Guided Architecture Trade Space Exploration

Fusing Model Based Engineering and Design by Shopping

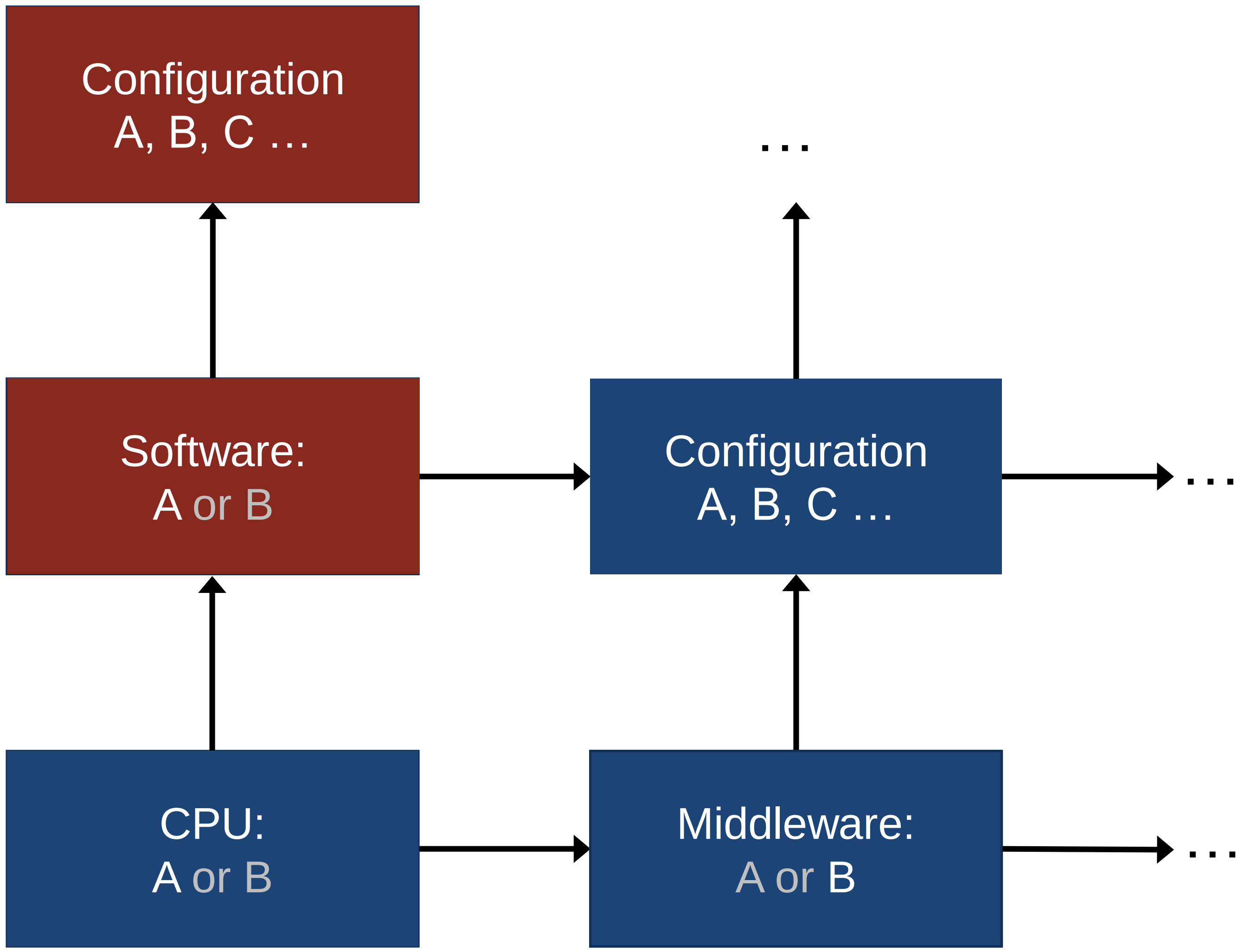
Sam Procter (sprocter@sei.cmu.edu)

Lutz Wrage

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Document Markings

More components, more complexity



What matters:

- Satisfy functional properties
- Achieve non-functional objectives

But that’s not actually how it all works.

System designers rely on their *expertise* and *intuition* instead

- Model-Based System Engineering (MBSE) supports that intuition, but has some drawbacks at large scale.
- Design Space Exploration works well at scale, but has some usability issues and rarely uses multipurpose system models

So, we created and evaluated the *Guided Architecture Trade Space Explorer*, which supports designers’ intuition by integrating:

- A standardized MBSE language and tool
- An established DSE tool

Outline

A Wheel-Braking System

Designing by Shopping

Guided Architecture Trade Space Exploration

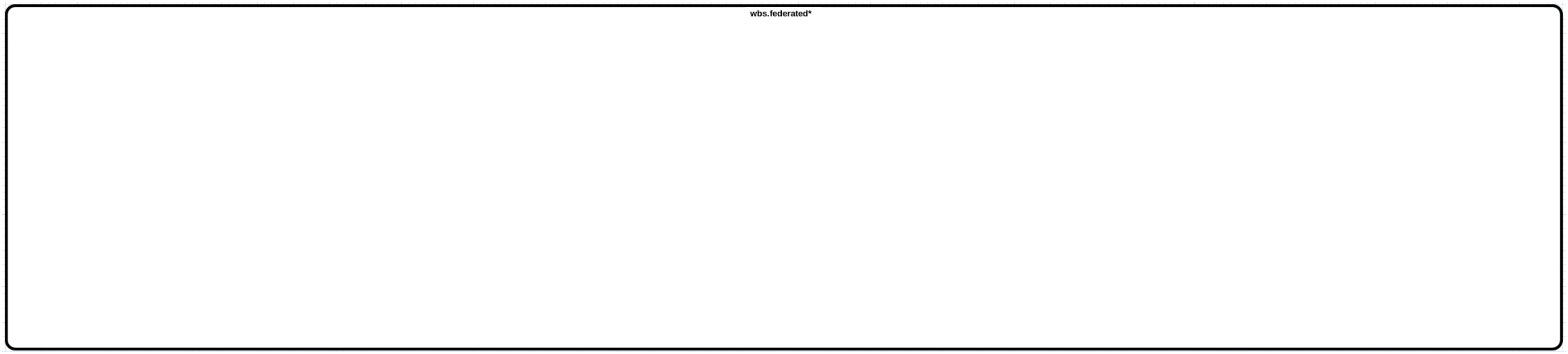
Outline

A Wheel-Braking System

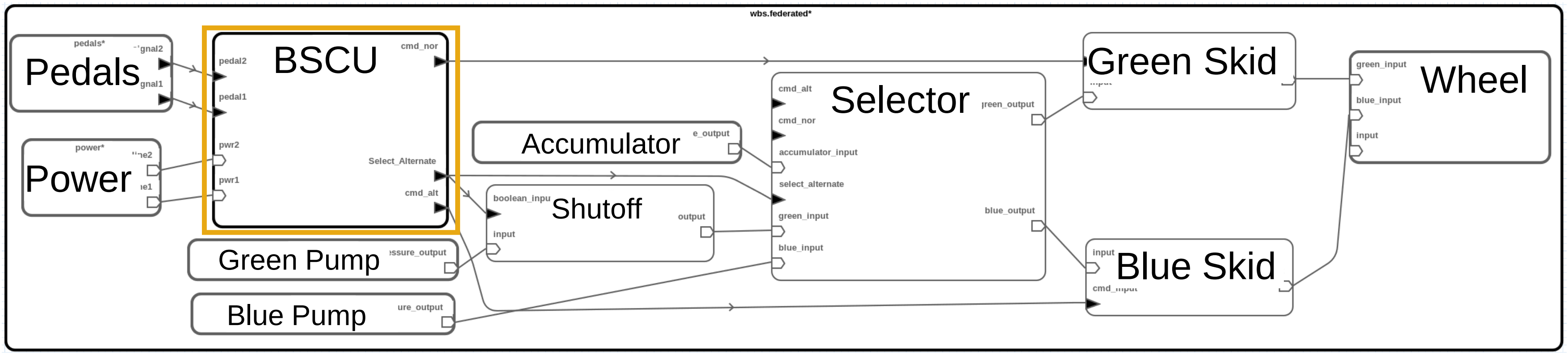
Designing by Shopping

Guided Architecture Trade Space Exploration

The wheel brake system



The wheel brake system

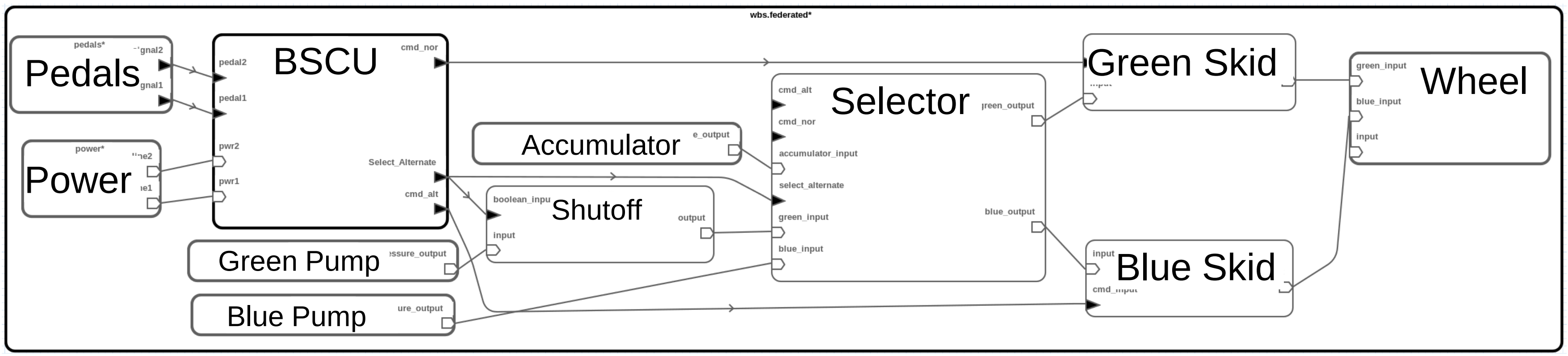


Two subsystems (command and monitor) + common platform

- Two monitor implementations, two command implementations
- Platform varies in power budget, wiring gauge, CPU architecture
  - Multiple CPUs must have the same architecture
  - Power required by CPUs must match platform provisions

… and that's just one component!

Architecture Analysis and Design Language



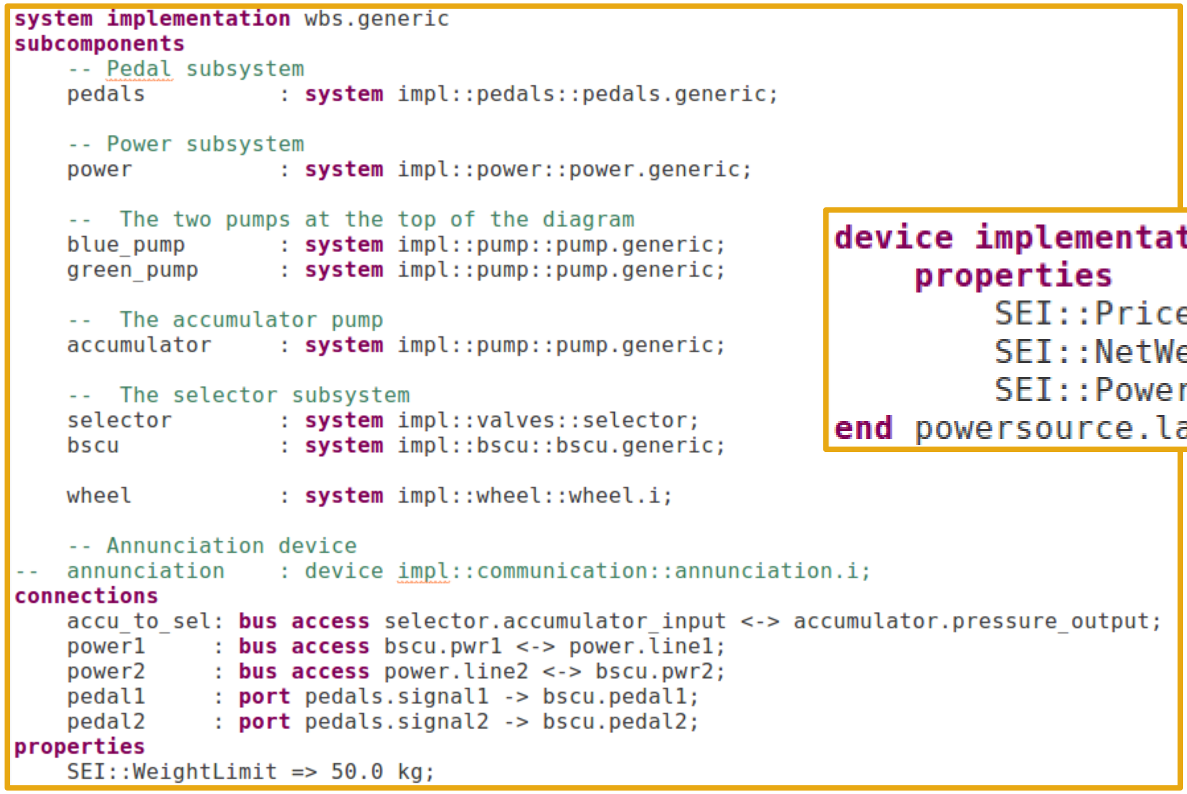
This is AADL's graphical syntax (textual syntax on… the next slide)

International standard (SAE AS5506C)

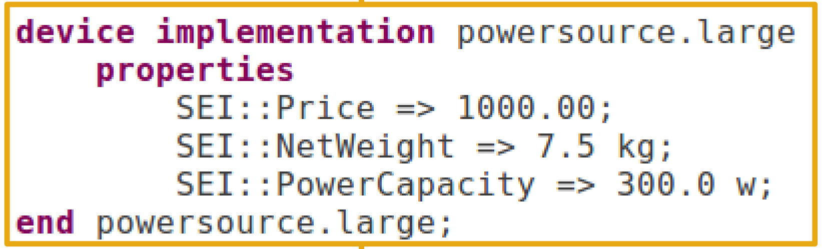
Used in academia, industry, government in the US, EU, China

