Sage: Model-Driven Agile Development of Reactive Distributed Systems

James Kirby, Jr.

Center for High Assurance Computer Systems

US Naval Research Laboratory
Code 5542
Washington, DC

James.Kirby@nrl.navy.mil

Some Challenges to Developing High Assurance Software

- Software that puts life, property, security at risk requires high assurance
  - Long, document-oriented development process
  - Documents provide evidence high assurance requires
  - Hard to distinguish ***motion from progress***
- ***Either*** software functionality not captured ***precisely*** until coding
  - Unable to provide ***precise*** answers to questions about software functionality until late in development
- ***Or*** software functionality captured multiple times
  - Maintaining consistency is a large ***sink of effort***
  - Failing to maintain consistency is a fertile ***source of error***

Agile Development Process Can Distinguish Motion from Progress

- **Working software** is the primary measure of progress
  - Highest priority is to satisfy the customer through **early and continuous delivery of valuable software**.
  - **Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale
- However:
  - Agile development chooses **working software** over **comprehensive documentation**
  - High assurance ***requires*** this documentation for evidence

Model-driven Agile Development

- **Working software and comprehensive documentation**
  - The model **is** the documentation
  - Supports high assurance certification
- Model-driven development supporting an agile process
  - Model subsets
  - Model slices
  - Incomplete models
  - Model development driven by external events
    - E.g., customer requests

Sage Supports Model-driven Agile Development

- Record software behavior precisely **throughout development**
  - Developers create **precise models of behavior**
  - Provide **evidence required of high assurance software**
- Minimal redundancy of models reduces effort and opportunity for error
- Distinguish ***progress from motion***
  - User validation of software functionality throughout development
    - **Simulation** of functionality throughout development
    - Frequent delivery of working software
  - **Automated analyses** throughout development provide **evidence required of high assurance**
- Process neutral
  - Supports agile process (and traditional waterfall)
  - Method and tool support capturing decisions as they're made

Model Driven Development

- Platform Independent Model (***PIM***)
  - High level model of an application independent of implementation technology
- Platform Specific Model (***PSM***)
  - Transformation of a PIM
  - Model of application that reflects chosen implementation technology
- Code
  - Transformation of PSM

Methods *versus* Process

- **Software methods** are concerned with how to record, organize, evaluate ***decisions***
  - What is the boundary of the software with its environment?
  - What are the pieces into which the software decomposed?
- **Software process** is concerned with the ***ordering*** of decisions and with the ***use of resources***
  - When to start coding?
  - When to deliver software?
  - Who will do the work?

Some Key Development Decisions

- What is the boundary of the software with its environment?
- What is the software functionality (or behavior or business logic)?
- How is the functionality organized into *design elements* to meet performance and Quality of Service goals?
  - Make good use of CPU cycles, network bandwidth, etc.
- How is the functionality organized into *design elements* to support intellectual control and to meet maintainability and reuse goals?

Sage Platform Independent Model

- Sage allocates development decisions to four models comprising the PIM
  - Environmental model
  - Behavioral model
  - Design model
  - Run-time model
- Each decision captured in one place
  - Decision shared by several models is captured in one model and reflected to the others

Sage PIM Comprises Four Models

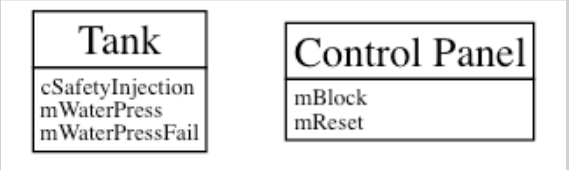
Behavioral Model

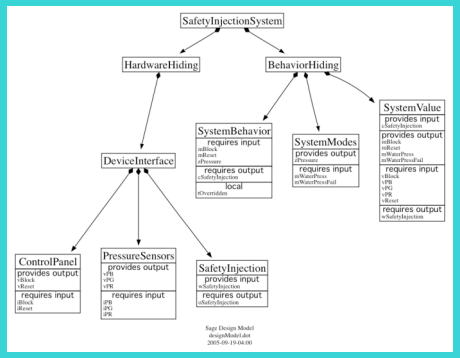
Captures software functionality.

cSafetyInjection =
f (zPressure,
mWaterPressFail,
tOverridden)

Environmental Model

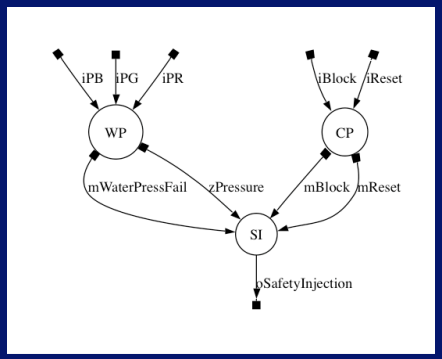
Captures boundary of a system with its environment.





