Synthesizing Executable Simulations from Structural Models of Component-Based Systems

Andreas Schuster and Jonathan Sprinkle

October 6, 2009

Outline I

Outline II

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
  - Large parameter space for various scenarios, experiments
- Effort
  - Many domain experts, few programming experts
  - Individually proven, but not system tested, software
  - Body of regression tests necessary for evolutionary approach

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
  - Large parameter space for various scenarios, experiments
- Effort
  - Many domain experts, few programming experts
  - Individually proven, but not system tested, software
  - Body of regression tests necessary for evolutionary approach

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
  - Large parameter space for various scenarios, experiments
- Effort
  - Many domain experts, few programming experts
  - Individually proven, but not system tested, software
  - Body of regression tests necessary for evolutionary approach

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
  - Large parameter space for various scenarios, experiments
- Effort
  - Many domain experts, few programming experts
  - Individually proven, but not system tested, software
  - Body of regression tests necessary for evolutionary approach

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
  - Large parameter space for various scenarios, experiments
- Effort
  - Many domain experts, few programming experts
  - Individually proven, but not system tested, software
  - Body of regression tests necessary for evolutionary approach

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
  - Large parameter space for various scenarios, experiments
- Effort
  - Many domain experts, few programming experts
  - Individually proven, but not system tested, software
  - Body of regression tests necessary for evolutionary approach

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
  - Large parameter space for various scenarios, experiments
- Effort
  - Many domain experts, few programming experts
  - Individually proven, but not system tested, software
  - Body of regression tests necessary for evolutionary approach

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
  - Large parameter space for various scenarios, experiments
- Effort
  - Many domain experts, few programming experts
  - Individually proven, but not system tested, software
  - Body of regression tests necessary for evolutionary approach

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
  - Large parameter space for various scenarios, experiments
- Effort
  - Many domain experts, few programming experts
  - Individually proven, but not system tested, software
  - Body of regression tests necessary for evolutionary approach

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
  - Large parameter space for various scenarios, experiments
- Effort
  - Many domain experts, few programming experts
  - Individually proven, but not system tested, software
  - Body of regression tests necessary for evolutionary approach

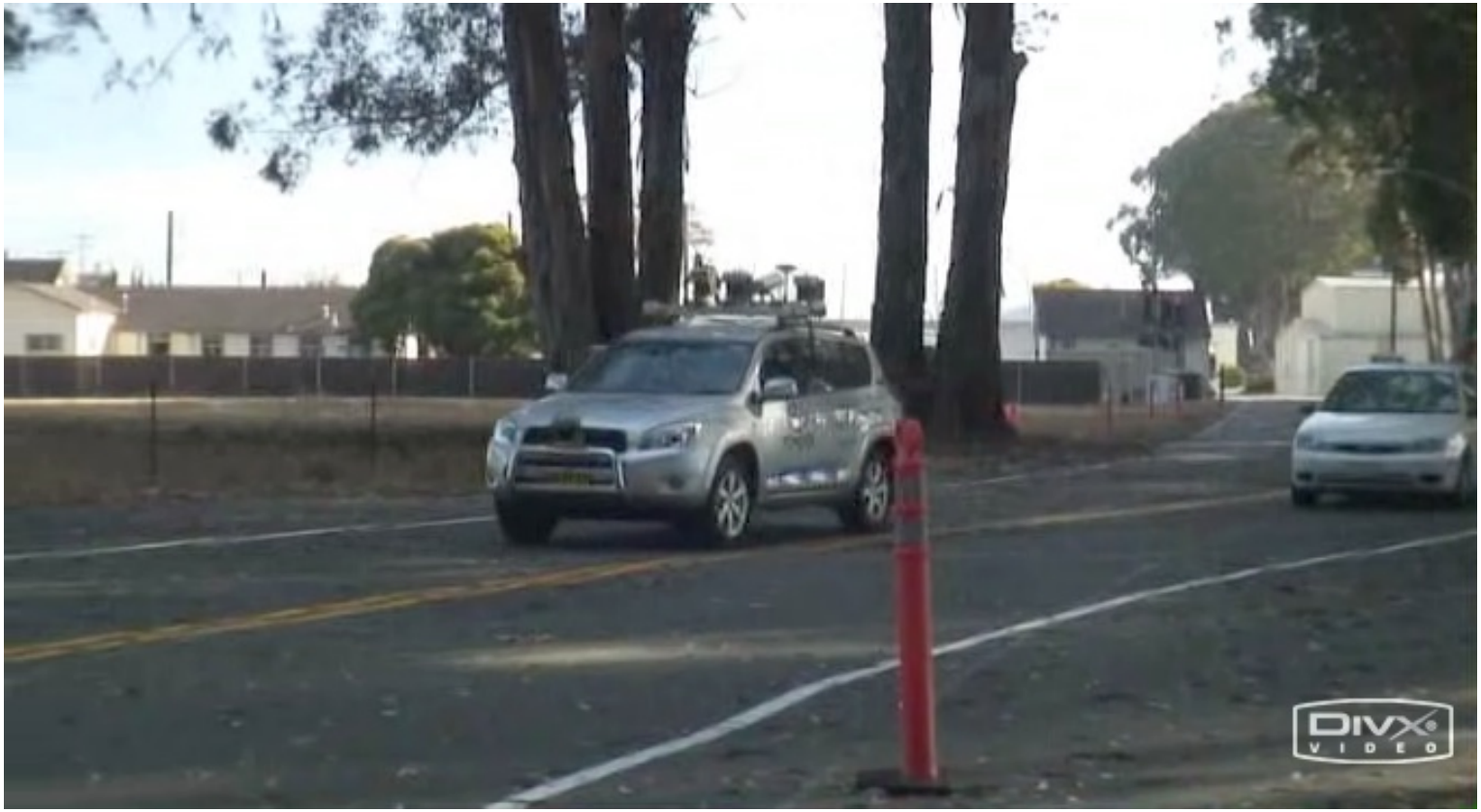
What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
  - Large parameter space for various scenarios, experiments
- Effort
  - Many domain experts, few programming experts
  - Individually proven, but not system tested, software
  - Body of regression tests necessary for evolutionary approach

