Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

Using Containers to Enforce Smart Constraints on Real-time Systems

Gabriel Moreno
Scott Hissam

Sponsored by the U.S. Department of Defense

Motivation

Software components are fundamental for the software industry.

- reusable implementation of functionality
- increased flexibility
- standard “plug” interfaces

BUT behavior of component assemblies is unpredictable.

- Little is known beyond plug interfaces.
  - behavior is hidden by design
  - information relevant to runtime behavior is not exposed (e.g., execution times, interaction patterns)

Plug does not imply play.

- Expensive integration and testing are needed to meet quality goals.

Predictable Assembly from Certifiable Components (PACC)

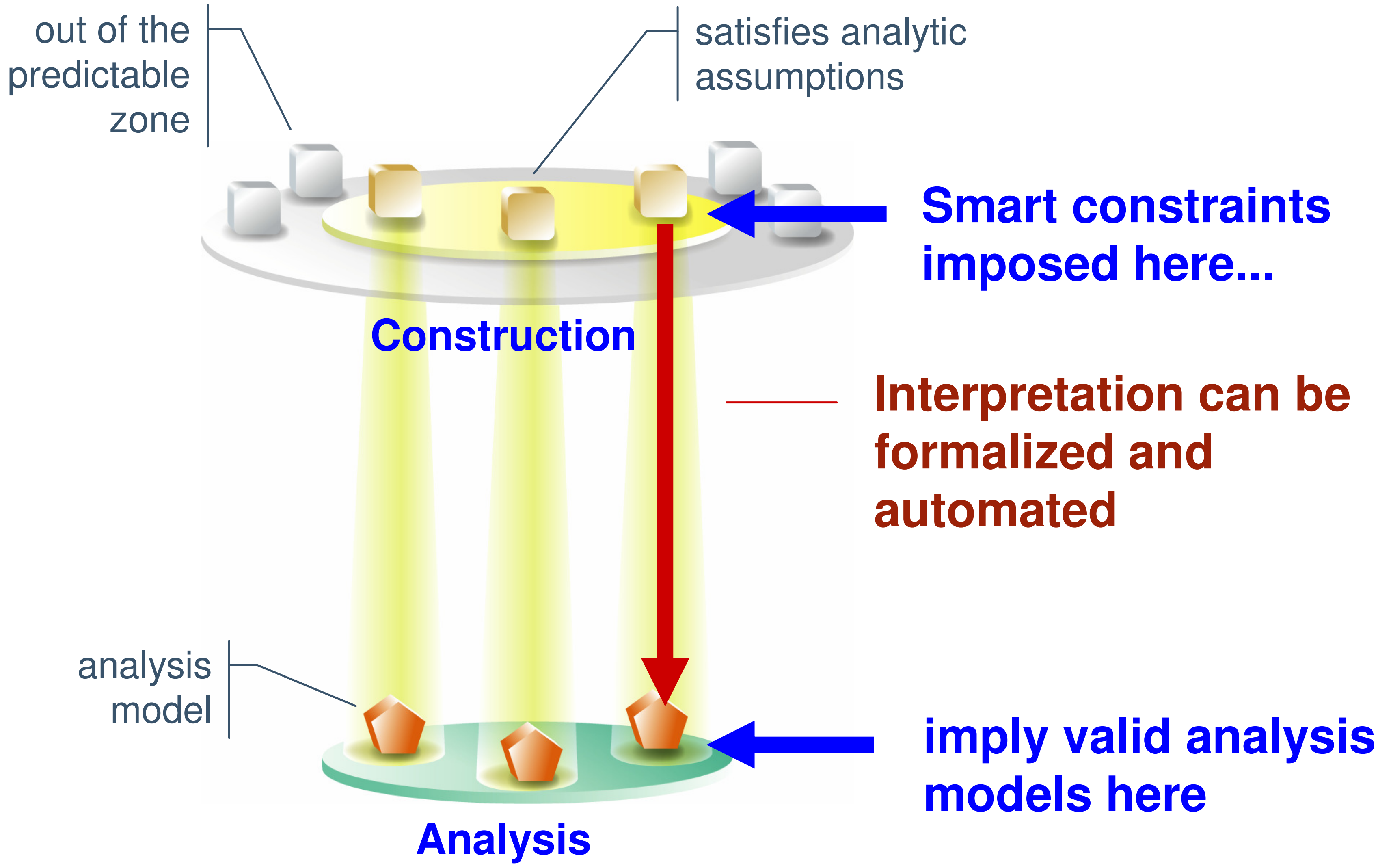
Enable the development of software systems from software components where