Design Model

Decomposition of software functionality supporting design, maintenance, and reuse goals.

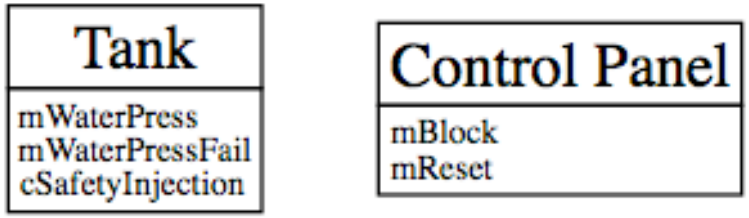


Run-time Model

Decomposition of software functionality supporting performance and QoS goals.

Environmental Model

- Objects in the environment and their ***attributes***
- These ***environmental attributes*** specify the boundary of the software with its environment
  - What the software can ***sense***, ***control***, or ***affect***
- Attribute declarations shared with Behavioral, Design, and Run-Time Models.
- Association of attributes with objects unique to environmental model



Sage Environmental Model
SISUMLEnvironment.dot
1/21/2005jck

Behavioral Model

- Comprises a set of dictionaries
- Attribute dictionary
  - Name, type, interpretation
  - Class of attribute, e.g., environmental, physical input/output, virtual input/output
- Function dictionary precisely captures software functionality
  - Each function specifies value of an attribute
- Type dictionary
- Constant dictionary
- Application property dictionary
  - Tools available for verifying software satisfies application properties

Functions in the PIM

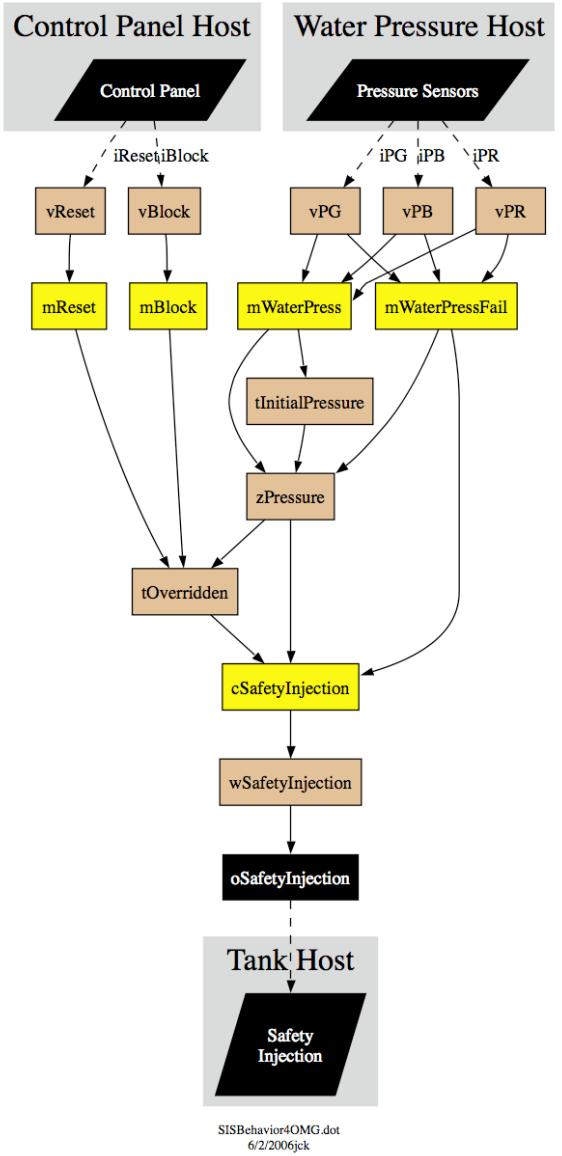
- Representation supports human review
  - E.g., for missing cases, non-determinism
- Functions always defined
  - From system initialization forward