https://aadl.info

Architecture Analysis and Design Language



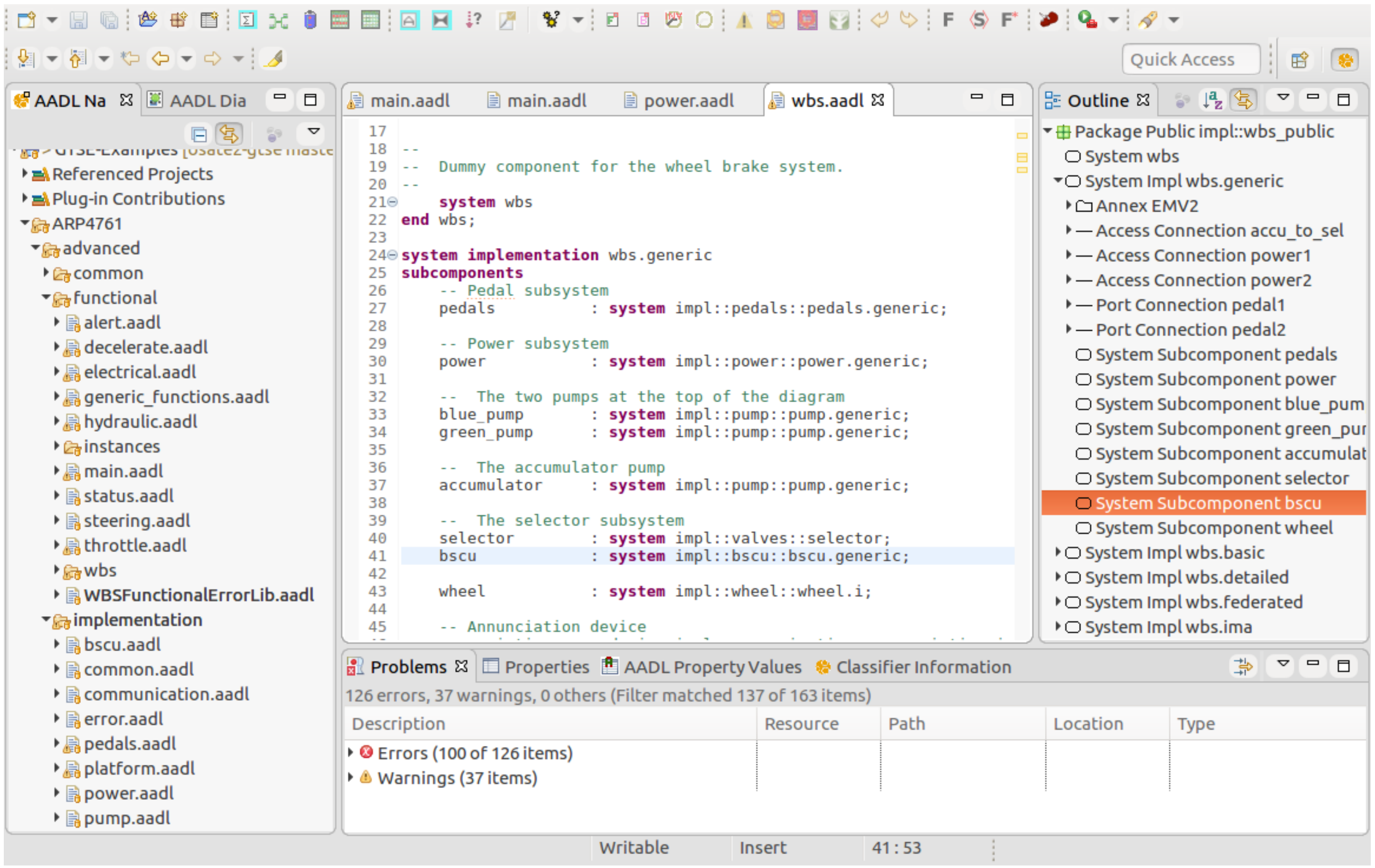
```
system implementation wbs.generic
subcomponents
    -- Pedal subsystem
    pedals           : system impl::pedals::pedals.generic;

    -- Power subsystem
    power            : system impl::power::power.generic;

    --  The two pumps at the top of the diagram
    blue_pump        : system impl::pump::pump.generic;
    green_pump       : system impl::pump::pump.generic;

    --  The accumulator pump
    accumulator      : system impl::pump::pump.generic;

    --  The selector subsystem
    selector         : system impl::valves::selector;
    bscu             : system impl::bscu::bscu.generic;

    wheel            : system impl::wheel::wheel.i;

    -- Annunciation device
--  annunciation     : device impl::communication::annunciation.i;
connections
    accu_to_sel: bus access selector.accumulator_input <-> accumulator.pressure_output;
    power1     : bus access bscu.pwr1 <-> power.line1;
    power2     : bus access power.line2 <-> bscu.pwr2;
    pedal1     : port pedals.signal1 -> bscu.pedal1;
    pedal2     : port pedals.signal2 -> bscu.pedal2;
properties
    SEI::WeightLimit => 50.0 kg;
```



```
device implementation powersource.large
    properties
        SEI::Price => 1000.00;
        SEI::NetWeight => 7.5 kg;
        SEI::PowerCapacity => 300.0 w;
end powersource.large;
```

- Textual syntax is better for (potentially custom) properties / computer scientists
- Graphical syntax is better for structure / system engineers

Open Source Architecture Tool Environment



AADL Na | AADL Dia

- Referenced Projects
- Plug-in Contributions
- ARP4761
  - advanced
    - common
    - functional
      - alert.aadl
      - decelerate.aadl
      - electrical.aadl
      - generic_functions.aadl
      - hydraulic.aadl
      - instances
      - main.aadl
      - status.aadl
      - steering.aadl
      - throttle.aadl
      - wbs
      - WBSFunctionalErrorLib.aadl
    - implementation
      - bscu.aadl
      - common.aadl
      - communication.aadl
      - error.aadl
      - pedals.aadl
      - platform.aadl
      - power.aadl
      - pump.aadl

main.aadl | main.aadl | power.aadl | wbs.aadl

```
--
--  Dummy component for the wheel brake system.
--
    system wbs
end wbs;

system implementation wbs.generic
subcomponents
    -- Pedal subsystem
    pedals          : system impl::pedals::pedals.generic;

    -- Power subsystem
    power           : system impl::power::power.generic;

    --  The two pumps at the top of the diagram
    blue_pump       : system impl::pump::pump.generic;
    green_pump      : system impl::pump::pump.generic;

    --  The accumulator pump
    accumulator     : system impl::pump::pump.generic;

    --  The selector subsystem
    selector        : system impl::valves::selector;
    bscu            : system impl::bscu::bscu.generic;

    wheel           : system impl::wheel::wheel.i;

    -- Annunciation device
```

Outline

- Package Public impl::wbs_public
  - System wbs
  - System Impl wbs.generic
    - Annex EMV2
    - Access Connection accu_to_sel
    - Access Connection power1
    - Access Connection power2
    - Port Connection pedal1
    - Port Connection pedal2
    - System Subcomponent pedals
    - System Subcomponent power
    - System Subcomponent blue_pum
    - System Subcomponent green_pur
    - System Subcomponent accumulat
    - System Subcomponent selector
    - System Subcomponent bscu
    - System Subcomponent wheel
  - System Impl wbs.basic
  - System Impl wbs.detailed
  - System Impl wbs.federated
  - System Impl wbs.ima

Problems | Properties | AADL Property Values | Classifier Information

126 errors, 37 warnings, 0 others (Filter matched 137 of 163 items)

Description	Resource	Path	Location	Type
Errors (100 of 126 items)				
Warnings (37 items)				

Writable | Insert | 41 : 53

OSATE is open source & SEI maintained

https://osate.org

OSATE is a system analysis toolbench

In addition to expected IDE functionality, OSATE supports:

- Latency analysis
- Power consumption / budgeting
- Scheduling analysis
- Much more (safety, security, etc.)

… more are being added by the SEI and external researchers.

Example Domain-Specific Plugin

```
private double calcBrakingPower(ComponentInstance ci) {
 double power = 0.0;
 /* Recurse into subcomponents */
 EList<ComponentInstance> cil = ci.getComponentInstances();
 for (ComponentInstance subi : cil) {
  power += calcBrakingPower(subi);
 }
 power += PropertyUtils.getRealValue(ci,
  GetProperties.lookupPropertyDefinition(ci,
  "DemoProperties", "BrakingPower"), 0.0);
 return power;
}
```

Outline

A Wheel-Braking System

Designing by Shopping

Guided Architecture Trade Space Exploration

Designing by Shopping (Balling 99)

What's wrong with optimization?

- "A priori articulation of preference" (Hwang and Masud) is hard.

How do we fix it?

- Visually display a range of options so users can intuitively understand tradeoffs
  - Display should be interactive
  - Options should be *pareto optimal*