- Critical runtime attributes (e.g., performance and safety) are reliably predicted (predictable assembly).
- The properties of software components needed for prediction are trusted (certifiable components).

Our vision is to achieve predictability by construction:

- Constrain the design and implementation to analyzable patterns.
- Use component technology to package and enforce the constraints.
- Result is that plug implies play

Predictable by Construction



out of the predictable zone

satisfies analytic assumptions

Smart constraints imposed here...

Construction

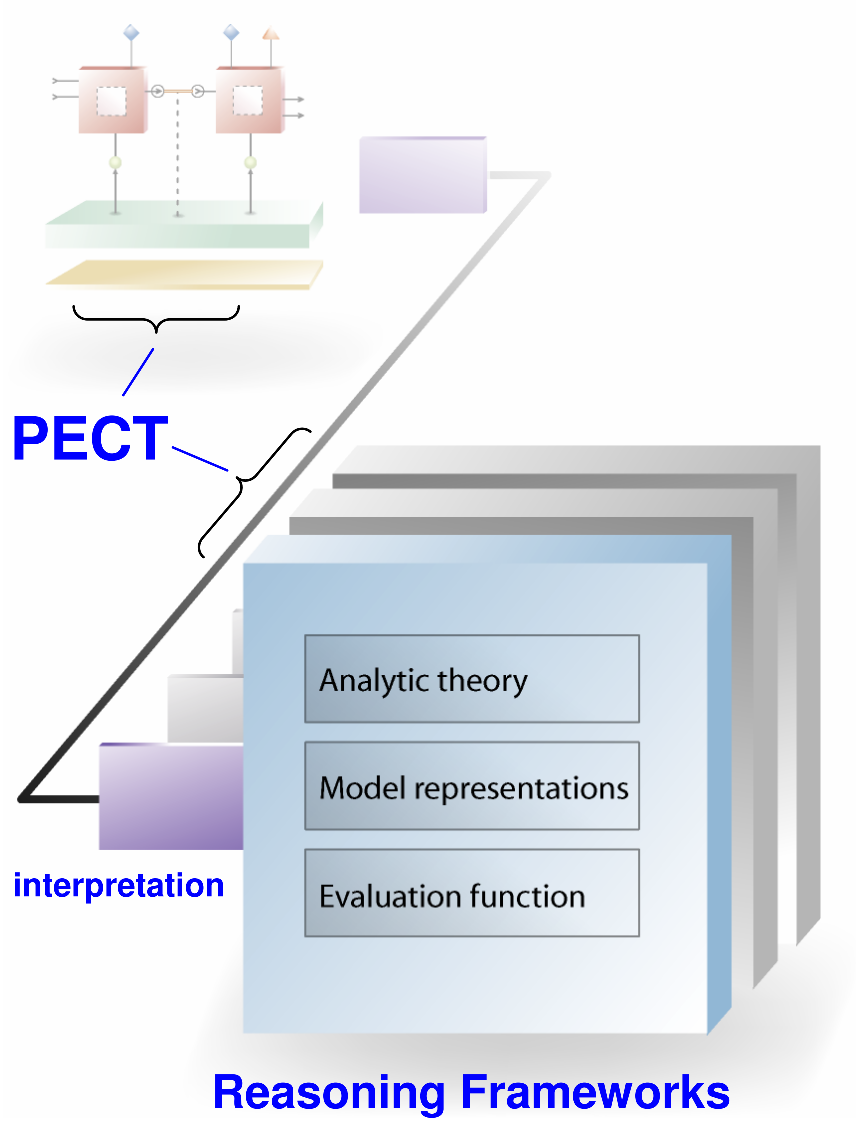
Interpretation can be formalized and automated

analysis model

imply valid analysis models here

Analysis

Packaging Predictability



Prediction-enabled component technology (PECT)

