



System Configurations

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Peter Feiler
Jan 2007



Two Dimensions of System Families

Configuration via properties

Component interfaces, variants, and implementations



Configuration by Properties

- variation through alternative source code files
 - Source_Text property
- variation through conditional compilation in source code
 - Compilation parameters as properties
- variation of execution platform binding
 - Allowed_Binding property
- variation through calibration parameters
 - As properties or data components with value as property
- variation of data sets used in the execution of system components
 - Managed simulation runs



Conditional Compilation

property set CondComp is

TurboType : type enumeration (NoTurbo, Bosch, Delphi);

Turbo : inherit TurboType applies to (all);

Cylinders : inherit aadlinteger applies to (all);

end CondComp;

system implementation car.diesel_automatic_GTD

extends car.diesel_automatic

properties

CondComp::Turbo => Delphi ;

CondComp::Cylinders => 6;

end car.diesel_automatic_GTD;



System Parameters

Property set SystemParameters is

Turbo constant aadlboolean => false;

ThrottleSetting: inherit aadlreal => 0.576 applies to (all);

Speed: type units (kph);

TopSpeed constant aadlinteger SystemParameters::speed => 250 kph;

TireType: type enumeration (Touring, Winter, Offroad, Highspeed);

Tires constant SystemParameters::TireType => Touring;

end SystemParameters;



Management of Configuration Parameters

system implementation carSystem.

DualProcessorDeploymentConfig2

extends carSystem. DualProcessorCondCompConfig1

properties

Actual_Processor_Binding =>

reference carECU.ProcLeft **applies to**
carApp.PowerTrain.ABS;

Actual_Processor_Binding =>

reference carECU.ProcRight **applies to**
carApp.PowerTrain.ETC;

end carSystem. DualProcessorDeploymentConfig2;



Configurations in Packages

```
package car::configuration2
system carSystem extends car::Baseline::carSystem
end carSystem;
system implementation carSystem. DualProcessorDeploymentConfig2
  extends car::Baseline::carSystem. DualProcessor
properties
  CondComp::CondCompParameter => ( turbo, ABS );
  Actual_Processor_Binding =>
    reference carECU.ProcLeft applies to carApp.PowerTrain.ABS;
  Actual_Processor_Binding =>
    reference carECU.ProcRight applies to carApp.PowerTrain.ETC;
end carSystem. DualProcessorDeploymentConfig2;
end car::configuration2;
```



Instance Models and Configurations

XML representation supports

- Instance model separate from declarative model
- Sets of property values separate from instance model



Variation in Hierarchy and Topology

Achieved through selection of classifier of subcomponent

Refinement of incomplete subcomponent declaration

Classifier substitution

Classifier parameterization through prototype

Component category refinement



Use of Component Classifiers

Component types represent interaction interface

Component type extensions to represent variation in interface

Multiple component implementations as component variants

Component implementations as variations of realizations

Component implementation extensions as variations in properties



A Common System Architecture

```
system car
end car;
system implementation car.common
subcomponents
  PowerTrain: system power_train;
  ExhaustSystem: system exhaust_system;
connections
  bus access ExhaustSystem.exhaustManifold ->
  PowerTrain.exhaustManifold;
end car.common;
system power_train
features
  exhaustManifold: requires bus access
  Manifold;
end power_train;
```

```
system implementation
power_train.singleengine
subcomponents
  ETC: system ThrottleController;
  ABS: system AntilockBrakingSystem;
  CruiseControl: system CruiseControl;
  Transmission: system Transmission;
  PowerPlant: system Engine;
end power_train.singleengine;
system implementation power_train.twoengine
extends power_train.singleengine
subcomponents
  AlternatePowerPlant: system Engine;
end power_train.twoengine;
```



Specific Configuration

system implementation power_train.diesel

extends power_train.singleengine

subcomponents

 ETC: **refined to system** ThrottleController.bosch;

 ABS: **refined to system** AntilockBrakingSystem.bosch;

 CruiseControl: **refined to system** CruiseControl.delphi;

 Transmission: **refined to system** Transmission.automatic;

 PowerPlant: **refined to device** Engine.Diesel;

end power_train.diesel;

system implementation car.diesel

extends car.common

subcomponents

 PowerTrain: **refined to system** power_train.diesel;

 ExhaustSystem: **refined to system** exhaust_system.sporty;

end car.diesel;