Think of buying a shirt online…

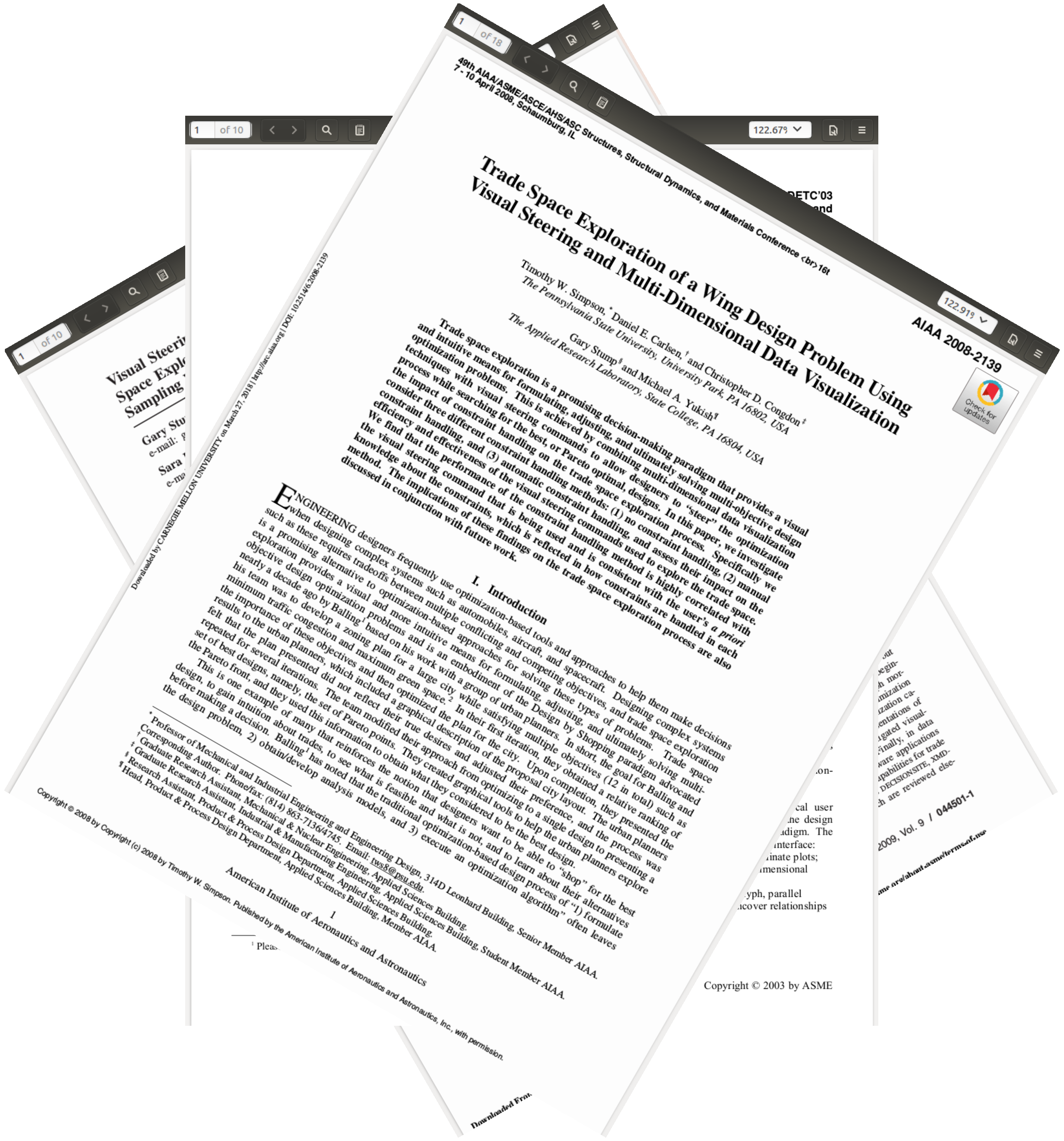
- It's hard to envision the perfect shirt without seeing any examples
  - And even if you do, what are the odds it exists?
- Look at some examples (yellow vs blue shirts, stripes vs dots) then refine your search

Penn State’s ARL Trade Space Visualizer

Java based software for design-by-shopping.

Includes both a range of evolutionary algorithms and a variety of visualizations.

Evaluated in aeronautics and aerospace domains.



48th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference
16t
7 - 10 April 2008, Schaumburg, IL

AIAA 2008-2139

Trade Space Exploration of a Wing Design Problem Using Visual Steering and Multi-Dimensional Data Visualization

Timothy W. Simpson,* Daniel E. Carlsen,† and Christopher D. Congdon‡
The Pennsylvania State University, University Park, PA 16802, USA

Gary Stump§ and Michael A. Yukish¶
The Applied Research Laboratory, State College, PA 16804, USA

Trade space exploration is a promising decision-making paradigm that provides a visual and intuitive means for formulating, adjusting, and ultimately solving multi-objective design optimization problems. This is achieved by combining multi-dimensional data visualization techniques with visual steering commands to allow designers to “steer” the optimization process while searching for the best, or Pareto optimal, designs. In this paper, we investigate the impact of constraint handling on the trade space exploration process. Specifically, we consider three different constraint handling methods: (1) no constraint handling, (2) manual constraint handling, and (3) automatic constraint handling, and assess their impact on the efficiency and effectiveness of the visual steering commands used to explore the trade space. We find that the performance of the constraint handling method is highly correlated with the visual steering command that is being used and is consistent with the user’s *a priori* knowledge about the constraints, which is reflected in how constraints are handled in each method. The implications of these findings on the trade space exploration process are also discussed in conjunction with future work.

I. Introduction

ENGINEERING designers frequently use optimization-based tools and approaches to help them make decisions when designing complex systems such as automobiles, aircraft, and spacecraft. Designing complex systems such as these requires tradeoffs between multiple conflicting and competing objectives, and trade space exploration is a promising alternative to optimization-based approaches for solving these types of problems. Trade space exploration provides a visual and more intuitive means for formulating, adjusting, and ultimately solving multi-objective design optimization problems and is an embodiment of the Design by Shopping paradigm advocated nearly a decade ago by Balling[1] based on his work with a group of urban planners. In short, the goal for Balling and his team was to develop a zoning plan for a large city while satisfying multiple objectives (12 in total) such as minimum traffic congestion and maximum green space.[2] In their first iteration, they obtained a relative ranking of the importance of these objectives and then optimized the plan for the city. Upon completion, they presented the results to the urban planners, which included a graphical description of the proposal city layout. The urban planners felt that the plan presented did not reflect their true desires and adjusted their preference and the process was repeated for several iterations. The team modified their approach from optimizing to a single design to presenting a set of best designs, namely, the set of Pareto points. They created graphical tools to help the urban planners explore the Pareto front, and they used this information to obtain what they considered to be the best design.

This is one example of many that reinforces the notion that designers want to be able to “shop” for the best design, to gain intuition about trades, to see what is feasible and what is not, and to learn about their alternatives before making a decision. Balling[1] has noted that the traditional optimization-based design process of “1) formulate the design problem, 2) obtain/develop analysis models, and 3) execute an optimization algorithm” often leaves

* Professor of Mechanical and Industrial Engineering and Engineering Design, 314D Leonhard Building, Senior Member AIAA. Corresponding Author. Phone/fax: (814) 863-7136/4745. Email: tws8@psu.edu.
† Graduate Research Assistant, Mechanical & Nuclear Engineering, Applied Sciences Building, Student Member AIAA.
‡ Graduate Research Assistant, Industrial & Manufacturing Engineering, Applied Sciences Building.
§ Research Assistant, Product & Process Design Department, Applied Sciences Building.
¶ Head, Product & Process Design Department, Applied Sciences Building, Member AIAA.

1
American Institute of Aeronautics and Astronautics

Copyright © 2008 by Copyright (c) 2008 by Timothy W. Simpson. Published by the American Institute of Aeronautics and Astronautics, Inc., with permission.

Downloaded by CARNEGIE MELLON UNIVERSITY on March 27, 2018 | http://arc.aiaa.org | DOI: 10.2514/6.2008-2139

Outline

A Wheel-Braking System

Designing by Shopping

Guided Architecture Trade Space Exploration

GATSE: What is it?

Plugin for OSATE, three main elements:

- Support for new *configuration* language (more detail coming)
- Modifies OSATE's instantiation and analysis logic
  - To make it headless
  - To support "skeleton" architectures
- Creates ATSV-connection artifacts

https://github.com/osate/osate2-gtse

A Configuration Language for AADL

An AADL Model

```
package P

    system S
    end S;

    system implementation S.i
        subcomponents
          sub: processor Intel;
    end S;

    processor Intel
    end Intel;

    processor implementation Intel.i3
    end Intel.i3;

    processor implementation Intel.i5
    end Intel.i5;

end P;
```

Assign a component implementation
and a property value

```
configuration C1 extends S.i {
    sub => Intel.i3;
    #SEI::Weight => 0.2 kg;
}
```