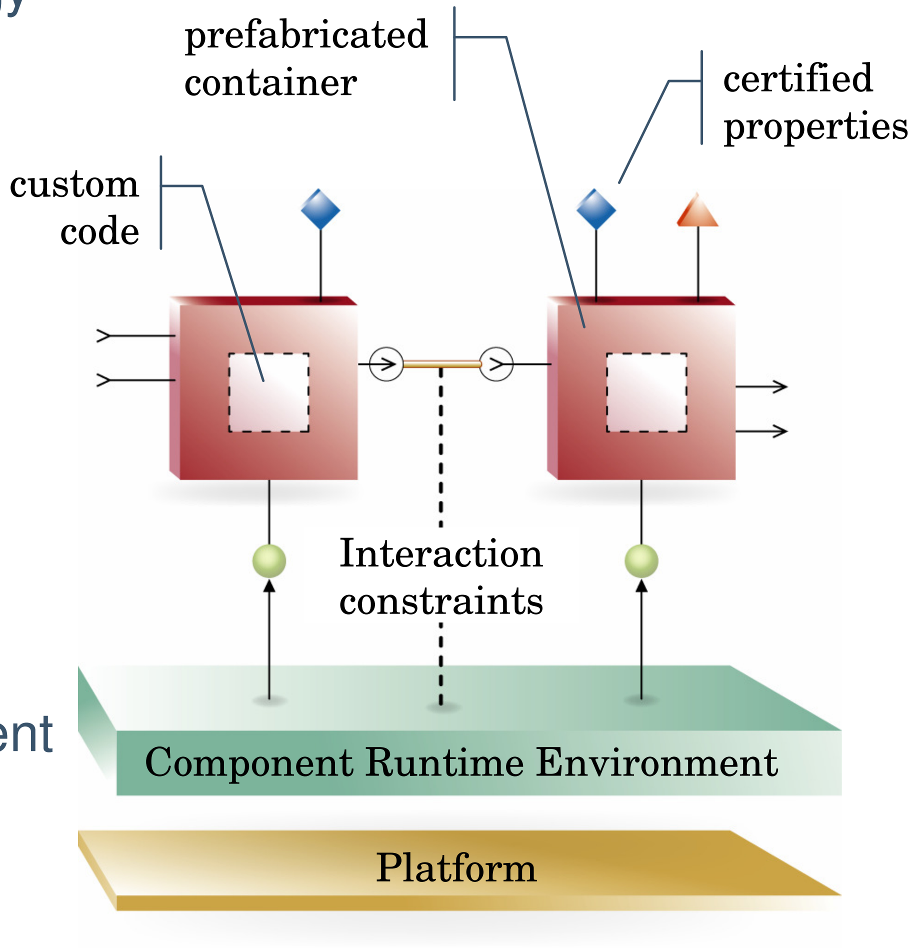
- reasoning frameworks make state-of-the-art analysis technology accessible
- component technology is a carrier for analysis-specific design constraints
- interpretation checks for well-formed-ness to constraints

Pin Component Technology

Lightweight component technology designed to support predictability

Features:

- strict encapsulation
  - -implements the container idiom
  - -interaction only through source and sink pins
- pure composition model
  - -declarative composition without “glue” code
  - -no hidden interactions
- component runtime environment
  - -provides system services
  - -enforces component interaction policies
  - -provides a portability layer

