MICHIGAN STATE UNIVERSITY

Modeling and Analyzing Dynamically Adaptive Software

Betty H.C. Cheng

Software Engineering and Network Systems Laboratory
Department of Computer Science and Engineering
Michigan State University

web: http://www.cse.msu.edu/SENS
email: chengb@cse.msu.edu

Acknowledgments

- Software Engineering and Network Systems Lab
  - Faculty and students
  - RAPIDware project team
  - Ji Zhang
  - Philip McKinley
- Daniel Berry

•This work was supported in part by the U.S. Department of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744, and in part by National Science Foundation grants CCR-9901017, EIA-0000433, EIA-0130724, CCF-0541131, and CNS-0551622, and ITR-0313142, and by Siemens Cooperate Research, and a Michigan State University Quality Fund Concept Grant.

High Assurance Computing Systems…

Designed to tolerate failures,
even direct attack,
in order to continue operation and
preserve system integrity

MICHIGAN STATE UNIVERSITY

Two Ongoing Revolutions

- Pervasive (or ubiquitous) computing
  - Dissolves boundaries for how when and where humans and computers interact
- Autonomic computing
  - Focuses on developing systems that can manage and protect themselves with only high-level human guidance

Pervasive Computing

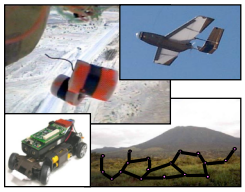
MICHIGAN STATE UNIVERSITY

- Driving Factors
  - Convergence of advanced electronic technologies (wireless, handheld, sensors, etc) and the Internet.
  - Promises anywhere, anytime access to data and computing.
- Need for assurance
  - Heterogeneity of hardware, network, software.
  - Dynamics of the environmental conditions, especially at the wireless edge of the Internet
  - Limited resources (such as battery lifetime).

Handheld/Wearable Computing

Military Applications

Sensor Networks



Autonomic Computing

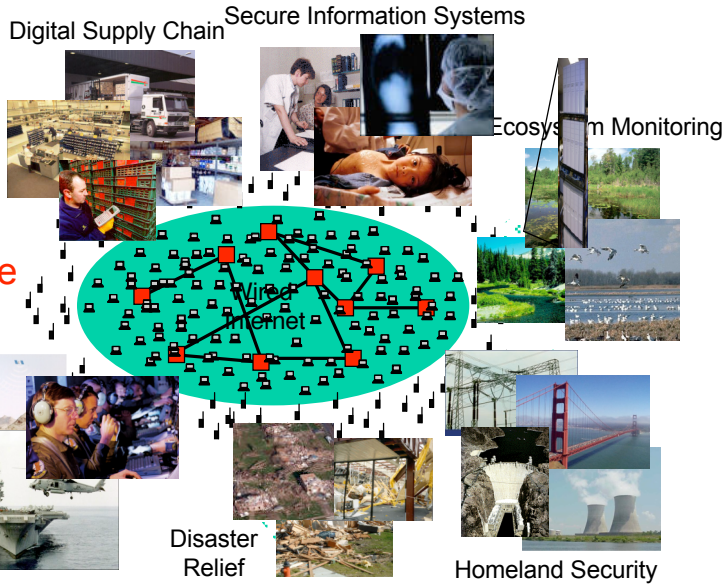
MICHIGAN STATE UNIVERSITY

- Promises self-managed and long-running systems that require only limited human guidance.
- Key driving force: Needed to manage and protect critical infrastructure: financial networks, transportation systems, and water and power systems.
- Systems must continue to ***operate correctly*** during exceptional situations, upgrades, and evolution
- Need for assurance
  - hardware component failures
  - network outages
  - software faults
  - security attacks
- Prediction: Autonomic will be essential to every future computing system.

Physical Cyberinfrastructure

MICHIGAN STATE UNIVERSITY

Remote
Safety-critical
Real-time
All the above
Privacy
Adaptable



The Software Crisis

MICHIGAN STATE UNIVERSITY

- Our increasing reliance on computing technology often fails to recognize that software is:
  - Brittle
  - Insecure
  - Continuously evolving
- Occasional wake-up calls…
  - Telephone service failure in 1990
  - Cell phone service on 9/11
  - Power outage of 2003
  - Breaches of credit card company databases…
  - Etc.

The Evolving Cyberinfrastructure

MICHIGAN STATE UNIVERSITY

- First Generation (proof-of-concept)
  - Driven by applications in science and engineering
  - Advanced sensor hardware
  - Efficient systems software, network protocol
  - Understanding issues, brittle prototypes
- Second Generation (sustainable infrastructure)
  - Driven by applications in science and engineering
  - Advanced sensor hardware
  - Efficient systems software, network protocols
  - High-assurance (generated) adaptable software
  - Intelligent systems that can ***learn*** how to respond and adapt
  - Designing and building robust, self-healing (autonomic) systems for deployment

Related Work

MICHIGAN STATE UNIVERSITY

- Where adaptations are applied
  - Programming languages [Adve01,Kasten02,Redmond02]
  - Adaptive frameworks [Fickas97,Sousa00]
  - Adaptive middleware [Blair98,Kon00]
  - Extensible operating systems [Bershad94, Appavoo03]
- Aspects in adaptation
  - Adaptation enabling techniques
  - Condition-monitoring techniques
  - Decision-making techniques
  - **Assurance techniques**

Related Work

MICHIGAN STATE UNIVERSITY

- Behavioral modeling approaches (process algebra)
  - CSP in Dynamic Wright [Allen08]
  - π-calculus in LEDA [Canal99]
  - FSP in Darwin [Kramer98]
- Architectures for dynamic systems
  - Darwin [Kramer98]
  - C2 Style [Taylor95]
  - Dynamic Wright [Allen98]
  - LEDA [Canal99]

Related Work

MICHIGAN STATE UNIVERSITY

- Safe adaptation protocols
  - Hot swapping in K42 [Appavoo03]
  - Graceful adaptation process in Cactus [Chen01]
  - Distributed safe adaptation [Kulkarni03]
  - Safe adaptation protocol [WADS04,ADS05]
- None of the above provides a software (or model-driven) development process that crosscuts requirements, design, and implementation to provide assurance in adaptive software

Model-Based Development of Adaptive Software

MICHIGAN STATE UNIVERSITY

- Objective: A software development process that provides assurance to dynamically adaptive software
- Approach: Focus on the design and analysis of formal design models and their relationships to requirements and implementations.