Conceptual Reference Architecture

component car

end car;

component implementation car.generic

prototypes

power_train: component powertrain;

exhaust_system: component exhaustsystem;

subcomponents

PowerTrain: prototype power_train;

ExhaustSystem: prototype exhaust_system;

end car.generic;



Conceptual Reference Architecture

component powertrain

features

 exhaustoutput: requires bus access Manifold;

end powertrain;

component exhaust_system

features

 exhaustManifold: provides bus access Mainfold;

end exhaust_system;

component implementation exhaust_system.spoty

end exhaust_system.spoty;



Refinement to Runtime Reference Architecture

```
system carRT extends car
end carRT;
-- prototypes bound to actuals that are of the process category
system implementation carRT.impl
  extends car.generic (
    power_train => process powertrain;
    exhaust_system => process exhaust_system; );
end carRT.impl;
-- prototypes restricted to the process category
system implementation carRT.impl
  extends car.generic
  prototypes
    power_train : refined to process powertrain;
    exhaust_system : process exhaust_system;
end carRT.impl;
```



Runtime Reference Architecture

```
process powertrainProcess extends powertrain
end powertrainProcess ;
```

```
process exhaust_systemProcess extends exhaust_system
end exhaust_systemProcess;
```

```
system carRT extends car
end carRT;
```

```
system implementation carRT.impl
  extends car.generic (
    power_train => process powertrainProcess;
    exhaust_system => process exhaust_systemProcess; );
end carRT.impl
```



Specific Instance Based on Runtime Architecture

```
system Toyota extends car
```

```
-- bind the component category to be system
```

```
end Toyota;
```

```
process Toyota_powertrain extends powertrain
```

```
end Toyota_powertrain;
```

```
process implementation Toyota_powertrain.hybrid
```

```
extends powertrain.dualengine (
```

```
  TheEngine => device Engine.gasoline,
```

```
  TheAlternateEngine => device Engine.Electric,
```

```
  TheTransmission => thread group Transmission.Automatic,
```

```
  Throttle_Controller => thread group ThrottleController.Bosch,
```

```
  Antilock_Braking_System => thread group AntilockBrakingSystem.Bosch,
```

```
  Cruise_Control => thread group CruiseControl.Delphi );
```

```
end Toyota_powertrain.hybrid;
```



System Instance based on Subsystem Instance

```
system implementation Toyota.Prius
extends carRT.impl (
  power_train => process Toyota_powertrain.hybrid,
  exhaust_system => process exhaustsystem.sporty );
end Toyota.Prius;
```



Flattened Parameterization of Architecture

```
system implementation car.single
prototypes
  tc: system ThrottleController;
  abs: system AntilockBrakingSystem;
  cc: system CruiseControl;
  tm: system Transmission;
  en: device Engine;
  exhaust_system: system exhaustsystem;
subcomponents
  PowerTrain: system powertrain.singleengine(
    Throttle_Controller => prototype tc,
    Antilock_Braking_System => prototype abs,
    Cruise_Control => prototype cc,
    TheTransmission => prototype tm,
    TheEngine => prototype en);
  ExhaustSystem: prototype exhaust_system;
end car.single;
```



Flattened System Configuration Parameters

```
system implementation Toyota.Prius
extends car.dualengine (
  en => device Engine.gasoline,
  se => device Engine.Electric,
  tm => system Transmission.Automatic,
  tc => system ThrottleController.Bosch,
  abs => system AntilockBrakingSystem.Bosch,
  cc => system CruiseControl.Delphi,
  exhaust_system => system exhaustsystem.sporty );
end Toyota.Prius;
```



Instance from Conceptual Architecture

```
system implementation Toyota.Prius
extends car.generic (
  power_train => process Toyota_powertrain.hybrid,
  exhaust_system => process exhaustsystem.sporty );
end Toyota.Prius;

system implementation Toyota.Gasoline
extends car.generic (
  power_train => system powertrain.singleengine (
    TheEngine => device Engine.gasoline,
    TheTransmission => process Transmission.Automatic,
    Throttle_Controller => process ThrottleController.Bosch,
    Antilock_Braking_System => process AntilockBrakingSystem.Bosch,
    Cruise_Control => process CruiseControl.Delphi ),
  exhaust_system => process exhaustsystem.sporty );
end Toyota.Gasoline;
```