What domain is this anyway?

- Autonomous Ground Vehicles
  - Complex, cyber-physical systems
  - Robotics, control, software, and information experts required
- Component-based middleware
  - Networked, real-time and soft real-time components
  - High bandwidth and low bandwidth components
  - Simple, component model
  - Large parameter space for various scenarios, experiments
- Effort
  - Many domain experts, few programming experts
  - Individually proven, but not system tested, software
  - Body of regression tests necessary for evolutionary approach

Sydney-Berkeley Driving Team



Link to online movie.

Domain Difficulties

- Robotics software in general
  - Individual task complexity and dynamic real-time nature [1]
  - Generalization of algorithms nontrivial
  - Large number of software contributors
  - Distributed, cross-platform computing environments are non-intuitive for domain experts
- Individual projects
  - Necessity of regression tests [2]
  - Simulation complexity increases dramatically when realistic simulations used

Domain Difficulties

- Robotics software in general
  - Individual task complexity and dynamic real-time nature [1]
  - Generalization of algorithms nontrivial
  - Large number of software contributors
  - Distributed, cross-platform computing environments are non-intuitive for domain experts
- Individual projects
  - Necessity of regression tests [2]
  - Simulation complexity increases dramatically when realistic simulations used

Domain Difficulties

- Robotics software in general
  - Individual task complexity and dynamic real-time nature [1]
  - Generalization of algorithms nontrivial
  - Large number of software contributors
  - Distributed, cross-platform computing environments are non-intuitive for domain experts
- Individual projects
  - Necessity of regression tests [2]
  - Simulation complexity increases dramatically when realistic simulations used

Domain Difficulties

- Robotics software in general
  - Individual task complexity and dynamic real-time nature [1]
  - Generalization of algorithms nontrivial
  - Large number of software contributors
  - Distributed, cross-platform computing environments are non-intuitive for domain experts
- Individual projects
  - Necessity of regression tests [2]
  - Simulation complexity increases dramatically when realistic simulations used

Domain Difficulties

- Robotics software in general
  - Individual task complexity and dynamic real-time nature [1]
  - Generalization of algorithms nontrivial
  - Large number of software contributors
  - Distributed, cross-platform computing environments are non-intuitive for domain experts
- Individual projects
  - Necessity of regression tests [2]
  - Simulation complexity increases dramatically when realistic simulations used