Outline

MICHIGAN STATE UNIVERSITY

- Requirements analysis
- Design models (focus)
- Implementation
- Discussion and conclusions

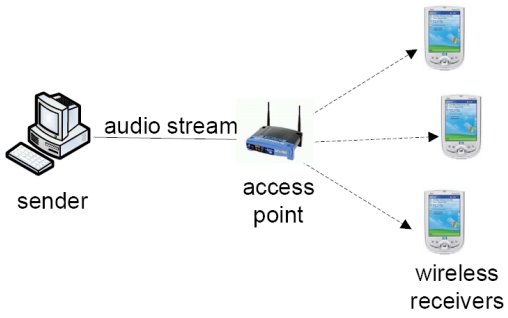
What Is Dynamic Adaptation

MICHIGAN STATE UNIVERSITY

- Program that changes its behavior at runtime
  - A set of programs P_1, P_2, …, P_N (FSA)
  - Changing from running in source program P_i to target program P_j
  - May involve intermediate states and transitions

Audio-Streaming Case Study

MICHIGAN STATE UNIVERSITY



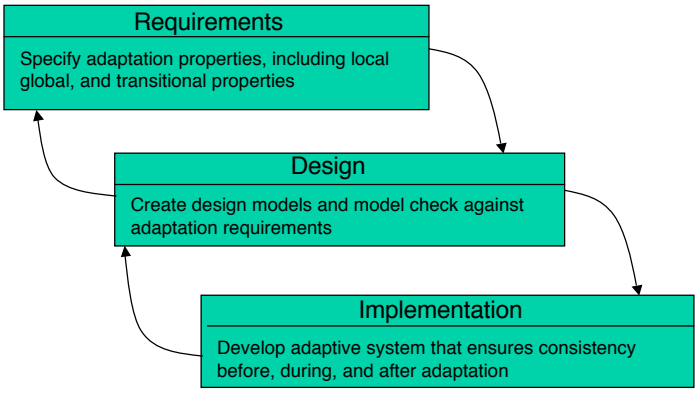
Packet loss-rate in the wireless communication is dynamic
- •High loss-rate
- •Low loss-rate

Adaptive error correction protocols
- •High loss-tolerant and low performance
- •Low loss-tolerant and high performance

RAPIDware Approach

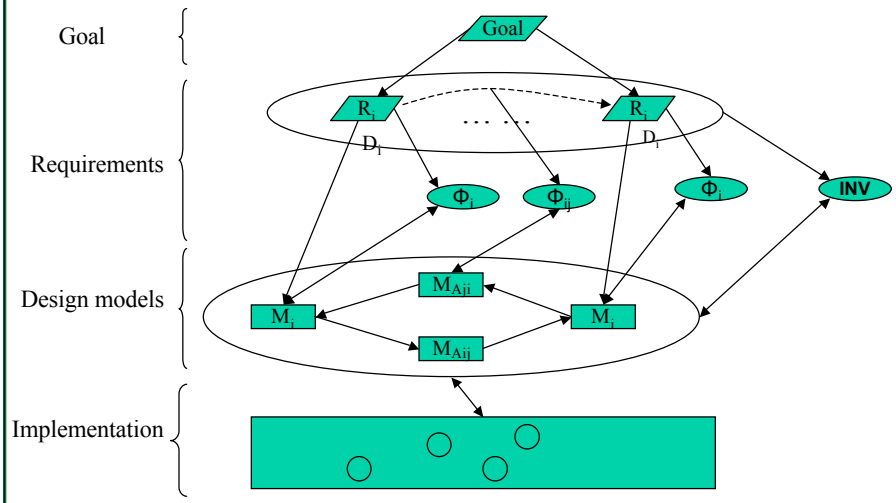
MICHIGAN STATE UNIVERSITY

- Gain assurance in adaptive software through
  - A systematic software development process
  - Application of formal methods at every stage



Overall Process

MICHIGAN STATE UNIVERSITY



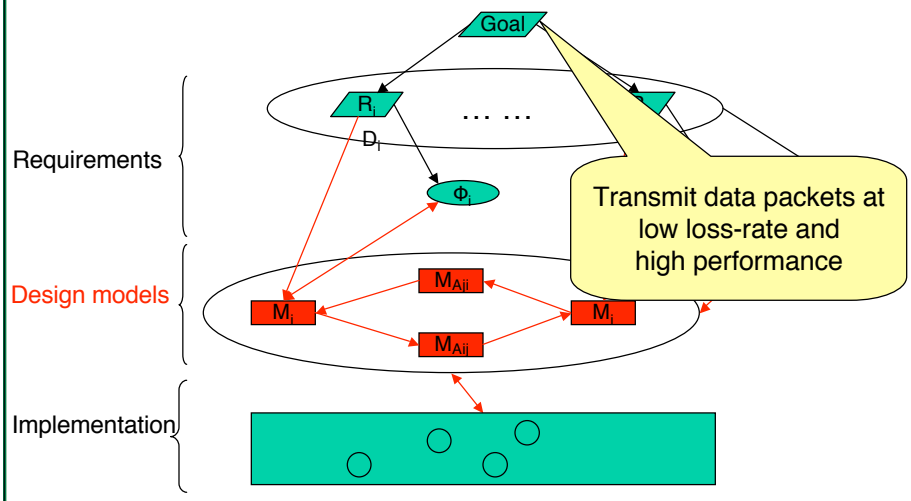
Goal – Requirements

MICHIGAN STATE UNIVERSITY

- Four Level requirements engineering [REFSQ05]
  - Adaptive program is running in a set of domains *D*
  - Each domain D_i is a set of inputs
  - Level 1 RE: Determine D_i and software's reaction to each input in D_i
- Goal-driven models (Mylopolous, van Lamsweerde, et al)
  - Use goal-driven model to analyze possible execution conditions and requirements in each condition

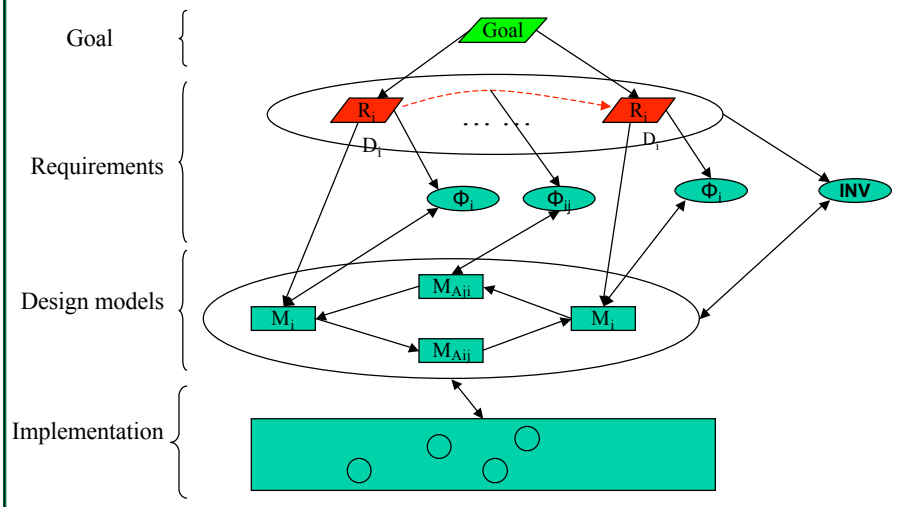
Goal

MICHIGAN STATE UNIVERSITY



Goal to Domains and Requirements

MICHIGAN STATE UNIVERSITY



Requirements Stage

MICHIGAN STATE UNIVERSITY

- Use temporal logics to specify adaptation semantics
  - Developed A-LTL to specify adaptation semantics. [WADS05, JSS06]
  - Developed TA-LTL to express real-time constraints in DASs (dynamically adaptive systems). [EASe06]
  - Precise specification
  - Enable automated analysis