cSafetyInjection =

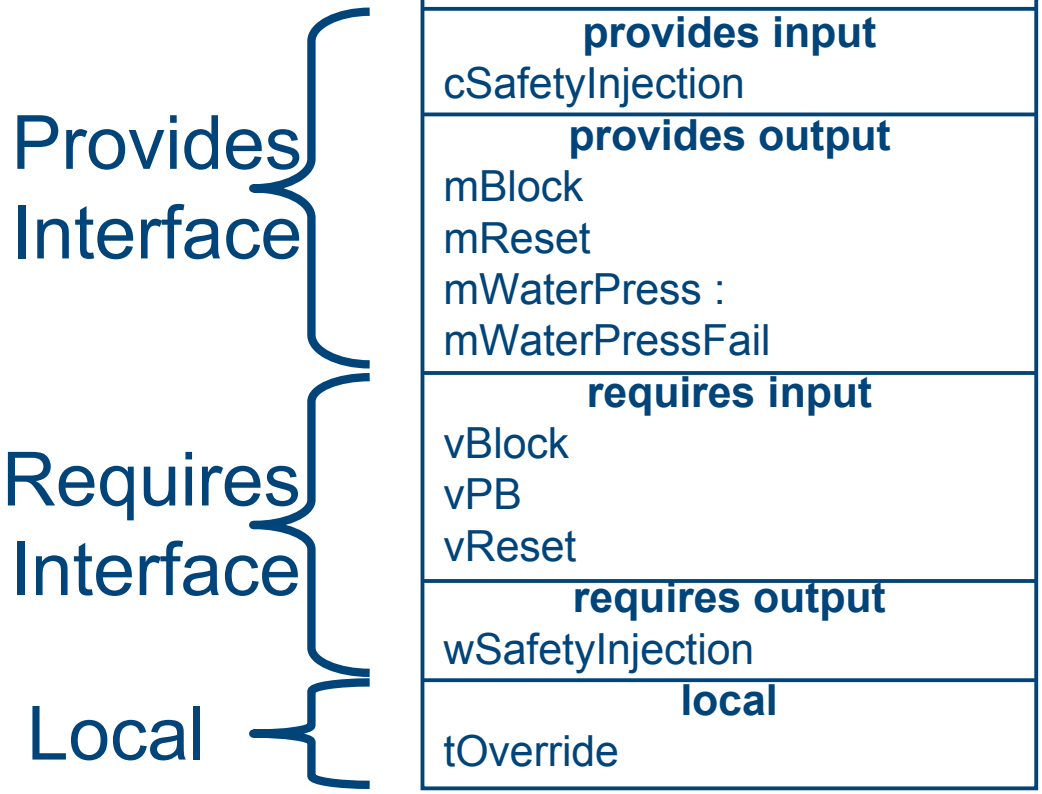
Context	Conditions	
zPressure		
high, permitted	! mWaterPressFail	mWaterPressFail
tooLow	tOverridden & ! mWaterPressFail	tOverridden | mWaterPressFail
	off	on

Functions in the PIM

- Synchronous semantics
- Acyclic current state dependency
- External stimuli drive system
  - ***Independent*** attributes
- Underlying model supports automated analyses
  - Consistency and completeness
  - Application-specific properties
    - E.g., safety properties



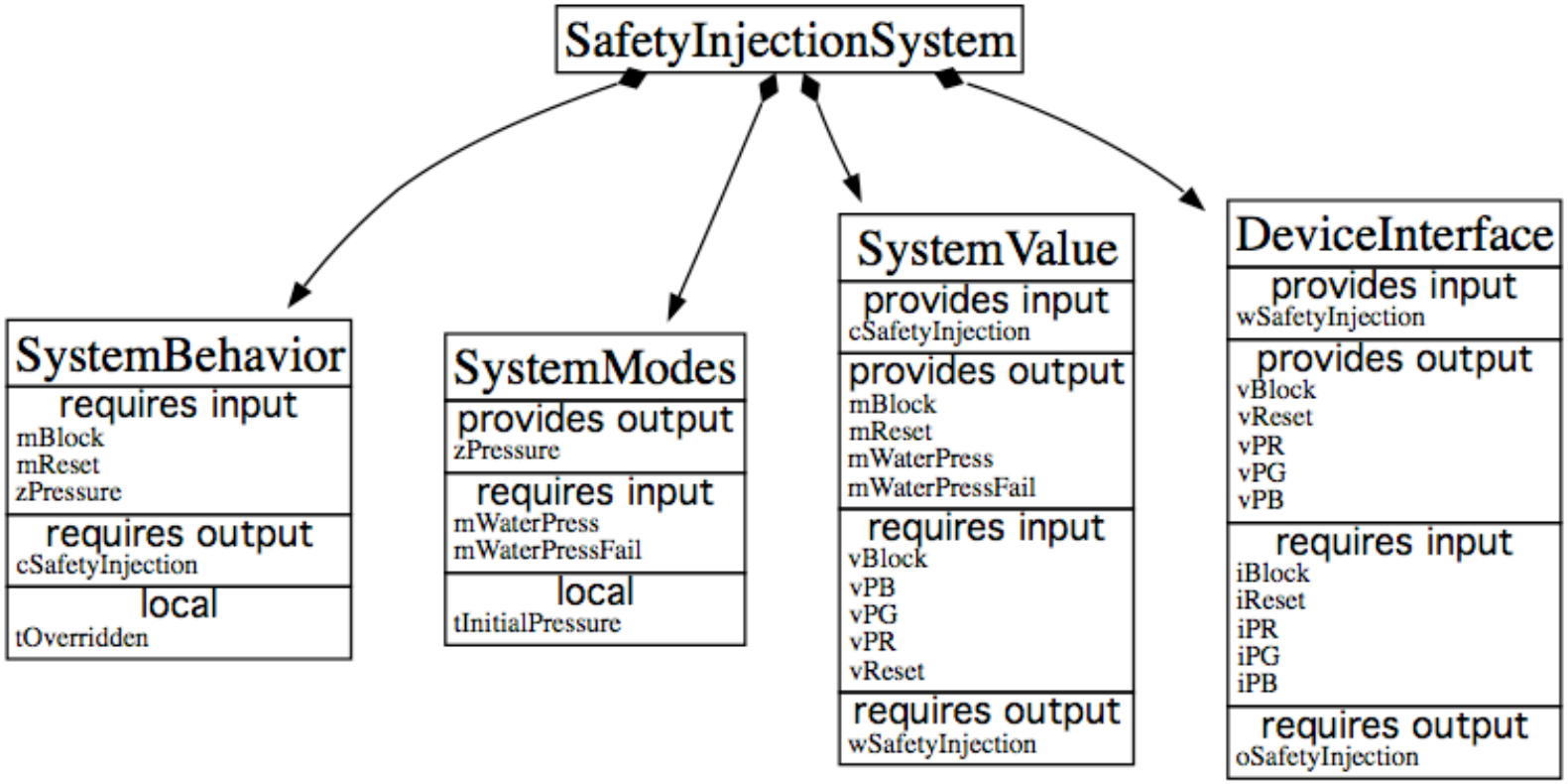
Design Classes and Run-Time Component Interfaces