Domain Difficulties

- Robotics software in general
  - Individual task complexity and dynamic real-time nature [1]
  - Generalization of algorithms nontrivial
  - Large number of software contributors
  - Distributed, cross-platform computing environments are non-intuitive for domain experts
- Individual projects
  - Necessity of regression tests [2]
  - Simulation complexity increases dramatically when realistic simulations used

Domain Difficulties

- Robotics software in general
  - Individual task complexity and dynamic real-time nature [1]
  - Generalization of algorithms nontrivial
  - Large number of software contributors
  - Distributed, cross-platform computing environments are non-intuitive for domain experts
- Individual projects
  - Necessity of regression tests [2]
  - Simulation complexity increases dramatically when realistic simulations used

Domain Difficulties

- Robotics software in general
  - Individual task complexity and dynamic real-time nature [1]
  - Generalization of algorithms nontrivial
  - Large number of software contributors
  - Distributed, cross-platform computing environments are non-intuitive for domain experts
- Individual projects
  - Necessity of regression tests [2]
  - Simulation complexity increases dramatically when realistic simulations used

Middleware Solutions

Middleware and component-based technologies facilitate the abstraction of communication, and location of computation.

- CORBA
- ICE

Distributed Real-Time Embedded Systems

- Composition of such systems a subject of significant effort by Schmidt et al.
- The CoSMIC Toolsuite [3] can
  1. model and analyze DRE application functionality and QoS requirements
  2. synthesize CCM-specific deployment metadata for end-to-end QoS (static and dynamic)

Why not just CoSMIC?

In our domain, we focus on experiment synthesis, and *must incorporate* other robotics middleware solutions.

In our domain, we consider that (probably) the code is already written, and in fact the middleware experts already have the ability to produce experiments. What is missing?

- Novice users, experts in their domain, have no idea how to *run* the system
- Integration strategies are often *ad hoc*
- Changes to component integration nontrivial, and parameter values used difficult to recall from previous experiments.

Why not just CoSMIC?

In our domain, we focus on experiment synthesis, and *must incorporate* other robotics middleware solutions.

In our domain, we consider that (probably) the code is already written, and in fact the middleware experts already have the ability to produce experiments. What is missing?

- Novice users, experts in their domain, have no idea how to *run* the system
- Integration strategies are often *ad hoc*
- Changes to component integration nontrivial, and parameter values used difficult to recall from previous experiments.

Why not just CoSMIC?

In our domain, we focus on experiment synthesis, and *must incorporate* other robotics middleware solutions.

In our domain, we consider that (probably) the code is already written, and in fact the middleware experts already have the ability to produce experiments. What is missing?

- Novice users, experts in their domain, have no idea how to *run* the system
- Integration strategies are often *ad hoc*
- Changes to component integration nontrivial, and parameter values used difficult to recall from previous experiments.

Robotics Middleware Solutions

Various robotics-specific middleware already attempt to aid robotics researchers in writing domain-independent, application independent software.

- Gearbox [1]
- Player/Stage [4]
- Orocos [5]
- Microsoft Robotics Studio [6]

In order to use software in more than one domain, it must be configurable, and parameterizable. This has the tradeoff that it becomes much more difficult to understand for new members of a team. Regrettably, none of these tools provide seamless integration at the model level, though many support code-level integration and application-specific parametrization of various software components.

Robotics Middleware Solutions

Various robotics-specific middleware already attempt to aid robotics researchers in writing domain-independent, application independent software.

- Gearbox [1]
- Player/Stage [4]
- Orocos [5]
- Microsoft Robotics Studio [6]

In order to use software in more than one domain, it must be configurable, and parameterizable. This has the tradeoff that it becomes much more difficult to understand for new members of a team. Regrettably, none of these tools provide seamless integration at the model level, though many support code-level integration and application-specific parametrization of various software components.

Robotics Middleware Solutions

Various robotics-specific middleware already attempt to aid robotics researchers in writing domain-independent, application independent software.

- Gearbox [1]
- Player/Stage [4]
- Orocos [5]
- Microsoft Robotics Studio [6]

In order to use software in more than one domain, it must be configurable, and parameterizable. This has the tradeoff that it becomes much more difficult to understand for new members of a team. Regrettably, none of these tools provide seamless integration at the model level, though many support code-level integration and application-specific parametrization of various software components.