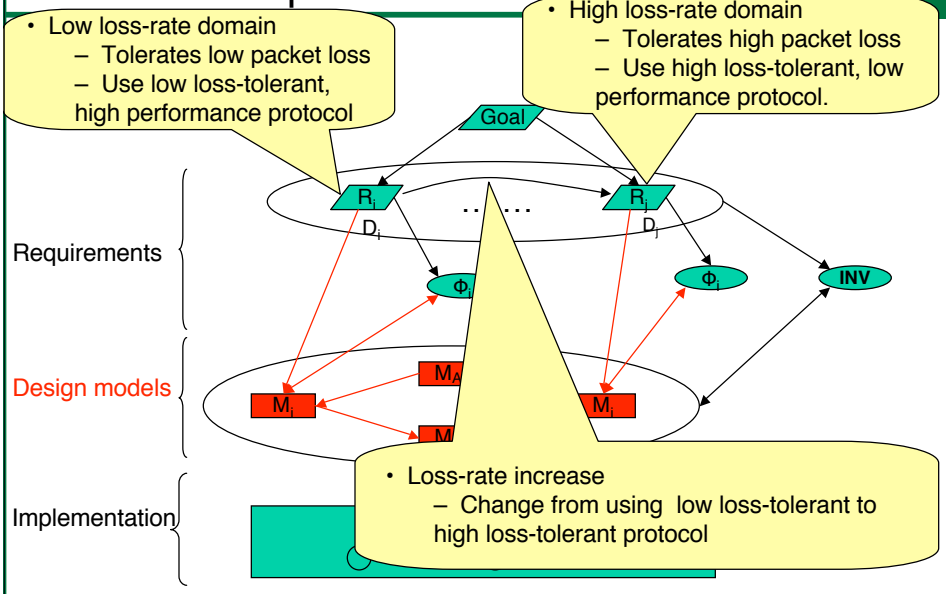
Requirements: Temporal Logic

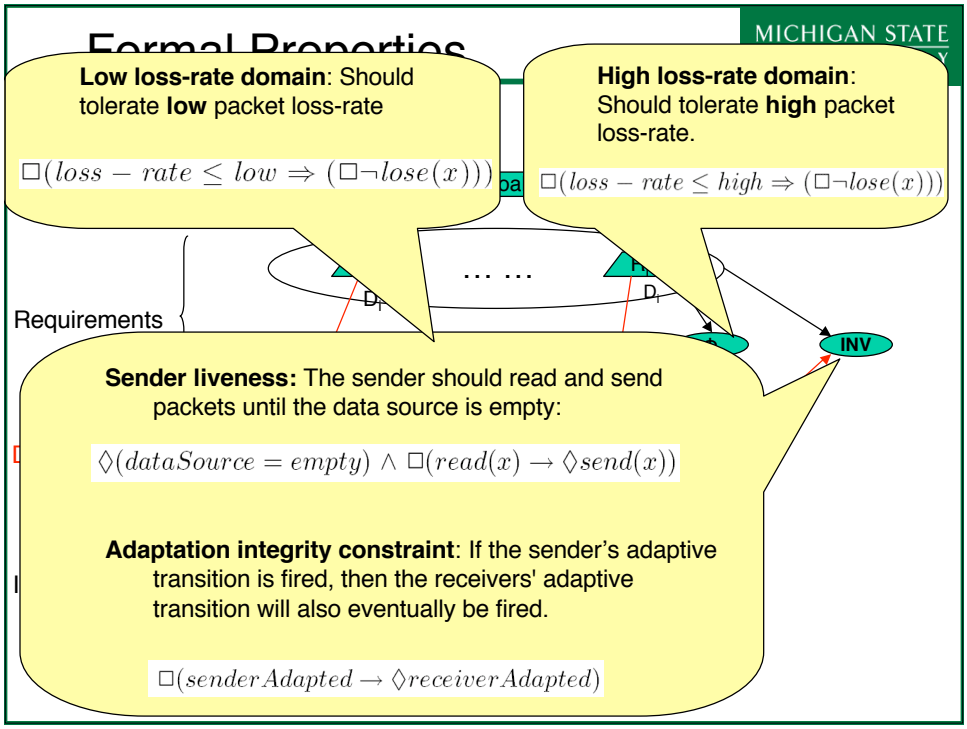
MICHIGAN STATE UNIVERSITY

- We have investigated using temporal logic to specify critical properties in adaptive programs [WADS05,JSS06,EASe06]
  - **Local properties:** Use LTL to specify program properties in each execution domain
  - **Global invariants**: Use LTL to specify properties of the adaptive program
  - **Transitional properties:** Use A-LTL to specify adaptations from one domain to another

Goal to Requirements

MICHIGAN STATE UNIVERSITY





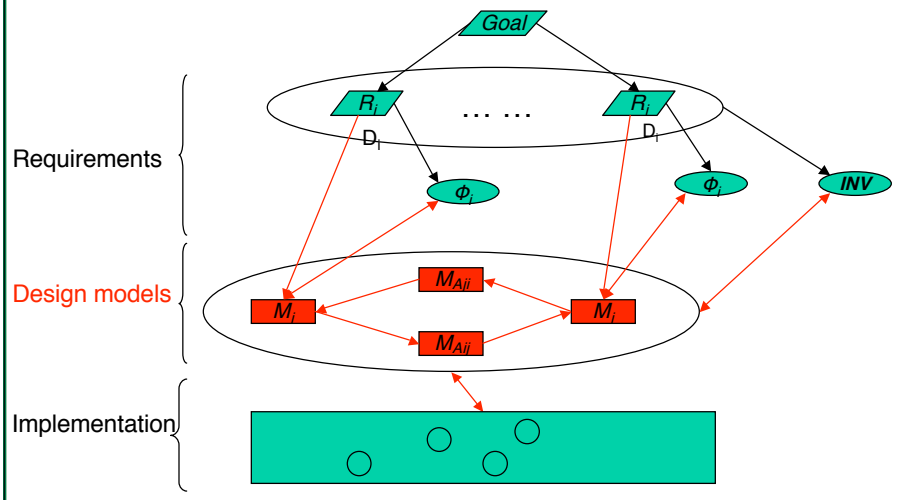
Design Stage

MICHIGAN STATE UNIVERSITY

- Model-based design technique
  - Use finite state machine to model DAS [ICSE06]
  - Non-adaptive models are separated from adaptive models.
  - Support state transfer
  - Support analyses
    - Simulation
    - Model checking

Design to Implementation

MICHIGAN STATE UNIVERSITY



Model-Based Development

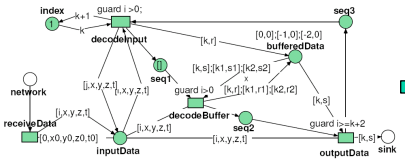
MICHIGAN STATE UNIVERSITY

- Model adaptive system using Petri Nets
  - Steady-state models M_i for each D_i
  - Adaptation models M_{Aij} from D_i to D_j

[ICSE06]

- Model check against global invariants
- Automatically generate adaptive programs

Petri Nets



control

Java program

```
pubic class sender extends…{
   readData(){...}
   encode() {…}
   sendData(){…}
    …
}
```