- Assigning attributes to compartments determines
  - Which attributes are
    - Visible
    - Inputs
    - Outputs
    - Local
  - Where functions go
- Implementation of interfaces is concern of ***PSM***
  - Get/set
  - Call-back
  - *Et cetera*

Design Model

- Decomposition of software functionality supporting design, maintenance, and reuse



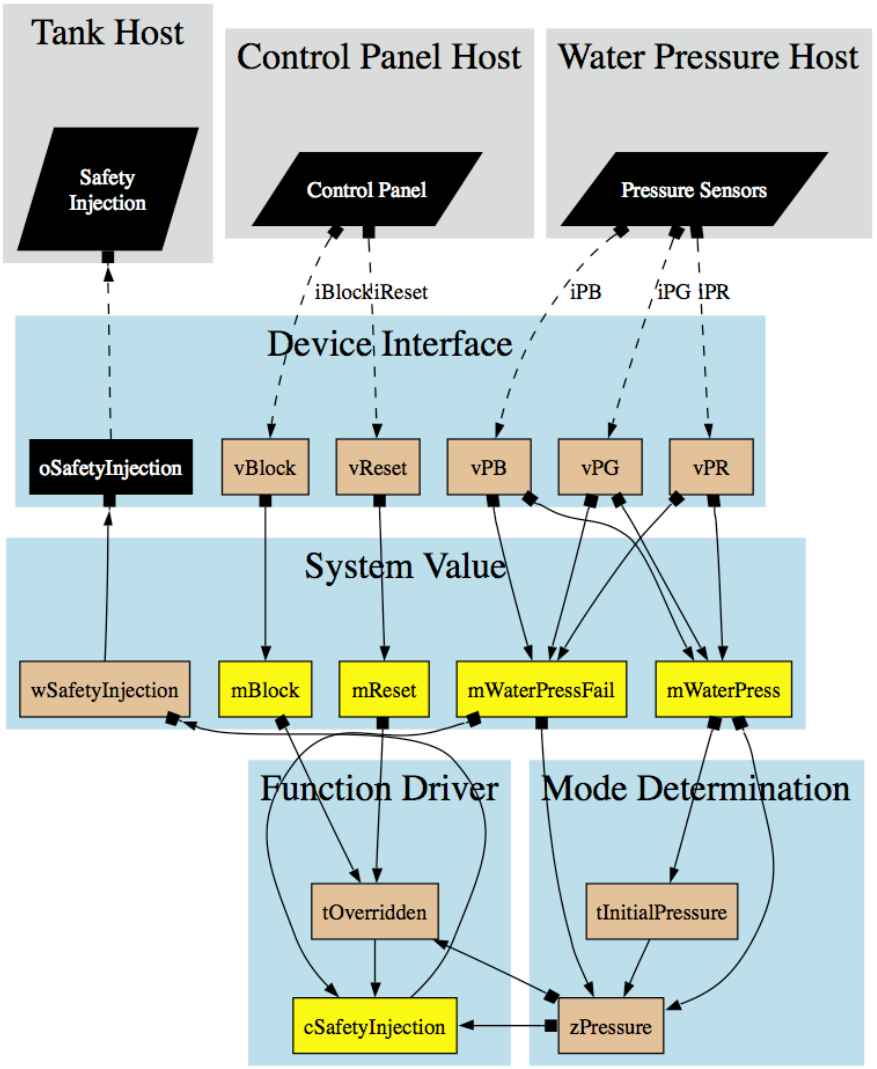
Sage Design Model
designModel.dot
2006-06-05-04:00