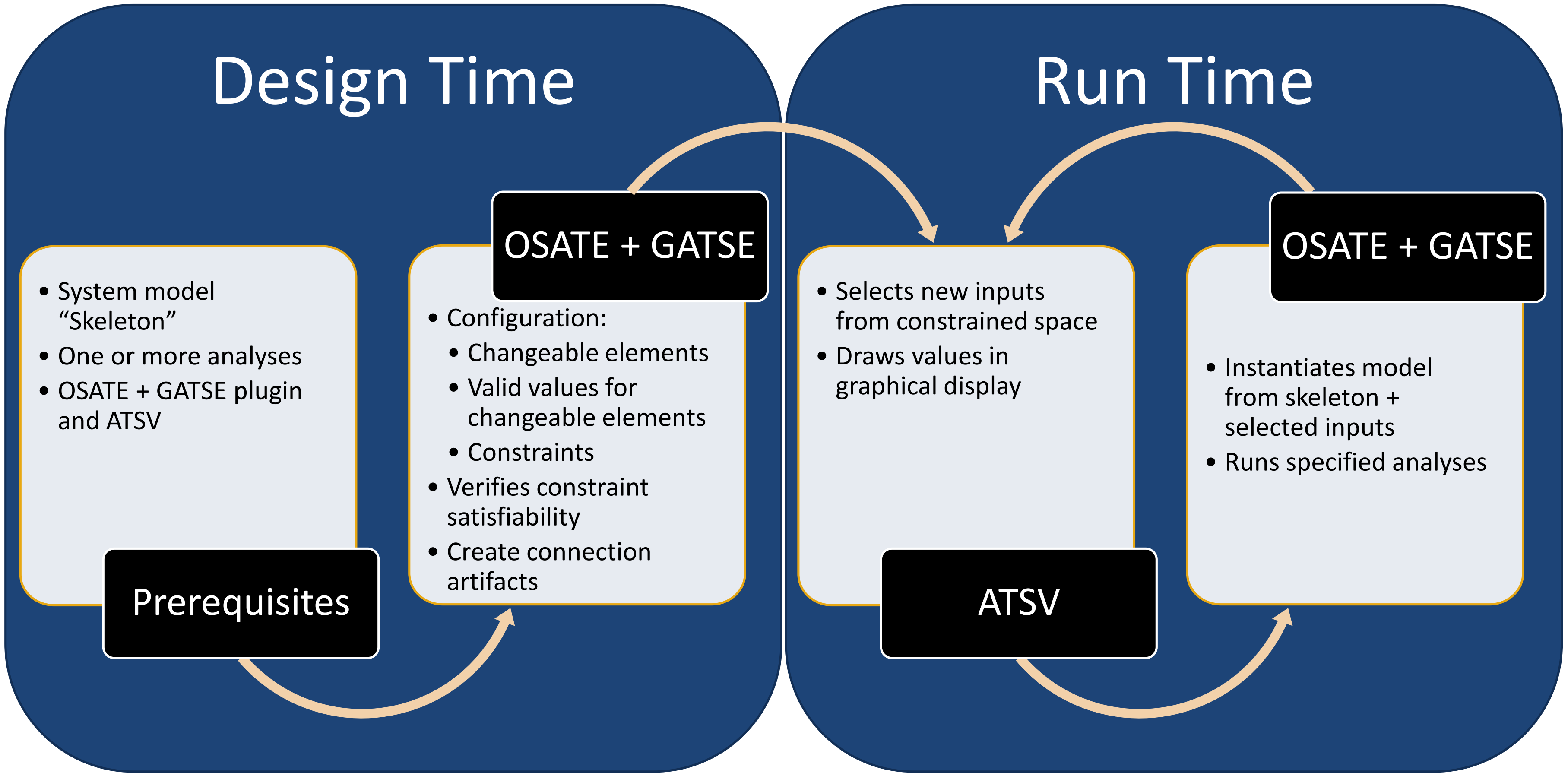
Extend a configuration and override an assignment
Assign a property in a nested configuration

```
configuration C2 extends S.i with C1 {
    sub => Intel.i5 {
        #SEI::MIPSCapacity => 1500 MIPS;
}   }
```

Parameterized configuration
with list of valid choices

```
configuration C3 (
    proc: processor Intel
             from (Intel.i3, Intel.i5)
) extends S.i {
    sub => proc;
    #SEI::MIPSCapacity => 1000MIPS;
}
```

GATSE: How do you use it?



Prerequisites:

- System model "Skeleton"
- One or more analyses
- OSATE + GATSE plugin and ATSV

OSATE + GATSE (Design Time):

- Configuration:
  - Changeable elements
  - Valid values for changeable elements
  - Constraints
- Verifies constraint satisfiability
- Create connection artifacts

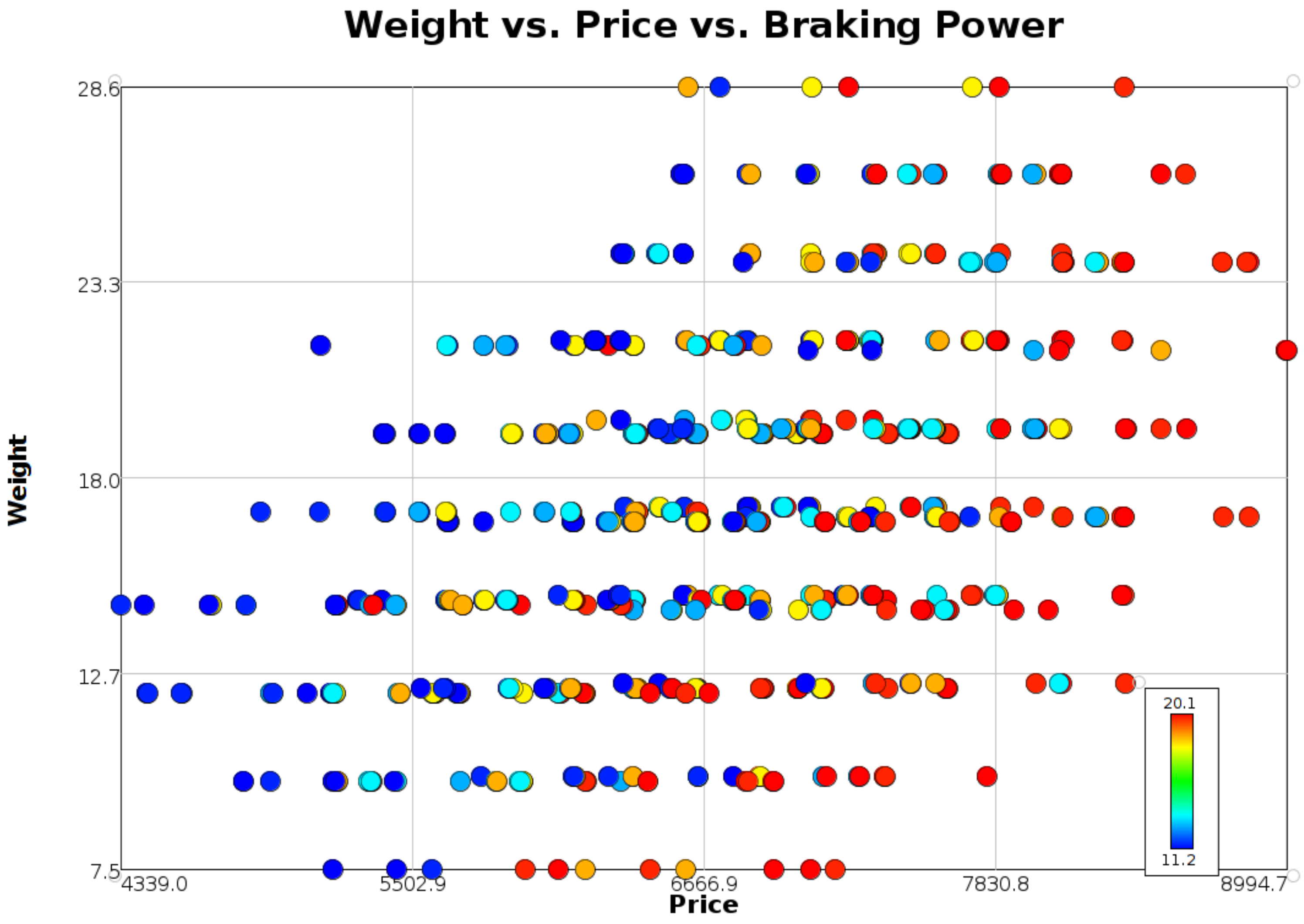
ATSV:

- Selects new inputs from constrained space
- Draws values in graphical display

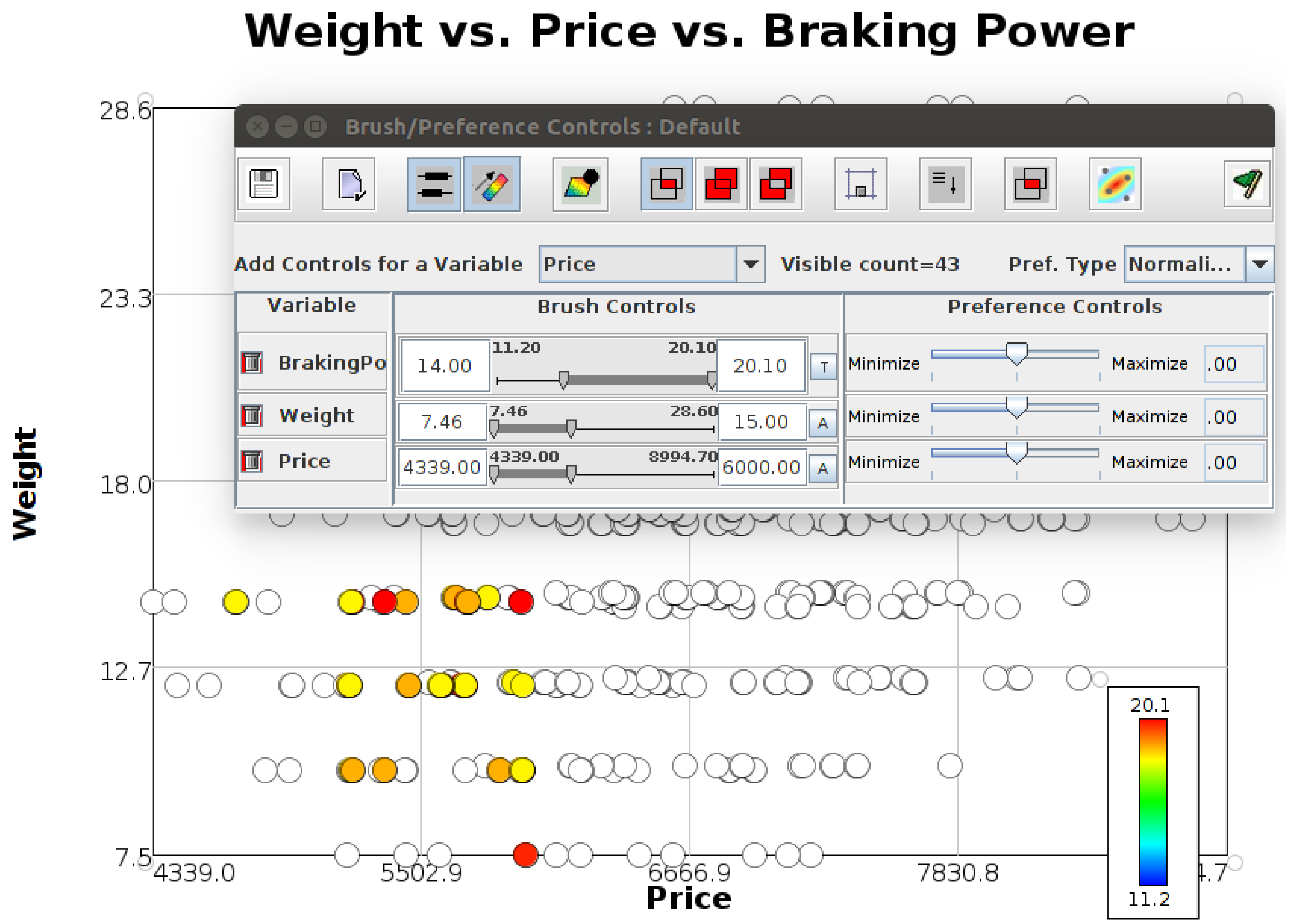
OSATE + GATSE (Run Time):

- Instantiates model from skeleton + selected inputs
- Runs specified analyses

GATSE (ATSV): In action – Viewing

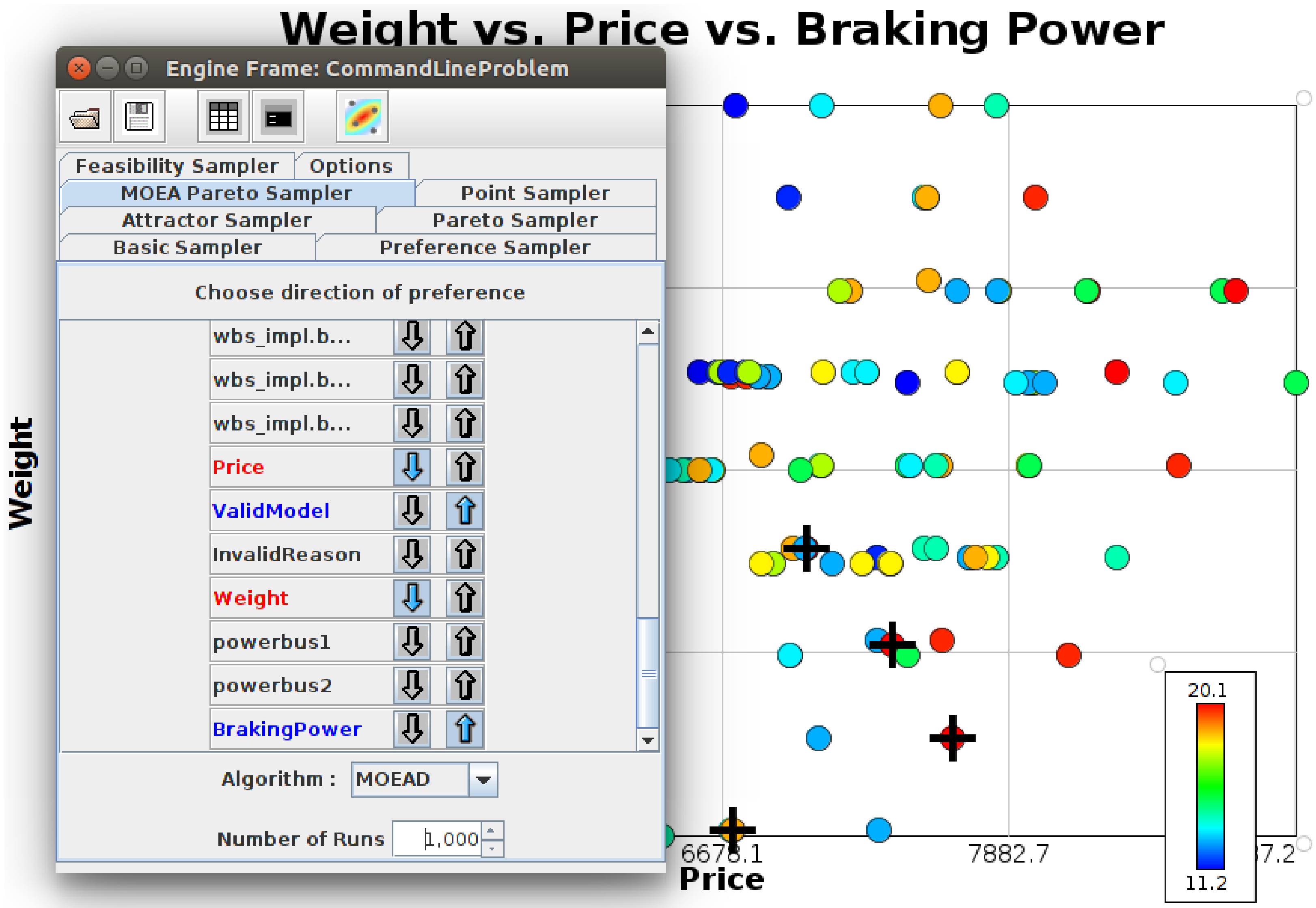


GATSE (ATSV): In action – Filtering



Variable	Brush Controls				Preference Controls	
BrakingPo	14.00	11.20	20.10	20.10	Minimize / Maximize	.00
Weight	7.46	7.46	28.60	15.00	Minimize / Maximize	.00
Price	4339.00	4339.00	8994.70	6000.00	Minimize / Maximize	.00

