latest message of source within finite time
  (2) the source node is notified about it



RBP – Value Added

Original pseudocode informal specification

- had mistakes and inconsitencies,
- did not consider some cases,
- did not make explicit some key assumptions,
- was not executable.

Maude specification

- found deadlocks and unexpected behaviors,
- made explicit all key assumptions,
- is complete and executable,
- allows systematic exploration of all behaviors, and
- is amenable to formal verification.

Formal Analysis of PLAN Algorithms

Collaboration with C. Gunter and Y. Wang (UPenn)

- Specified a PLAN algorithm that sent active packets on a network to find optimal routes.

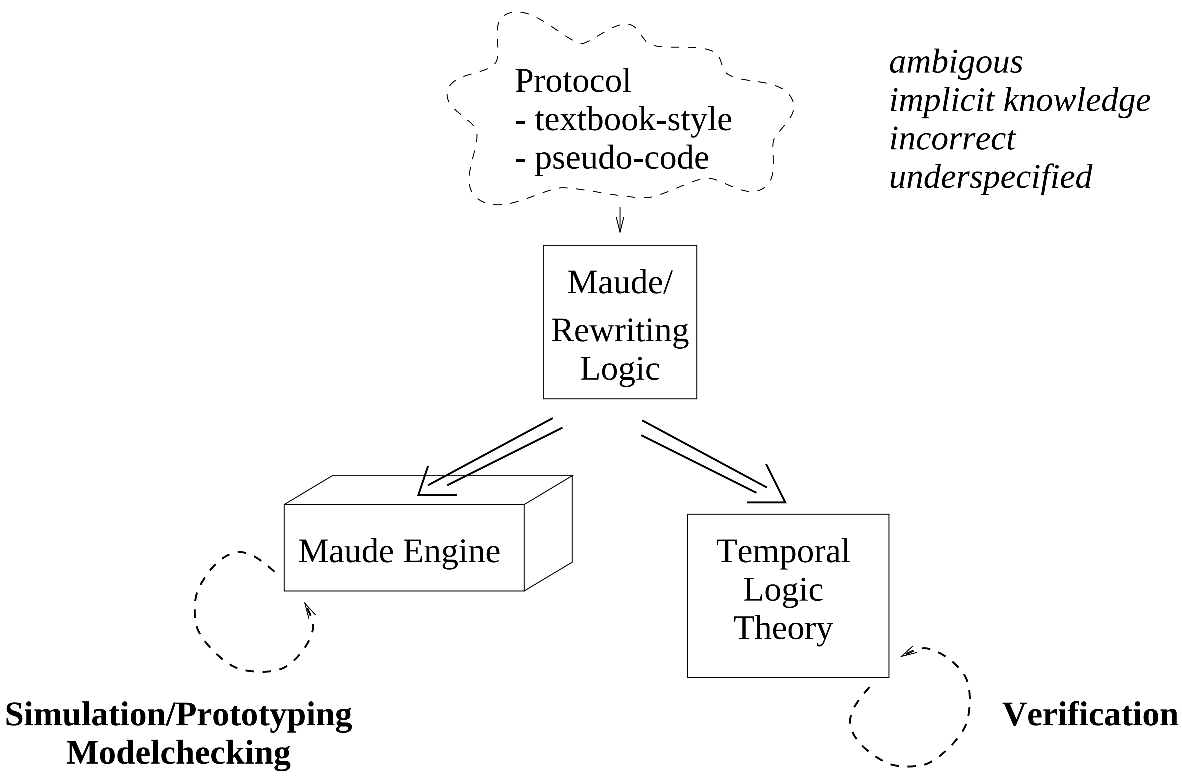
- Validated the correctness of the specification using model checking techniques and parallelization.

- Opens up possibilies of formal semantics for PLAN and formal verification of PLAN algorithms.

Cryptographic Protocol Analysis

Collaboration with G. Denker, J. Millen (SRI)

- formal, executable specification of cryptographic protocols
- model checking approach to find bugs
- basis for high-level verification of formal properties



Cryptographic Protocol Case Studies

Specification, execution, and analysis of:

Needham-Schroeder Public-Key Protocol

specification time: weeks (first cryptographic protocol/strategies)

breadth-first-search strategy (depth 11) delivers attack ([Low96])

(300Mhz Linux 4.2. PC, 96 minutes, 28 M. rewrites

Ping-Pong Protocol (Dolev-Yao)

specification time: 10.5 hours (optimized intruder, etc.)