Design Class

- Design classes capture decisions supporting design, maintenance, reuse goals
- Attribute declarations shared with other models
- ***Functions*** in Behavioral Model provide values of ***output*** and ***local*** attributes
- Services and other classes provide values of ***input*** attributes

System Value
The rules defining the values of monitored and term attributes and how to set the values of the controlled attributes.
provides input
cSafetyInjection
provides output
mBlock mReset mWaterPress mWaterPressFail
requires input
vBlock vPB vReset
requires output
wSafetyInjection
local
tOverride

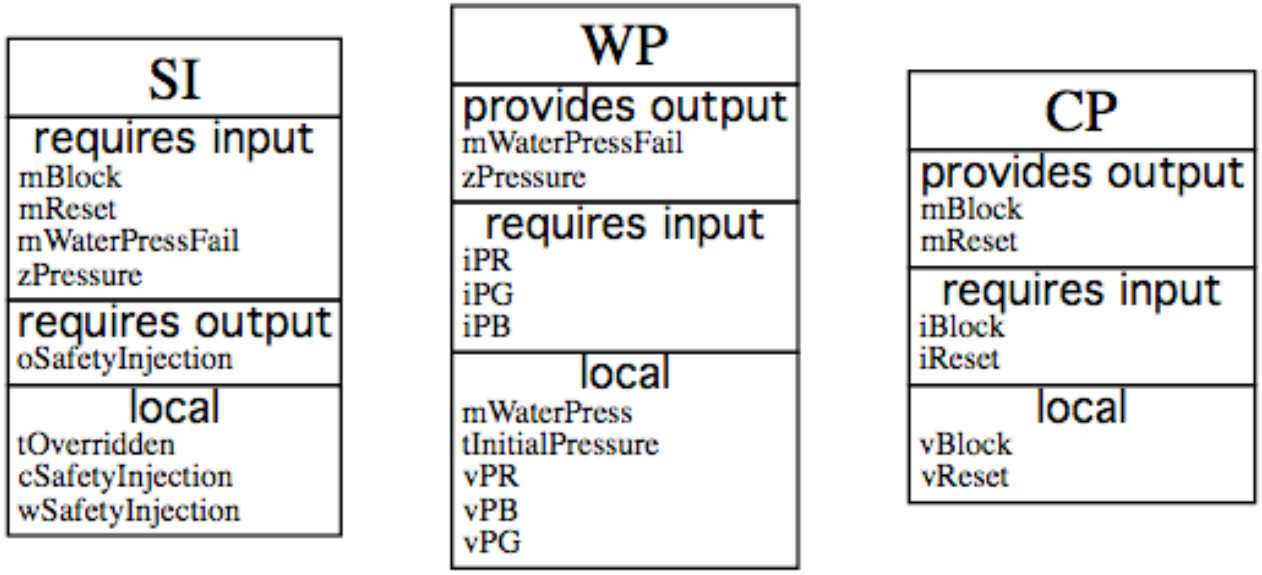
Functions Assigned to Classes



SISDesign.dot
6/5/2006jck