GATSE (ATSV): In action – Tailoring



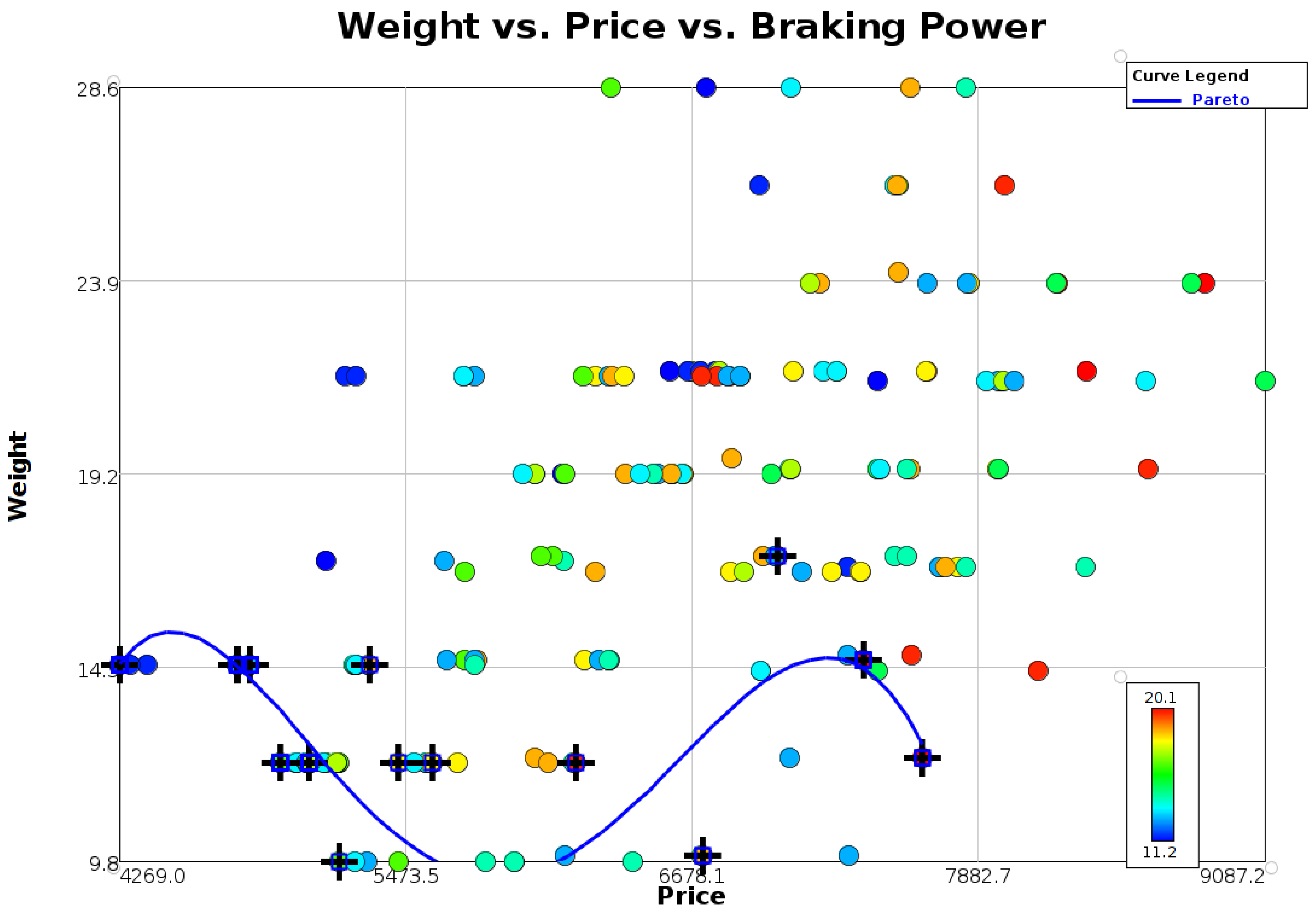
Choose direction of preference

	⇩	⇧
wbs_impl.b...	⇩	⇧
wbs_impl.b...	⇩	⇧
wbs_impl.b...	⇩	⇧
Price	[✓ ⇩]	⇧
ValidModel	⇩	[✓ ⇧]
InvalidReason	⇩	⇧
Weight	[✓ ⇩]	⇧
powerbus1	⇩	⇧
powerbus2	⇩	⇧
BrakingPower	⇩	[✓ ⇧]

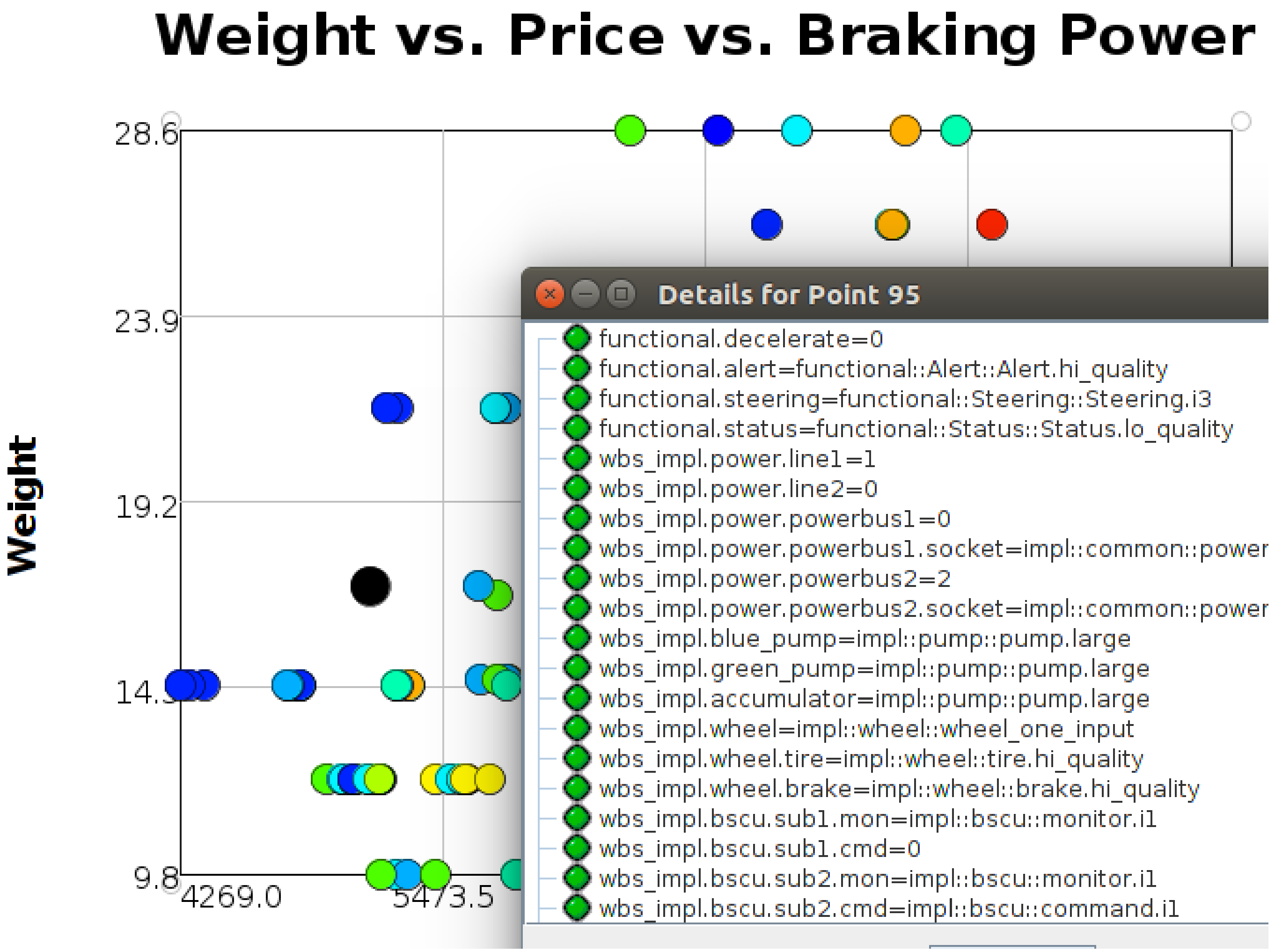
Algorithm : MOEAD

Number of Runs 1,000

GATSE (ATSV): In action – Pareto



GATSE (ATSV): In action – Detail



functional.decelerate=0
functional.alert=functional::Alert::Alert.hi_quality
functional.steering=functional::Steering::Steering.i3
functional.status=functional::Status::Status.lo_quality
wbs_impl.power.line1=1
wbs_impl.power.line2=0
wbs_impl.power.powerbus1=0
wbs_impl.power.powerbus1.socket=impl::common::power
wbs_impl.power.powerbus2=2
wbs_impl.power.powerbus2.socket=impl::common::power
wbs_impl.blue_pump=impl::pump::pump.large
wbs_impl.green_pump=impl::pump::pump.large
wbs_impl.accumulator=impl::pump::pump.large
wbs_impl.wheel=impl::wheel::wheel_one_input
wbs_impl.wheel.tire=impl::wheel::tire.hi_quality
wbs_impl.wheel.brake=impl::wheel::brake.hi_quality
wbs_impl.bscu.sub1.mon=impl::bscu::monitor.i1
wbs_impl.bscu.sub1.cmd=0
wbs_impl.bscu.sub2.mon=impl::bscu::monitor.i1
wbs_impl.bscu.sub2.cmd=impl::bscu::command.i1