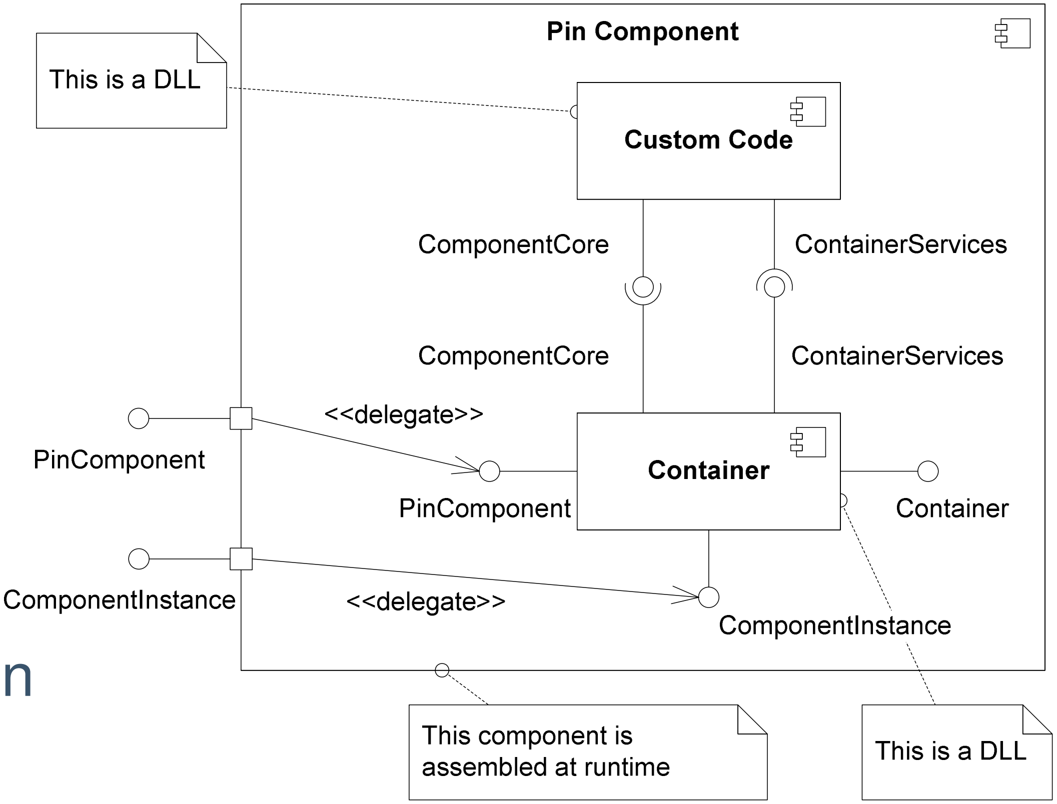


Components and Containers

Custom code and containers are independently deployable and dynamically bound at runtime.

Containers

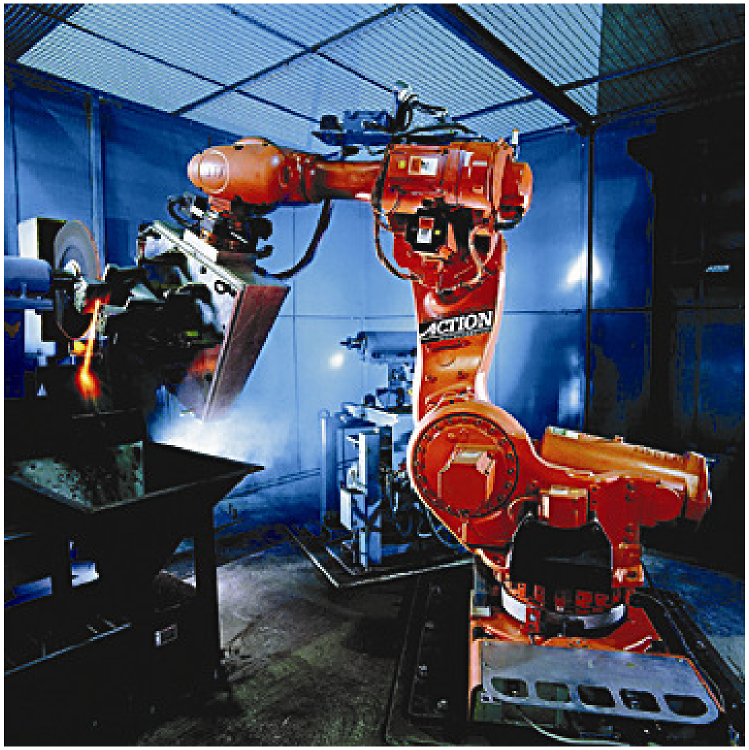
- create threads as needed
- mediate interactions between the custom code and the environment
- dispatch message handlers
- provide communication services