breadth-first-search strategy (depth 7) delivers attack

(266Mhz Linux 4.2. PC, 2.54 minutes, 1.57 M. rewrites)

Common Authentication Protocol Specification Language (CAPSL)

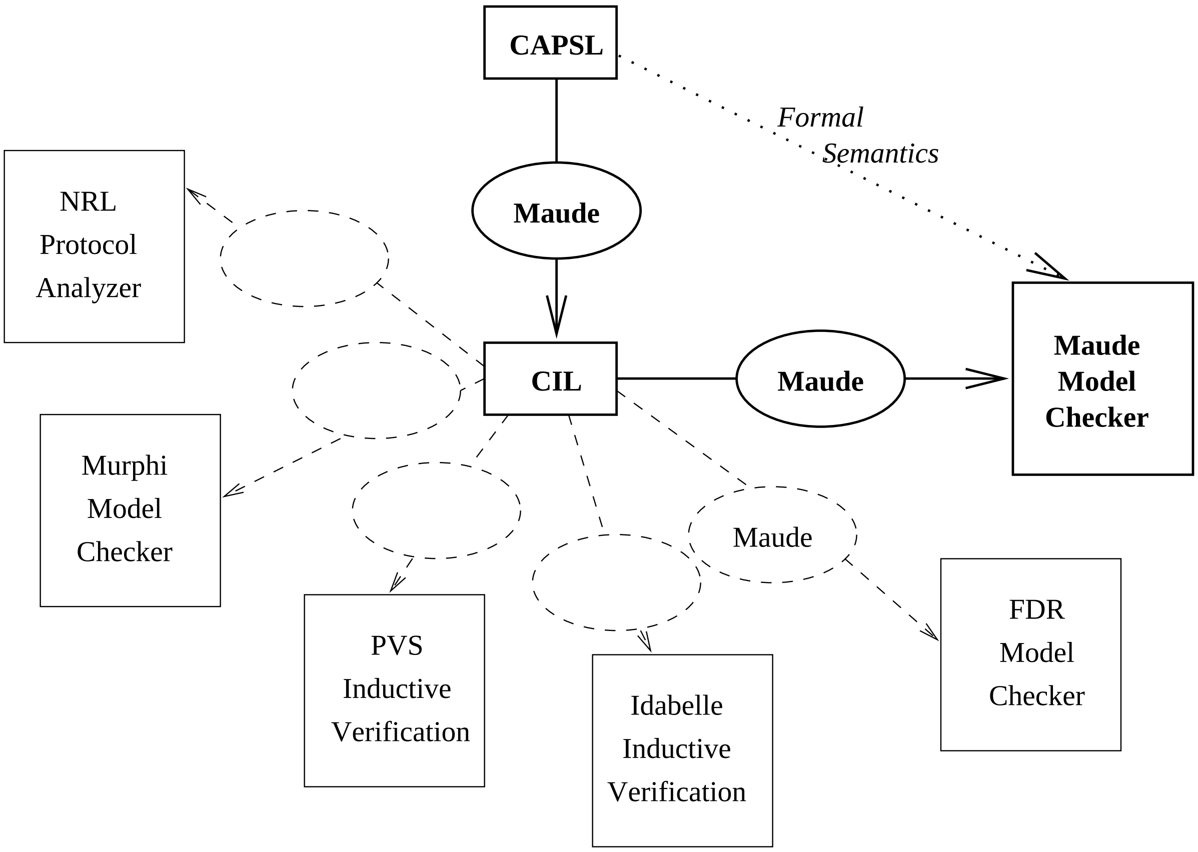
G. Denker and J. Millen (SRI)

- single common protocol specification language that can be used as the input format for many formal protocol analysis techniques
- CAPSL environment: language and supporting tools
- CAPSL uses CIL as an intermediate language for use by various analysis tools

Maude has been used to:

- give formal semantics to CAPSL
- specify translation from CAPSL to CIL, and from CIL to executable Maude specification
- provide interfaces to other tools

Using Maude to connect CAPSL with other Tools



Composable Distributed Services

Modular and **composable** approach to **services and service guarantees** in distributed computing (e.g., *security, fault-tolerance, boosting performance*)