Robotics Middleware Solutions

Various robotics-specific middleware already attempt to aid robotics researchers in writing domain-independent, application independent software.

- Gearbox [1]
- Player/Stage [4]
- Orocos [5]
- Microsoft Robotics Studio [6]

In order to use software in more than one domain, it must be configurable, and parameterizable. This has the tradeoff that it becomes much more difficult to understand for new members of a team. Regrettably, none of these tools provide seamless integration at the model level, though many support code-level integration and application-specific parametrization of various software components.

Robotics Middleware Solutions

Various robotics-specific middleware already attempt to aid robotics researchers in writing domain-independent, application independent software.

- Gearbox [1]
- Player/Stage [4]
- Orocos [5]
- Microsoft Robotics Studio [6]

In order to use software in more than one domain, it must be configurable, and parameterizable. This has the tradeoff that it becomes much more difficult to understand for new members of a team. Regrettably, none of these tools provide seamless integration at the model level, though many support code-level integration and application-specific parametrization of various software components.

Robotics Middleware Solutions

Various robotics-specific middleware already attempt to aid robotics researchers in writing domain-independent, application independent software.

- Gearbox [1]
- Player/Stage [4]
- Orocos [5]
- Microsoft Robotics Studio [6]

In order to use software in more than one domain, it must be configurable, and parameterizable. This has the tradeoff that it becomes much more difficult to understand for new members of a team. Regrettably, none of these tools provide seamless integration at the model level, though many support code-level integration and application-specific parametrization of various software components.

Robotics Middleware Solutions

Various robotics-specific middleware already attempt to aid robotics researchers in writing domain-independent, application independent software.

- Gearbox [1]
- Player/Stage [4]
- Orocos [5]
- Microsoft Robotics Studio [6]

In order to use software in more than one domain, it must be configurable, and parameterizable. This has the tradeoff that it becomes much more difficult to understand for new members of a team. Regrettably, none of these tools provide seamless integration at the model level, though many support code-level integration and application-specific parametrization of various software components.

The Orca Robotics Project

Orca is an open-source software framework for developing robotic systems [7].

- An *implementation framework*, not an architecture
- Developers can run their code on *any* system/network, as long as the interfaces are connected
- Known OS support includes
  - Linux
  - QNX Neutrino
  - Windows XP[1]
- Uses ICE middleware, all interfaces, etc., specified using proprietary IDLs

[1] Not all components are supported on all operating systems, but some components are supported on each.

Outline I

Outline II

Goals

Our contribution in this work can be summarized in the following goals:

- Synthesis of experiments from static design
- Management of configuration space to permit archives of previous experiments
- A graphical, constraint-based, approach that restricts ill-advised implementations

Configuration Space

Configuration files generally have the following structure:

1. component identity;
2. provided interfaces;
3. required interfaces; and
4. component configuration options.

Template Configuration File

```
# Component
COMP_TAG.Platform=p1
COMP_TAG.Component=c1
COMP_TAG.Endpoints=e1
# Provided interfaces
COMP_TAG.Provides.IFACE_TAG1.Name=i1
COMP_TAG.Provides.IFACE_TAG2.Name=i2
# Required interfaces
# Direct binding
COMP_TAG.Requires.IFACE_TAG1.Proxy=i1:e1
# Indirect binding
COMP_TAG.Requires.IFACE_TAG2.Proxy=i2@p1/c1
# Configuration Options
COMP_TAG.Config.SET_TAG1.PARAM_TAG1=0.1
COMP_TAG.Config.SET_TAG1.SET_TAG2.PARAM_TAG2=1
```

Figure: Sample component identity, provided, and required sections.

Best Practices

Generally, configuration options represent a static family of parameters with values that depend on the system under simulation or test.