Objectives

MICHIGAN STATE UNIVERSITY

- Separation of concerns
  - Separate adaptive behavior from non-adaptive behavior
- General
  - Can be used for other state-based formalisms
- Flexible
  - Support state transfer.
- Formal
  - Amenable to analysis, e.g., model checking, simulation, etc

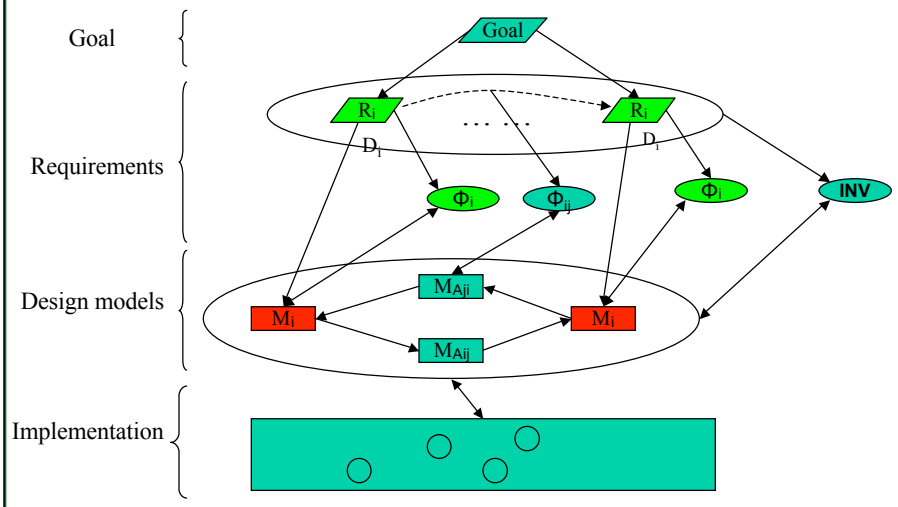
Requirements to Design

MICHIGAN STATE UNIVERSITY

- Requirements vs Design
  - Requirements: declarative
  - Design: operational
- State-based model design
  - Petri nets
  - graphical (intuitive)
  - Formal
  - Numerous analysis tools e.g. Renew, Maria
- Two types of models
  - Steady-state models M_i for each D_i
  - Adaptation models M_{Aij} from D_i to D_j
- Verify the models against local properties and global properties

Local Requirements-Steady-State Models

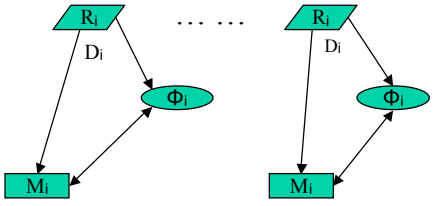
MICHIGAN STATE UNIVERSITY



Steady-State Models

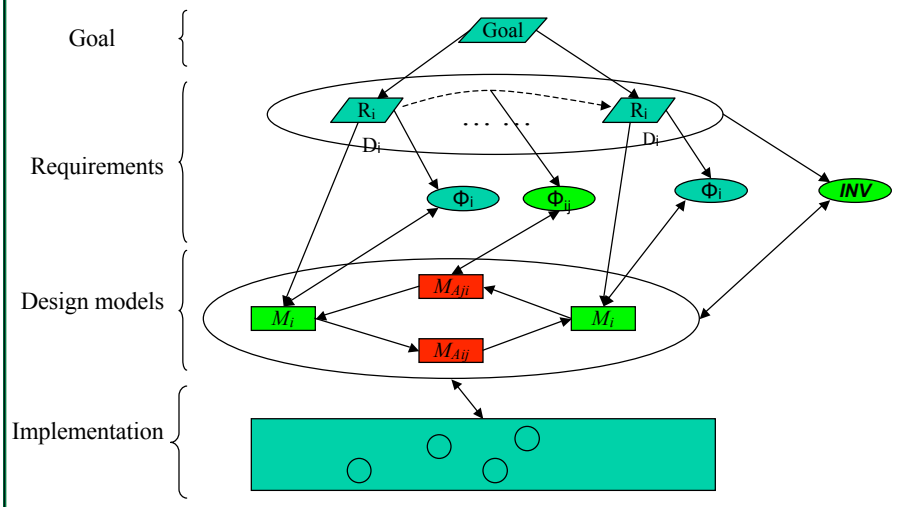
MICHIGAN STATE UNIVERSITY

1. Build a design model for each domain.
2. Run simulation to validate the requirements of each domain
3. Model check the models against local properties



Global Invariants-Adaptation Models

MICHIGAN STATE UNIVERSITY



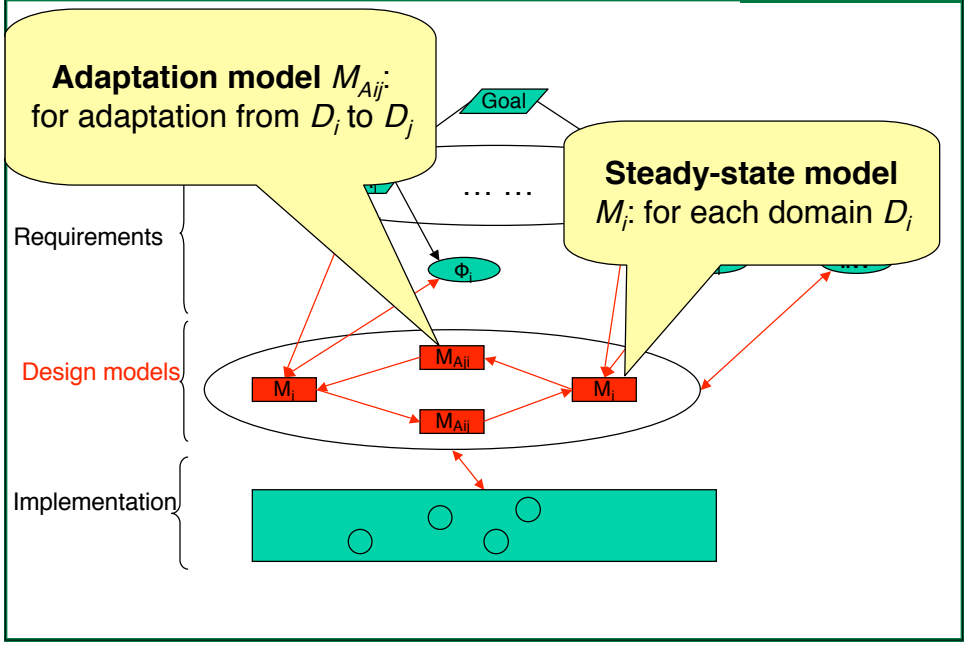
Adaptation Models

MICHIGAN STATE UNIVERSITY

1. Build an adaptation design model M_{Aij} for each adaptation from one steady-state program M_i to another M_j
2. Run simulation to validate the adaptation
3. Model check the adaptation models against global invariants and transitional properties.