- dynamic and selective installation at runtime

- reusability, maintenance

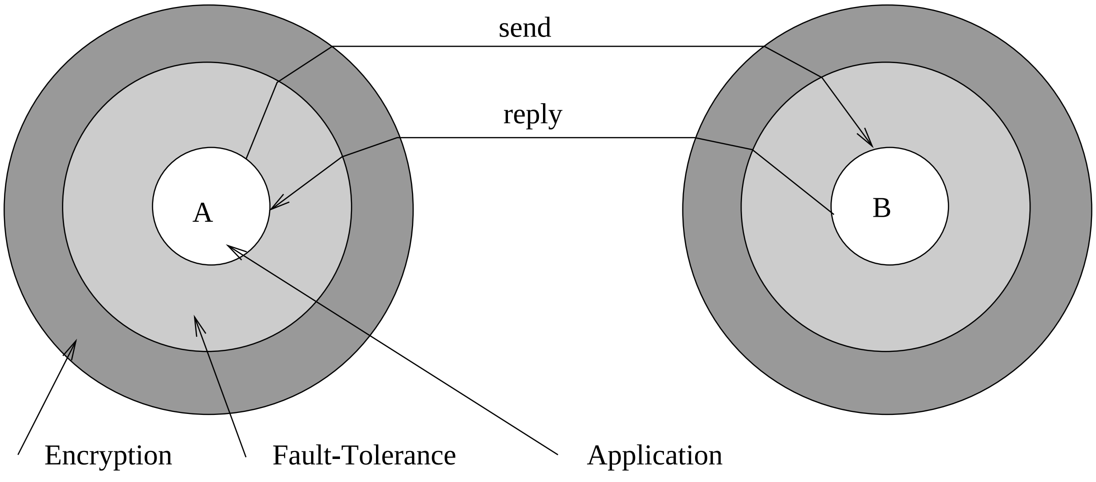
- systematic service composition (reflective, vertical) based on semantics

- compositional approach to formal service guarantees

Onion Skin Model of Actor Reflection

Each actor has a meta-actor that defines the semantics of its primitive actions.

Example: Composable communication services:
Secure, reliable communication service



Results:

- Formal, executable semantics of the onion skin model of service compositions
- Formalization of service guarantees in a composable way
- Detailed case study fully specified (secure and fault-tolerant communication service)

Future Work:

- Specification of service interfaces
- Formal verification techniques
- Other composition operations
- More case studies

Semantic Interoperation of Heterogeneous Architectures

Architectural Description Languages (ADLs) are useful to document and analyze high-level system designs. However, they often lack a formal semantics, and it is very difficult to interoperate them.

Within the EDCS program Maude has demonstrated how:

- ADLs can be given a formal executable semantics in Maude
- Heterogeneous architectures expressed with different ADLs can be interoperated, executed, and formally analyzed
- Maude can be used to build execution and formal analysis tools for ADLs (Wright/CSP)

Possible Applications within Team 4

We are currently exploring several possible applications of Maude within Team 4:

- Executable specification and formal analysis of Reliable Broadcast Protocols

- Formal analysis and validation of some security properties in the Seraphim system

- Study of the relationship between composable services in CANES and the formal specification of composable services in Maude

- Possible use of Maude to specify the architectural integration between subsystems.

Future Directions

The results obtained so far are encouraging, and suggest that Maude can be frutifully applied to Active Nets in a variety of ways.

More work, mostly in collaboration with other AN research groups, is needed to capitalize on the current results, and to show by example the new design, specification, and modeling paradigm that Maude supports.

Future Directions (II)

Specifically, we would like to:

- Continue the fruitful collaborations already discussed, making more advances in all of them

- Explore new applications of Maude to AN integration and interoperation, including AN Architectures, and Middleware

- Investigate Mobile Maude, a mobile extension of Maude ideally suited for mobile computation over networks.

- Have new releases of Maude with improvements and additional features towards Maude 2.0 in February 2,000

AN Architectures

Maude could be used to specify important aspects of the software architecture of different AN subsystems and their integration including:

- formalization of the architecture of specific integration efforts
- interoperation of different interaction models
- prediction and analysis of emergent properties

Middleware

Maude could be used to give precise high-level formal specifications of AN middleware, including:

- specification of services and their compositionality properties

- specification of interfaces

- specification of the middleware's distributed interaction models

- symbolic simulation and formal analysis

Mobile Maude

At present it is unfeasible to guarantee the correctness of critical high-level properties of mobile programs written in conventional mobile languages such as Java.

Having a *declarative* mobile language with a logical semantics and allowing very high-level, yet efficient, programming of mobile applications seems a promising research direction.

Maude's declarative character and efficient execution, together with its intrinsic concurrency, object-orientation, and reflection make it a promising candidate for a next-generation mobile language supporting high levels of assurance through formal mehtods.