Example: Industrial Robot Controller

Given: hard real-time robot controller

- hard periodic deadlines
- predictable using GRMA*



Want to: allow third-party stochastic extensions

- give best service to the extension
- predict extension’s average latency
- *retain predictability* of controller

*GRMA: Generalized Rate Monotonic Analysis.

Solution: Sporadic Server

The sporadic server (SS) is a mechanism to schedule aperiodic tasks:

- reserves a budget of high priority execution time for aperiodics
- the budget is replenished one period after consumed
- if budget is exhausted, aperiodics can execute only in background priority

Benefits of the sporadic server:

- gives good quality of service to aperiodic tasks
- it is no more invasive than an equivalent periodic task, even in the face of a burst of arrivals

Prediction: λss Reasoning Framework

Reasoning framework to predict the average latency of aperiodic tasks

Main assumption: aperiodic tasks executes in a sporadic server

Other assumptions:

- sporadic server executes at highest priority
- replenishment policy
- execution time = budget (for now)
- inter-arrival times follow an exponential distribution
- fixed priority scheduler

Constraint Enforcement Options

Require component developer to adhere to the constraints

- sporadic server implementation not trivial
- replenishment policy inconsistencies
- lack of flexibility

Provide sporadic server library to the component developer

- rely on the developer to use it correctly
- policy can be circumvented
- lack of flexibility

Use a container to enforce the constraints

- component developer oblivious of constraints
- component can be used in different settings

The Sporadic Server Container

Takes advantage of the visibility provided by the container idiom.

- Component instance life cycle
  - creates sporadic server manager thread
  - registers aperiodic task with the sporadic server manager
  - shuts the sporadic server manager down when done

- Message handling and dispatching loop
  - arms the sporadic server before waiting for message
  - sends request to the sporadic server manager before dispatching the handler

Component Loading Example

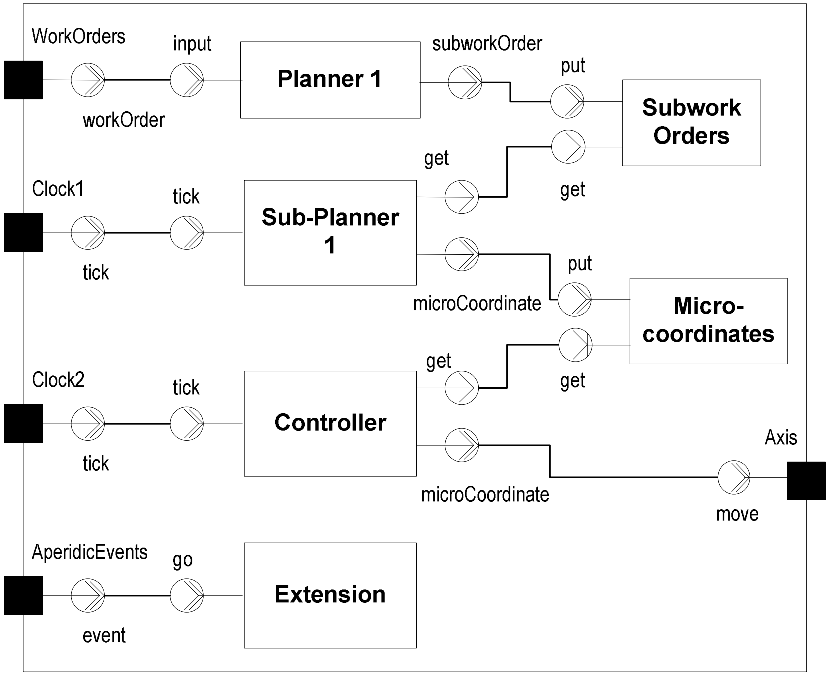
```
container =
       loadContainer(“standard.dll");

ssContainer =
       extendContainer(“ss.dll",
                      container);

extensionInSs =
       loadComponent(“extension.dll",
                     ssContainer);
```