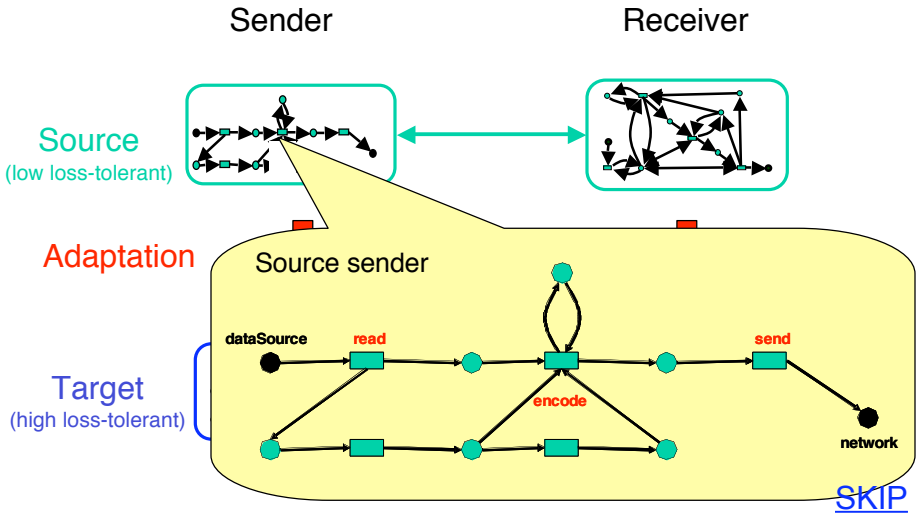
Types of Design Models

MICHIGAN STATE UNIVERSITY



Design Models

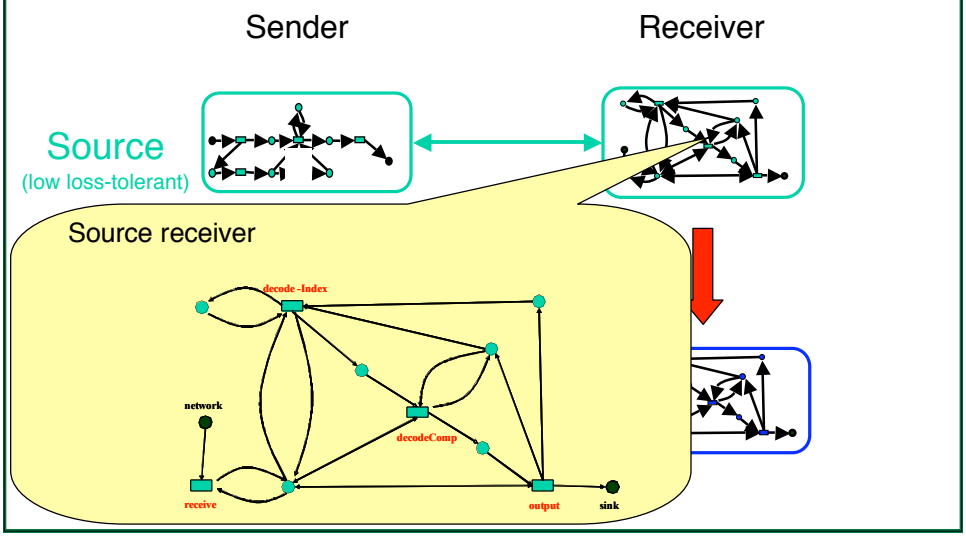
MICHIGAN STATE UNIVERSITY



SKIP

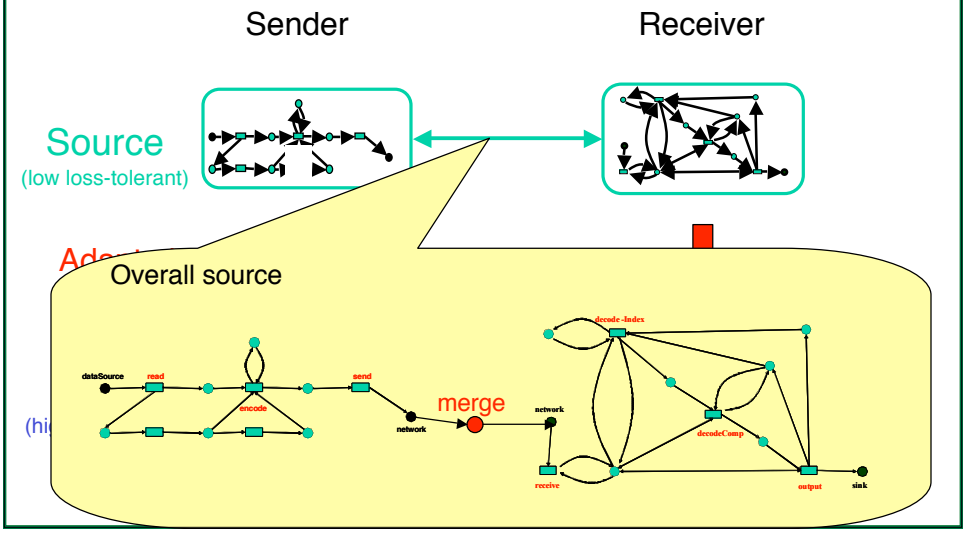
Design Models

MICHIGAN STATE UNIVERSITY



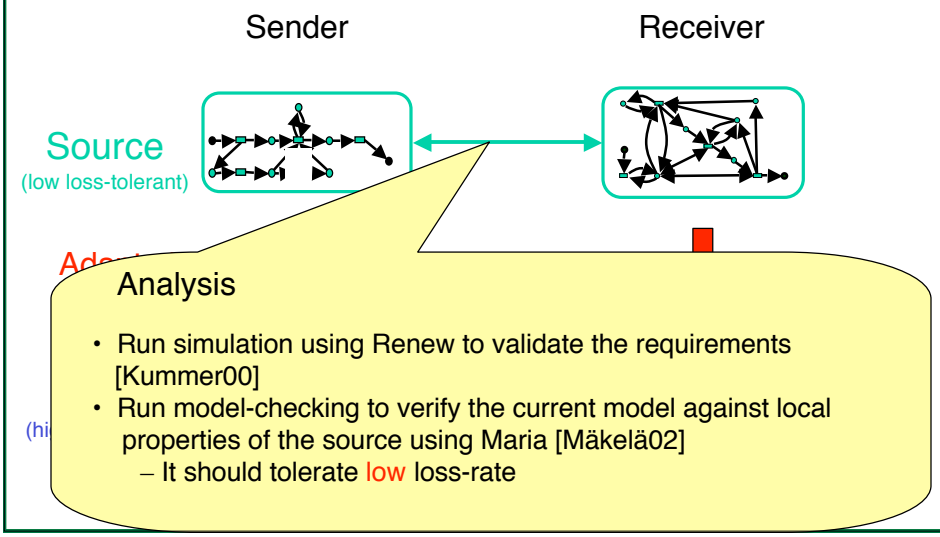
Design Models

MICHIGAN STATE UNIVERSITY



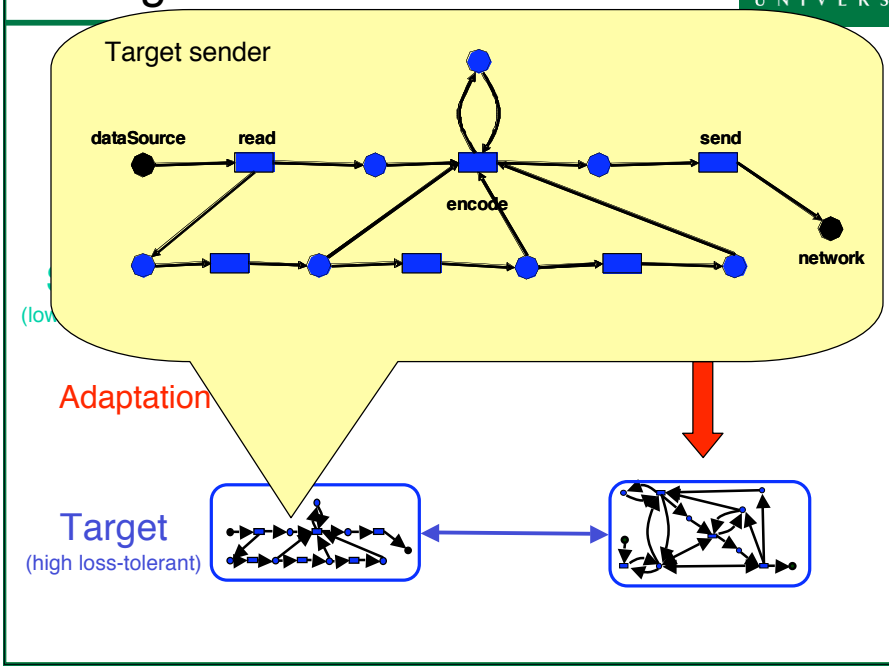
Design Models

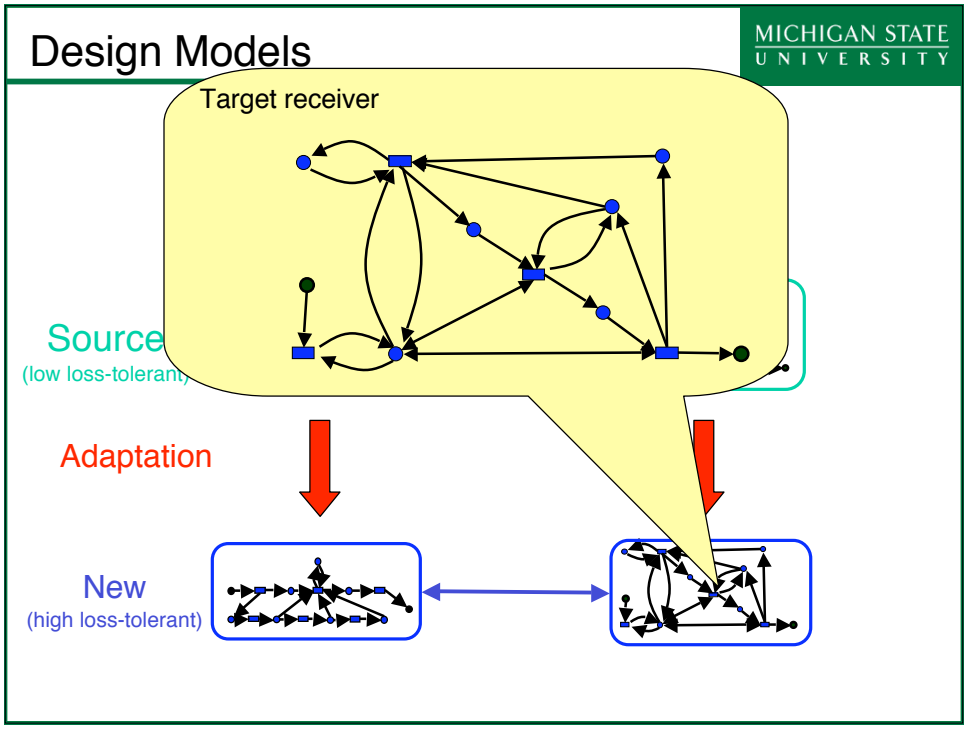
MICHIGAN STATE UNIVERSITY

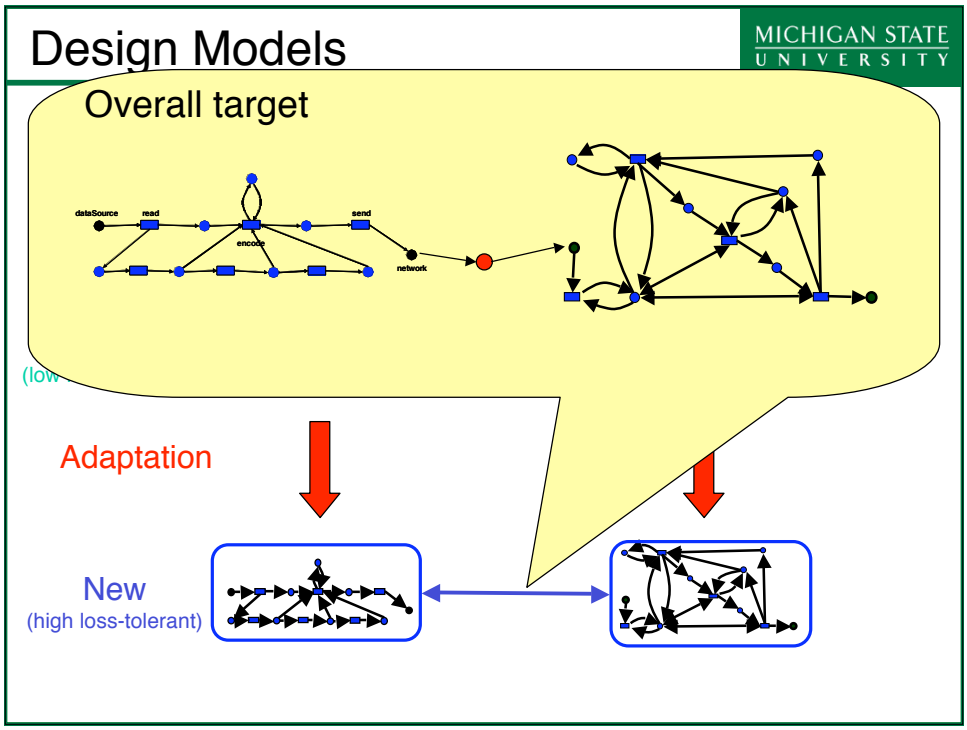


Design Models

MICHIGAN STATE UNIVERSITY




Design Models
MICHIGAN STATE UNIVERSITY
Target receiver
Source
(low loss-tolerant)
Adaptation
New
(high loss-tolerant)


Design Models
MICHIGAN STATE UNIVERSITY
Overall target
dataSource
read
send
encode
network
Adaptation
New
(high loss-tolerant)

Design Models

MICHIGAN STATE UNIVERSITY

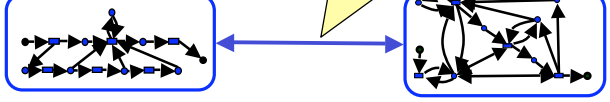
Analysis

- Run simulation using Renew to validate the requirements
- Run model-checking to verify target model against local properties of the source
  - It should tolerate high loss-rate

(low

Adaptation

New
(high loss-tolerant)



Design Models

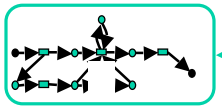
Sender adaptation

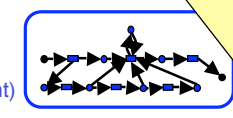
Sender

Source
(low loss-tolerant)