Example Domain-Specific Plugin

```
public class BrakingPower extends AbstractAnalysis {

 @Override public void runAnalysis(SystemInstance instance,
  SystemOperationMode som, AnalysisErrorReporterManager errMgr,
  IProgressMonitor progressMonitor, Response resp) {

    resp.addVariable("BrakingPower", ATSVVariableType.FLOAT,
    String.valueOf(calcBrakingPower(instance)));
 }

 private double calcBrakingPower(ComponentInstance ci) {
  double power = 0.0;
  /* Recurse into subcomponents */
  EList<ComponentInstance> cil = ci.getComponentInstances();
  for (ComponentInstance subi : cil) {
   power += calcBrakingPower(subi);
  }
  power += PropertyUtils.getRealValue(ci,
   GetProperties.lookupPropertyDefinition(ci,
   "DemoProperties", "BrakingPower"), 0.0);
  return power;
 }
}
```

The GATSE Vision

AADL: Custom Properties

```
device implementation tire.hi_quality
 properties

  -- Built-in properties supporting
  --  cost and weight analyses
  SEI::Price => 1000.0;
  SEI::NetWeight => 3.5 kg;

  -- Custom property supporting domain-
  --  specific analysis, potentially
  --  derived from other analysis /
  --  modeling tools
  DemoProperties::BrakingPower => 10.0;
end tire.hi_quality;
```

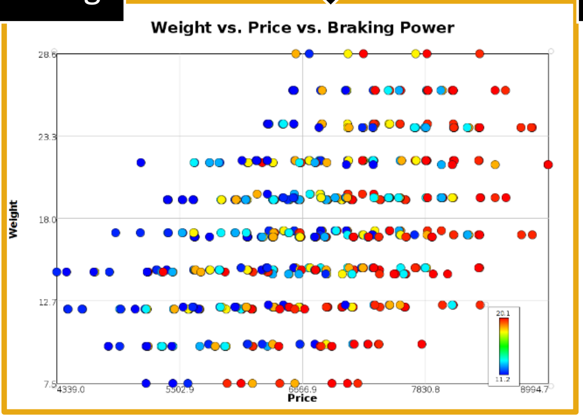
Used By

OSATE: Custom Analyses

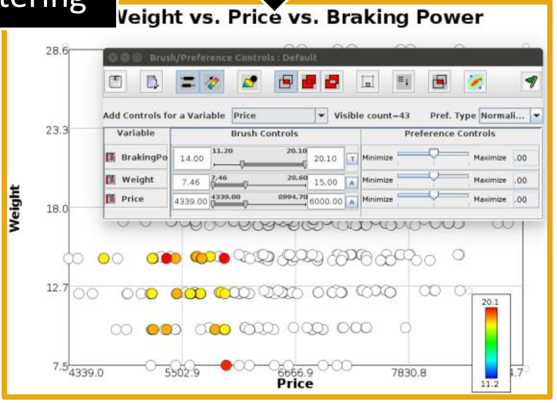
```
public class BrakingPower extends AbstractAnalysis {

 @Override public void runAnalysis(SystemInstance instance,
  SystemOperationMode som, AnalysisErrorReporterManager errMgr,
  IProgressMonitor progressMonitor, Response resp) {

    resp.addVariable("BrakingPower", ATSVVariableType.FLOAT,
    String.valueOf(calcBrakingPower(instance)));
 }

 private double calcBrakingPower(ComponentInstance ci) {
  double power = 0.0;
  /* Recurse into subcomponents */
  EList<ComponentInstance> cil = ci.getComponentInstances();
  for (ComponentInstance subi : cil) {
   power += calcBrakingPower(subi);
  }
  power += PropertyUtils.getRealValue(ci,
   GetProperties.lookupPropertyDefinition(ci,
   "DemoProperties", "BrakingPower"), 0.0);
  return power;
 }
}
```

Enables

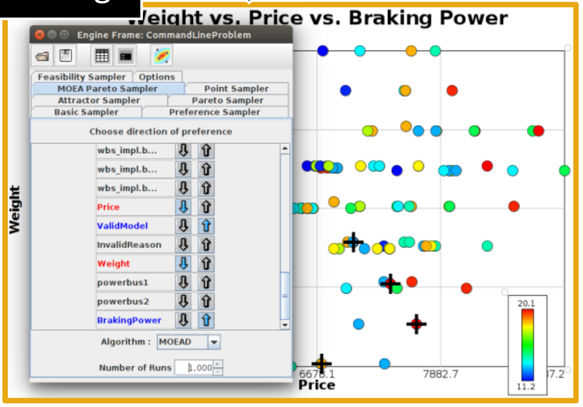
Viewing



Filtering



Tailoring



Future Work

Engineering

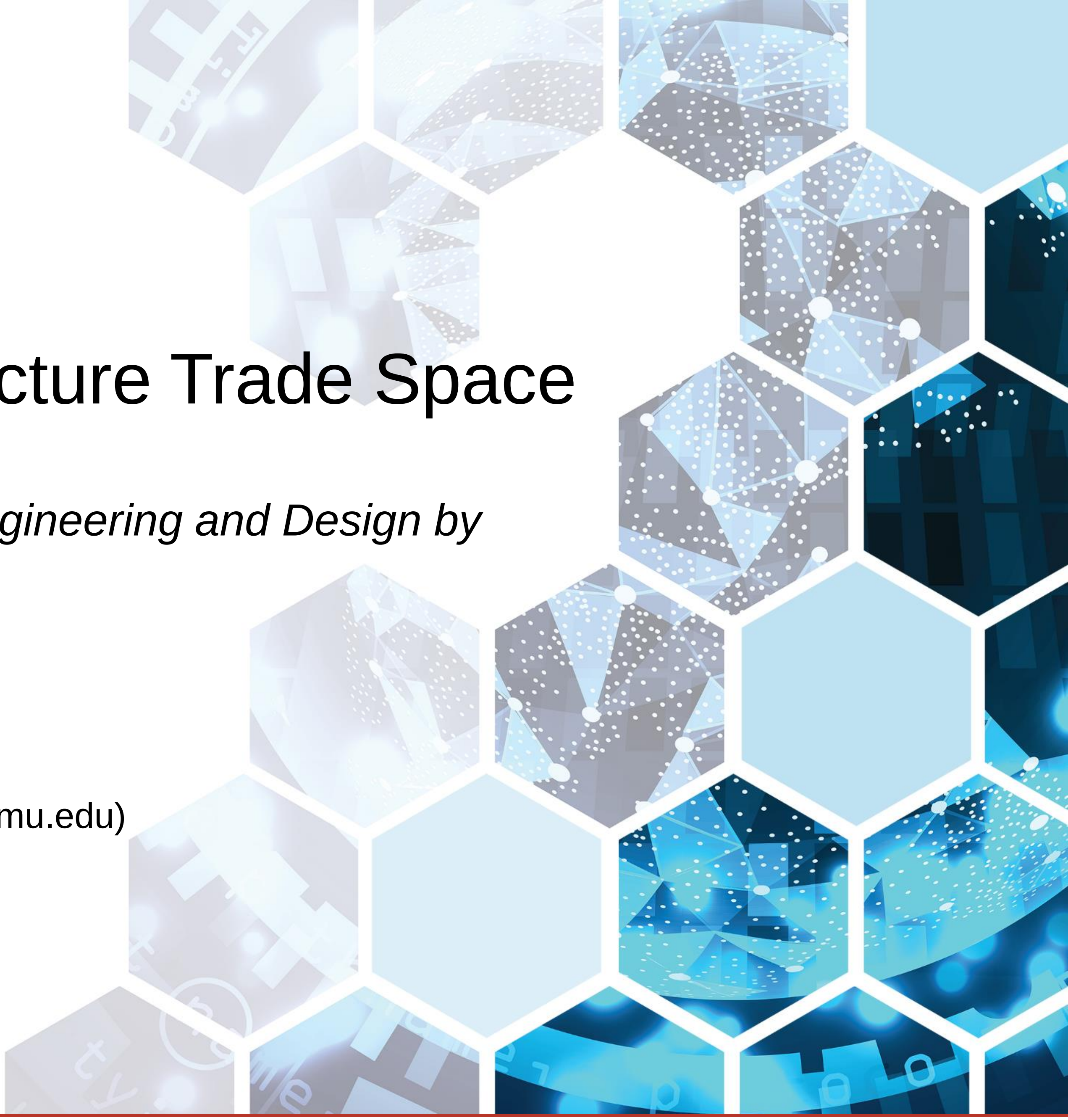
- Replace ATSV

Research

- Configuration language usability
- Novel quantification strategies

Evaluation

- Get this in the hands of a customer



Guided Architecture Trade Space Exploration

Fusing Model Based Engineering and Design by Shopping

Sam Procter (sprocter@sei.cmu.edu)

Lutz Wrage

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213