standard container provides all basic services and policies

simple inheritance mechanism

dynamic binding of container and custom code

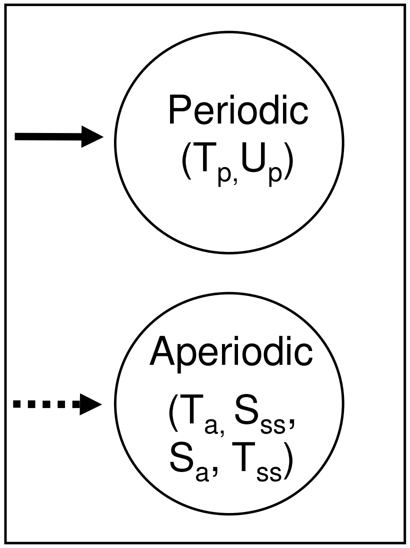
Prediction: Automated Interpretation



Component Assembly

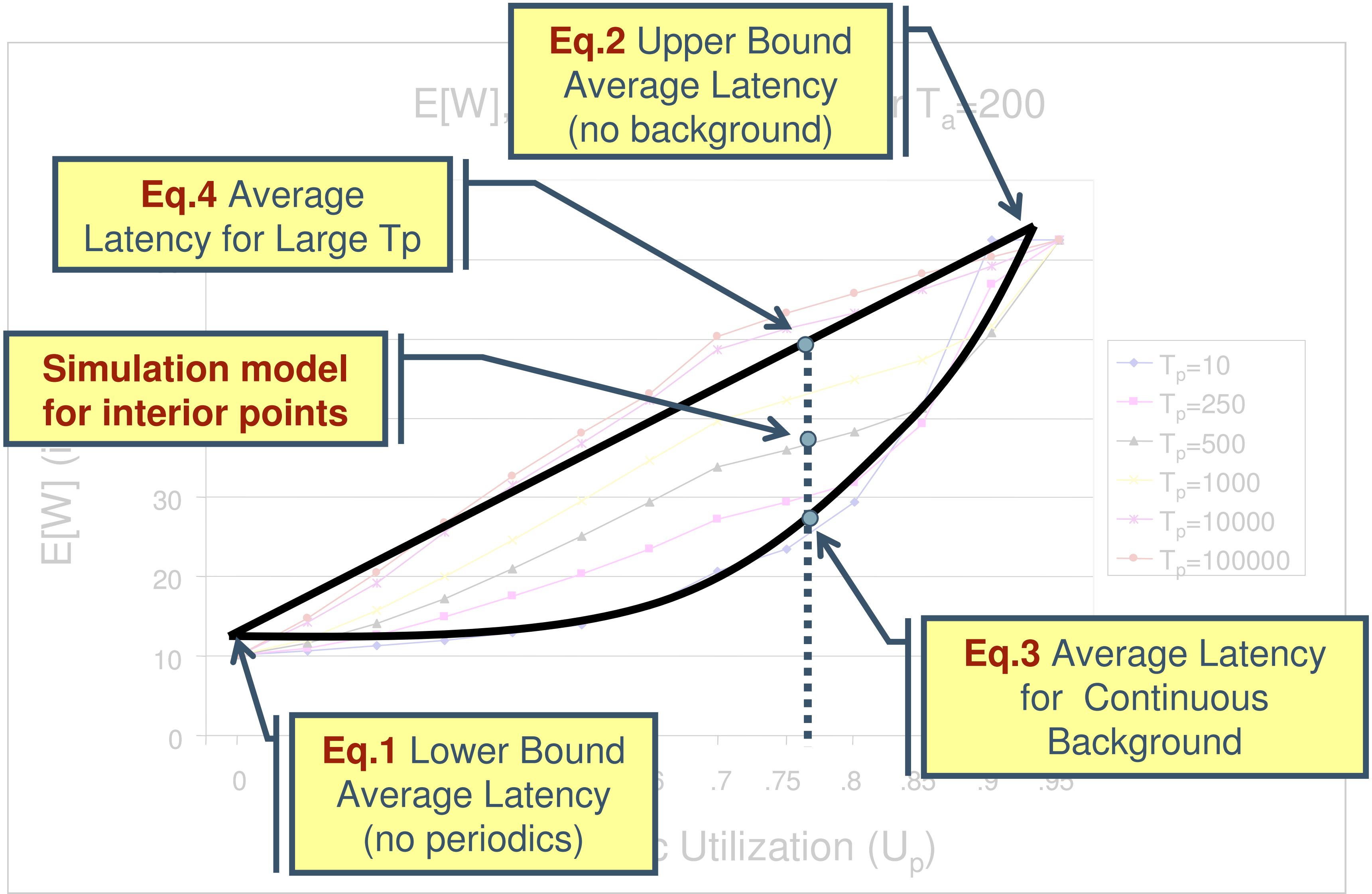
Construction and Composition Language

Interpretation



Performance Model

Prediction: λ_{ss} Evaluation Function



Eq.2 Upper Bound Average Latency (no background)

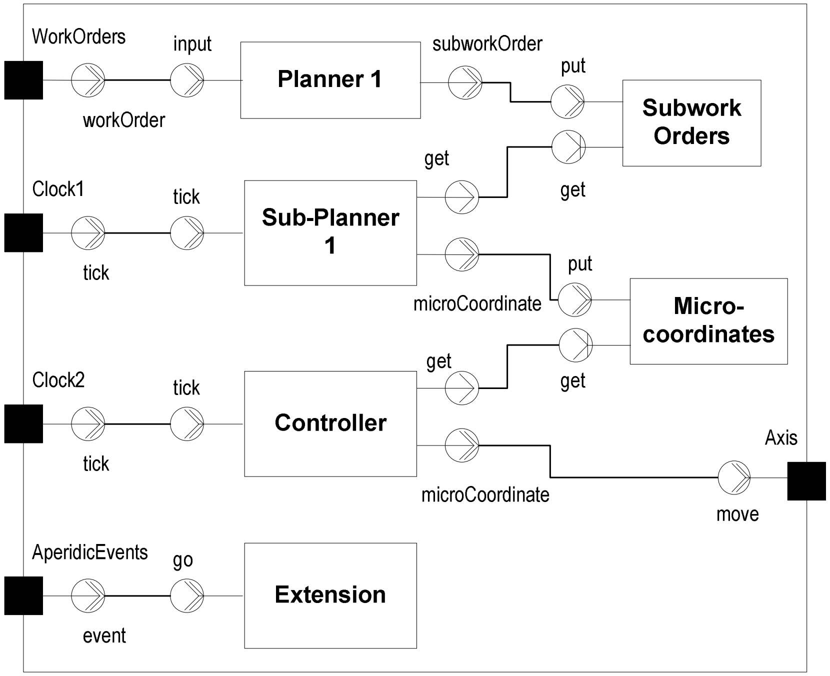
Eq.4 Average Latency for Large Tp

Simulation model for interior points

Eq.3 Average Latency for Continuous Background

Eq.1 Lower Bound Average Latency (no periodics)

Code Generation from Model



Component Assembly

Construction and
Composition Language

Code Generation

Executable Implementation

Summary

Smart constraints are used to ensure predictable behavior.

- constraints correspond to assumptions of analysis theories
- construct assemblies that are predictable

Containers can be used to enforce smart constraints.

- relieve component developers from adhering to the constraints
- reuse components in different settings (different policies, different assumptions)

Reasoning frameworks package engineering knowledge to analyze the behavior of systems.

- package engineering competence as a reusable asset
- enable nonexperts to predict critical runtime qualities

For More Information

Gabriel Moreno
Member of Technical Staff
Software Engineering Institute
Carnegie Mellon University
Email: gmoreno@sei.cmu.edu

Predictable Assembly from Certifiable Components (PACC) Initiative
http://www.sei.cmu.edu/pacc