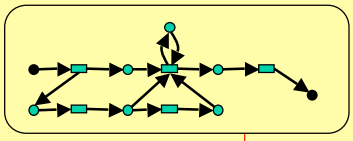
Adaptation

Target
(high loss-tolerant)



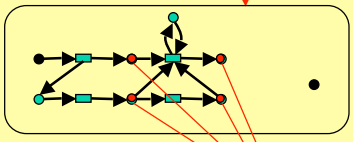


source sender



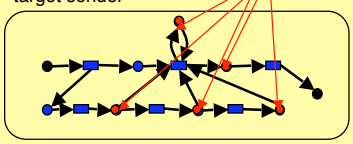
(1) restrict

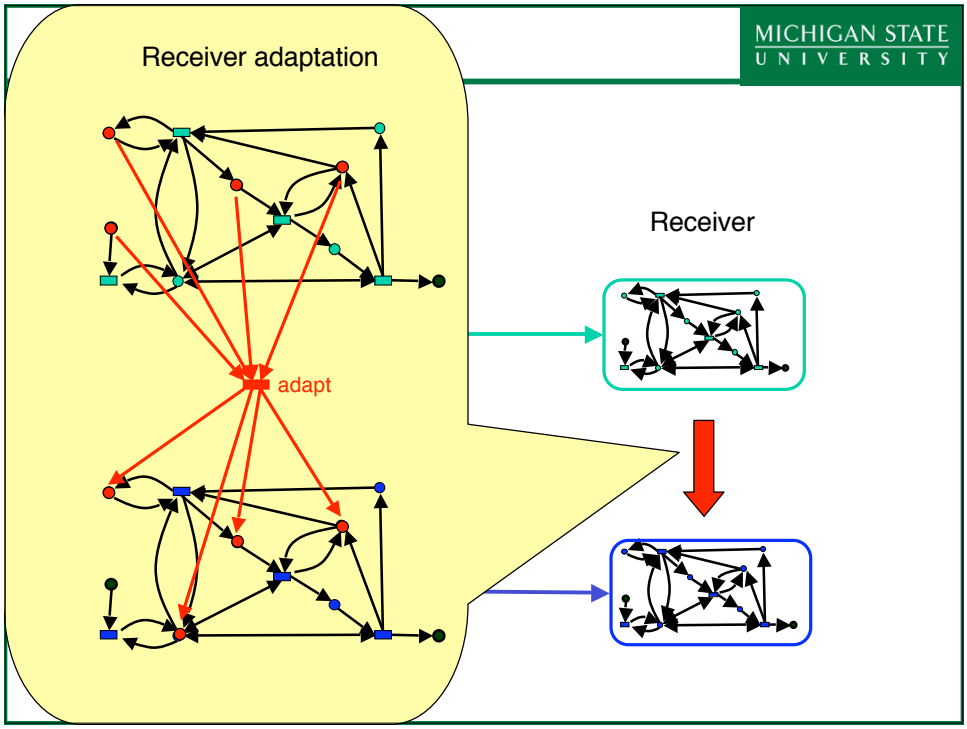
sender restricted

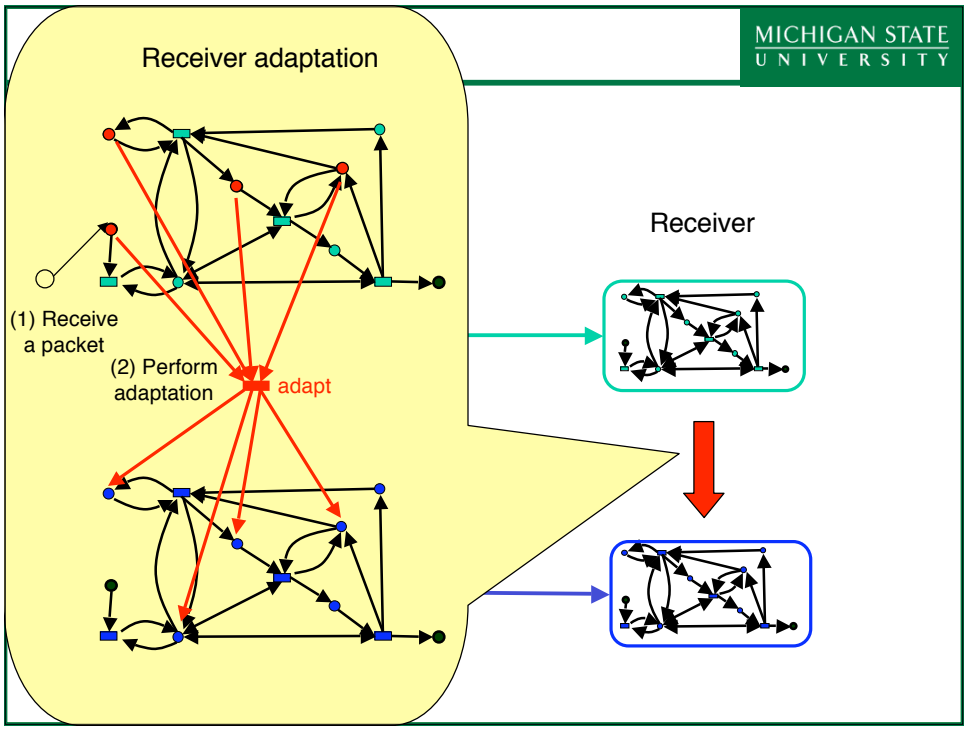


(2) adapt

target sender

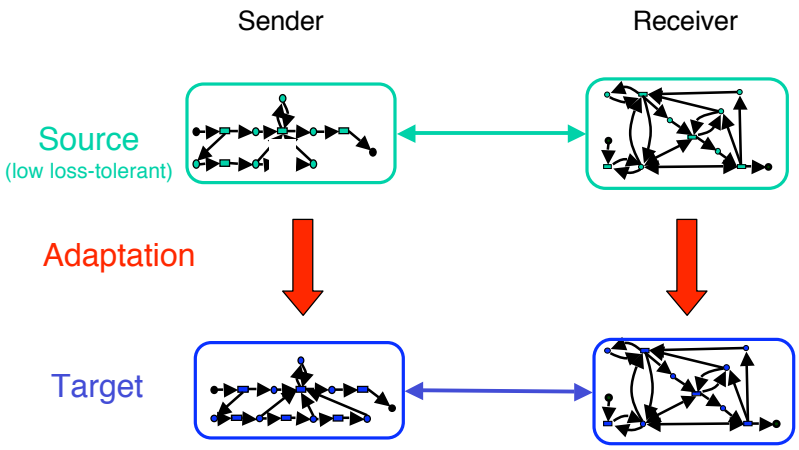



Receiver adaptation
MICHIGAN STATE UNIVERSITY
Receiver
adapt


Receiver adaptation
MICHIGAN STATE UNIVERSITY
Receiver
(1) Receive a packet
(2) Perform adaptation
adapt