Run-Time Model

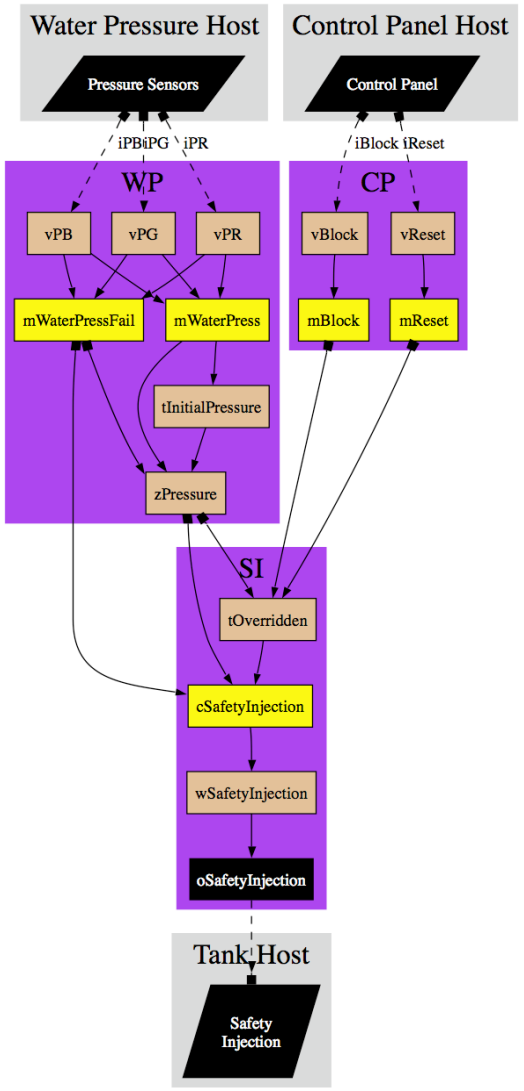
- Decomposition of software functionality supporting performance and QoS
- Synchronous, location-transparent software components



Sage Run Time Model
runTimeModel.dot
2006-06-02-04:00

Functions Assigned to Components

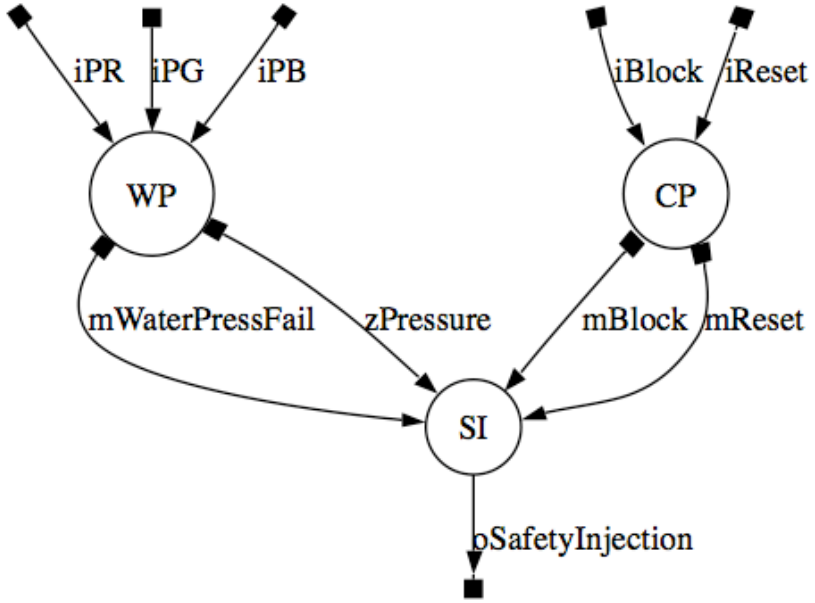
- Attribute declarations shared with other models
- ***Functions*** in Behavioral model provide values of ***output*** and ***local*** attributes
- Services and other components provide values of ***input*** attributes



SISComponent.dot
6/2/2006jck

Run-Time Components

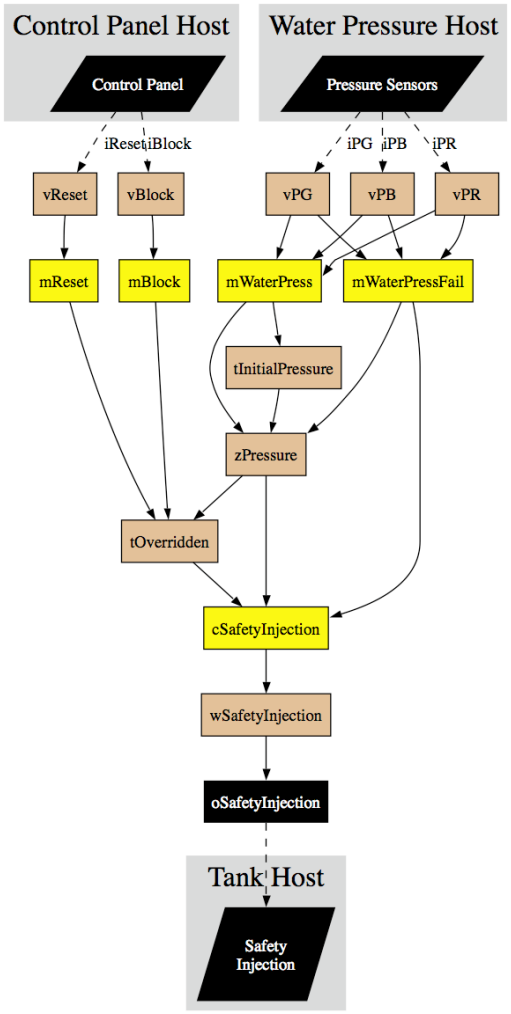
- Run-time components as black boxes
  - Provides interfaces
  - Requires interfaces



Sage Application Model Safety Injection System
Run-time Model Actual
Actual.dot
Mon Jun 19 13:28:56 2006

Supporting Agility

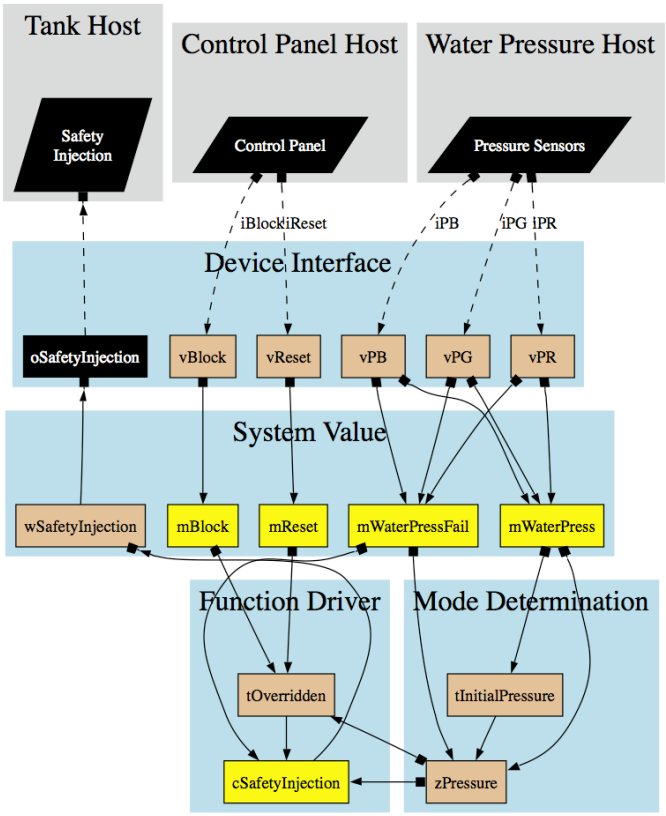
Behavior Model



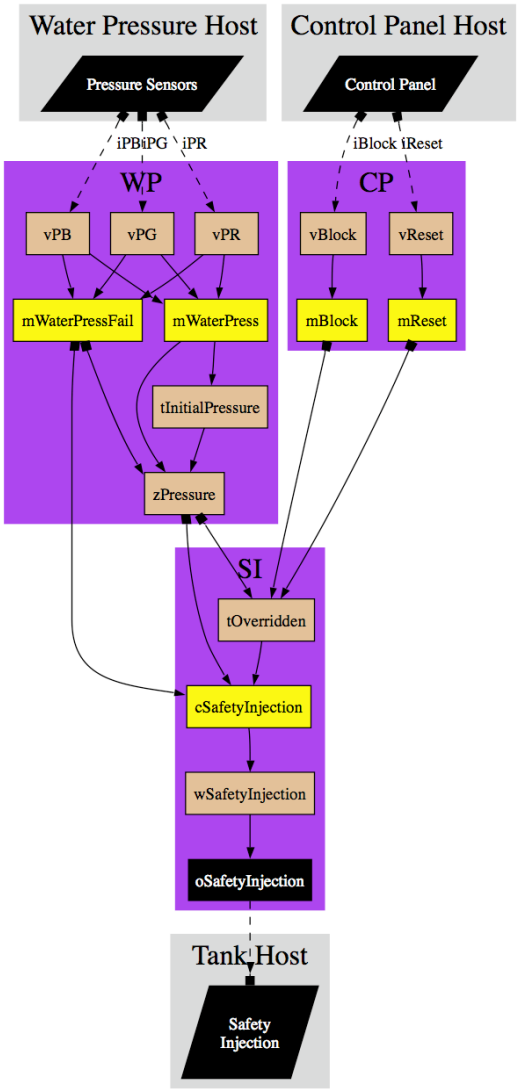
SISBehavior4OMG.dot
6/2/2006jck

Run-Time Model

Design Model

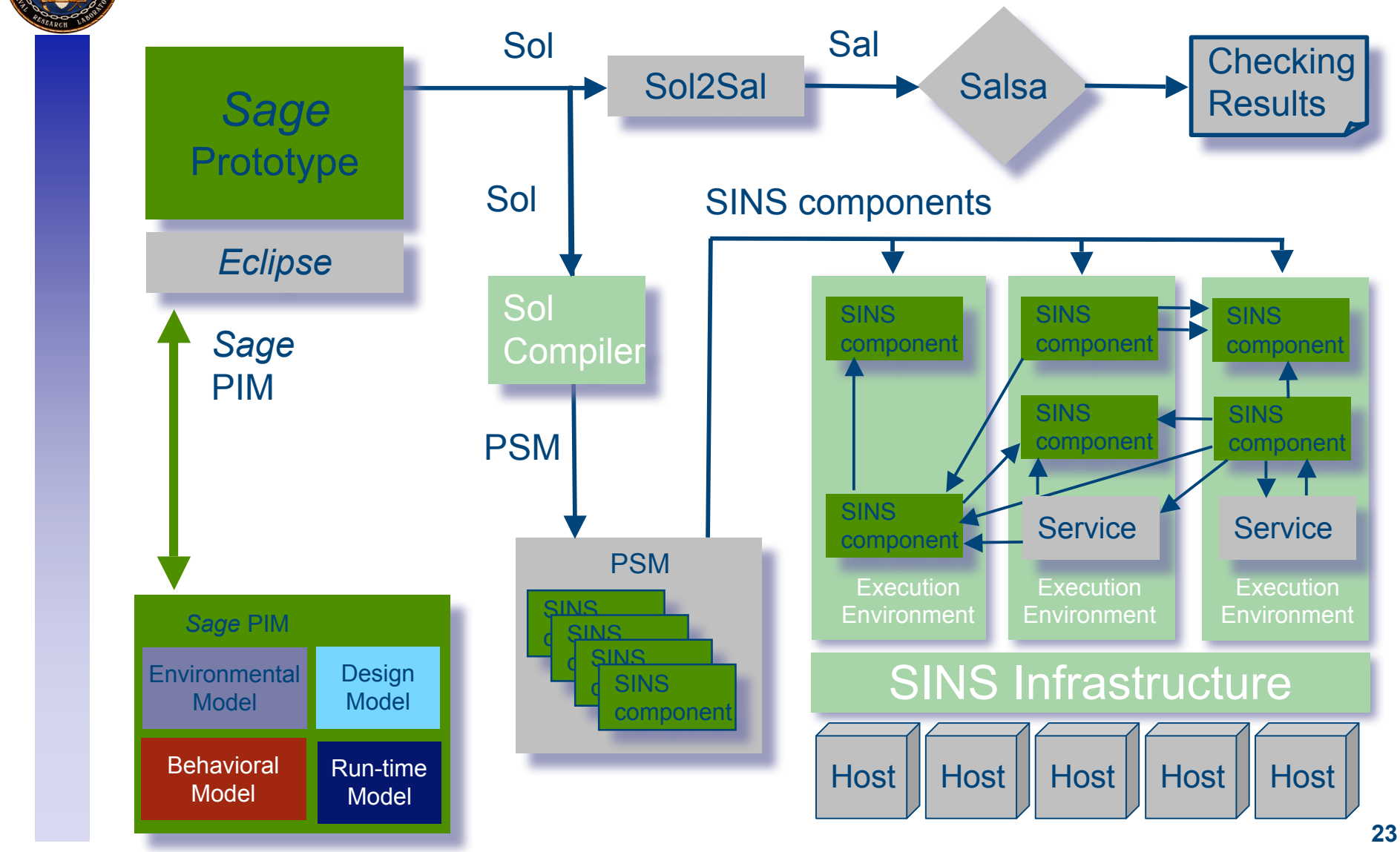


SISDesign.dot
6/5/2006jck



SISComponent.dot
6/2/2006jck

Sage Prototype Tool Chain



Sage Prototype

- Eclipse Modeling Framework (EMF) generates Sage prototype from Sage metamodel
  - *Sage.ecore*
- Additional, hand-coded plug-in provide more refined views of Sage models
- Sage PIM is an instantiation of Sage metamodel
- External tools generate
  - Graphical views
  - Sol
  - Executables

Exercising the Tool Chain

- Weapons Control Panel (WCP)
  - Translation of contractor-developed SRS
  - ***258*** attributes
    - ***108*** independent attributes
    - ***150*** dependent attributes
- Environmental model partitioned ***198*** attributes among ***five*** environmental classes
- Design model not developed
- Two run-time models deployed and executed
  - Sol compiler generated PSM for SINS environment
  - Single, monolithic component model
  - Six component model
  - Demonstrated using subset of contractor-developed scenario
- Sol2sal and Salsa checker exercised
  - Salsa found one non-deterministic function

Summary

- Model-driven
  - Sage PIM provides evidence required by high assurance
    - Supports automated analyses
- Agile
  - Sage supports realistic development processes which are driven by outside events
    - Agile
    - Opportunistic
    - Iterative

Questions?