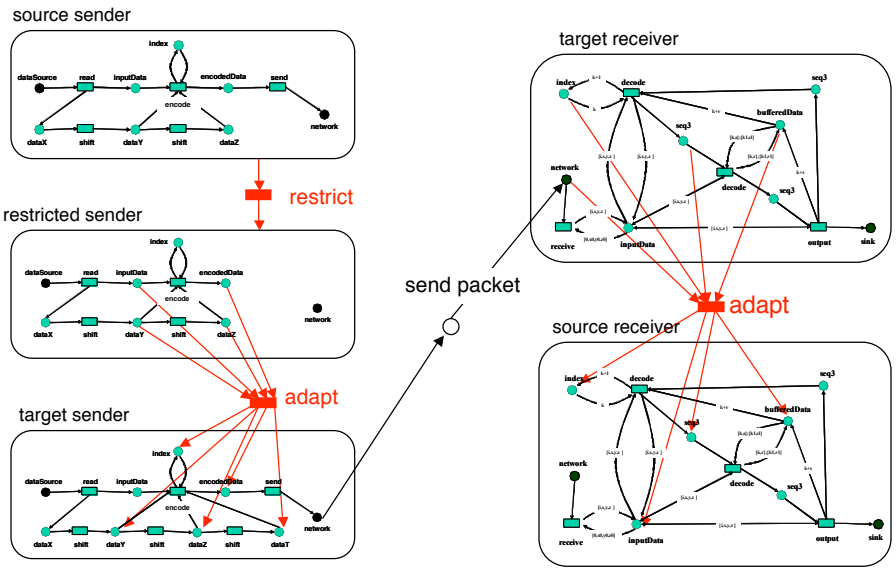
Design Models

MICHIGAN STATE UNIVERSITY



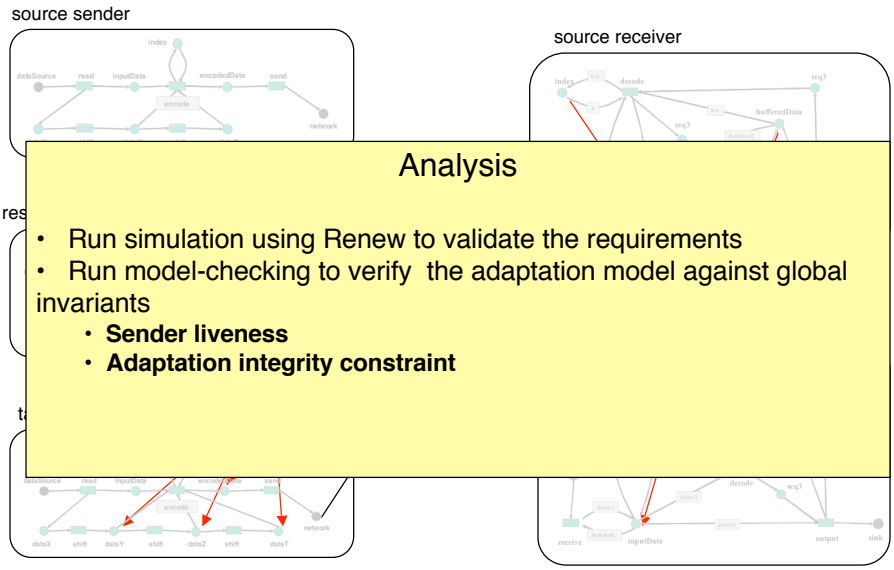
Overall Adaptation Model

MICHIGAN STATE UNIVERSITY



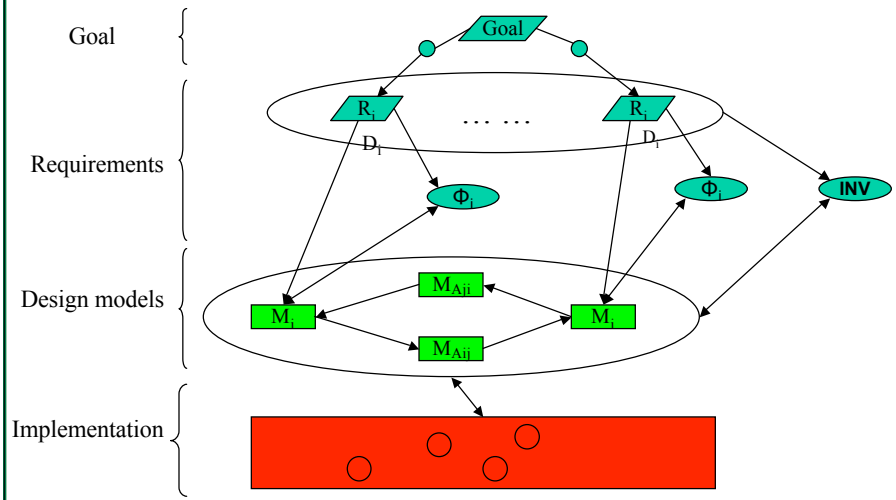
Design Models

MICHIGAN STATE UNIVERSITY



Design to Implementation

MICHIGAN STATE UNIVERSITY

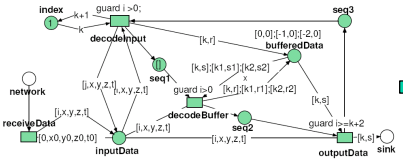


Reifying the Models

MICHIGAN STATE UNIVERSITY

- Prototyping (supported by Renew)
  - Petri net models can be integrated in Java programs to drive Java function calls.

Petri Nets



control

Java program

```
pubic class sender extends…{
  readData(){...}
  encode() {…}
  sendData(){…}
   …
}
```

Reifying the Models

MICHIGAN STATE UNIVERSITY

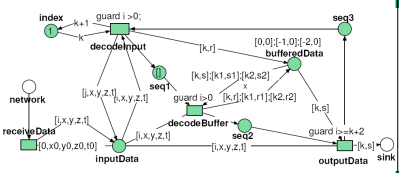
- Model-based testing (supported by Renew)
  - Java program can invoke corresponding Petri net transitions.
  - Use Petri nets to test allowable execution paths.

Java program

```
pubic class sender extends…{
  readData(){...}
  encode() {…}
  sendData(){…}
   …
}
```

control

Petri Nets



Discussion and Conclusions

MICHIGAN STATE UNIVERSITY

- A model-based adaptive software development process.
  - Crosscuts requirements, design, and implementation
  - Steady-state programs requirements and models are specified **separately** from those for adaptations
  - Identify **quiescent states** to be a property of the adaptation, rather than properties of each steady-state program
  - Support flexible **state transfer**

Future Work

MICHIGAN STATE UNIVERSITY

- Integrate our approach with ADL approaches
- Explore modular model-checking of global invariants [Modular06]
- Explore run-time model checking
- Further validation
  - Apply to other adaptation domains
  - Apply to other formalisms

Acknowledgements

MICHIGAN STATE UNIVERSITY

- Faculty and students in the Software Engineering and Network Systems Laboratory at Michigan State University
- ICSE reviewers
- Grants: This work was supported in part by the U.S. Department of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744, and in part by National Science Foundation grants CCR-9901017, EIA-0000433, EIA-0130724, CCF-0541131, and CNS-0551622, and ITR-0313142, and by Siemens Cooperate Research, and a Michigan State University Quality Fund Concept Grant.