Example

- Parameterization of optimization choices of a controller
- standard deviation of a sensor

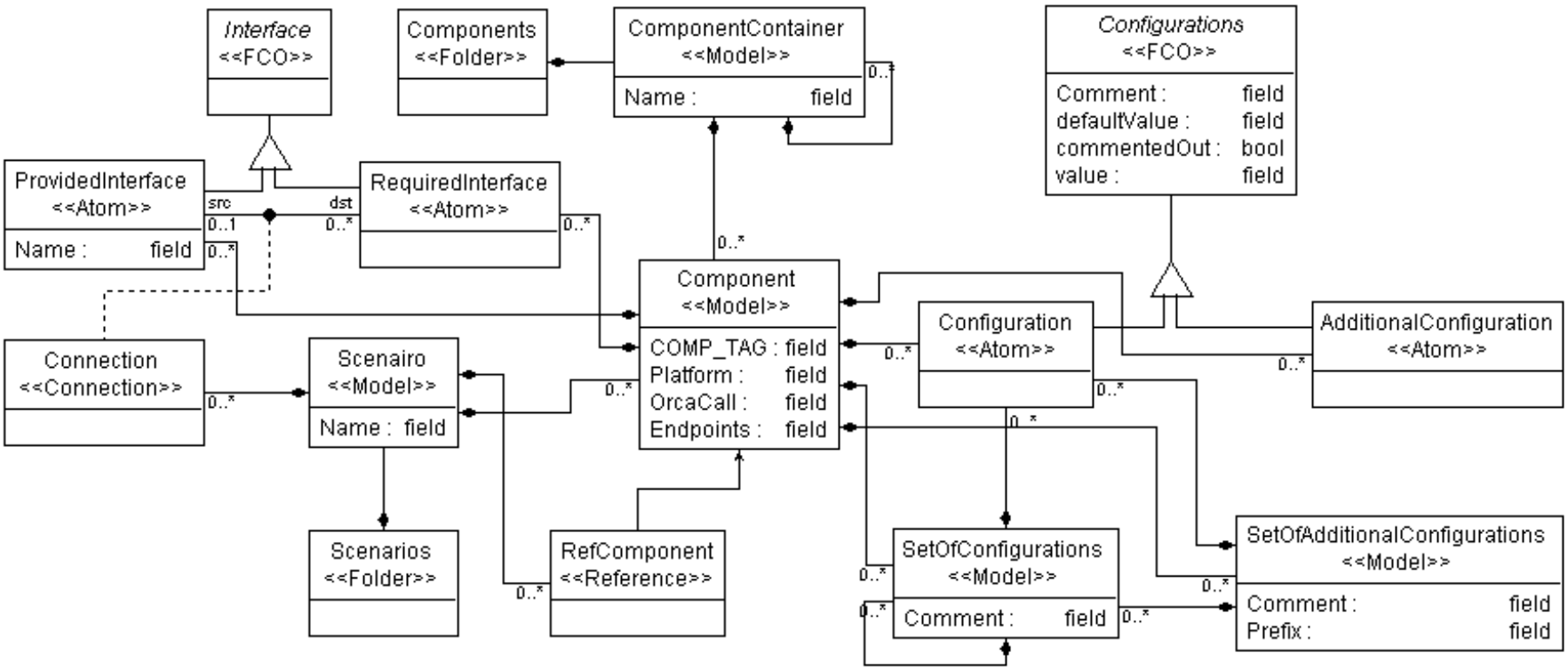
For a particular piece of hardware, the configuration options are used to specify driver information and file information such as the serial port on which the hardware is located.

Example

The `laser2d0` component gets its data from `/dev/ser0`

Language Design

We developed a language to capture data from the configuration file, as well as the component interconnection, using the GME toolsuite [8].



A few comments

The language design bears some minor emphasis in a few points:

1. Connections between components are through strong types of provided/required interfaces
2. Directional associations restrict misconstructions
3. Components can be connected to *references* of other components (to permit reuse of all parameters)
4. The configuration space can be hierarchically managed
5. The execution platform can be specified in another aspect (not shown in this metamodel, for brevity)

A few comments

The language design bears some minor emphasis in a few points:

1. Connections between components are through strong types of provided/required interfaces
2. Directional associations restrict misconstructions
3. Components can be connected to *references* of other components (to permit reuse of all parameters)
4. The configuration space can be hierarchically managed
5. The execution platform can be specified in another aspect (not shown in this metamodel, for brevity)

A few comments

The language design bears some minor emphasis in a few points:

1. Connections between components are through strong types of provided/required interfaces
2. Directional associations restrict misconstructions
3. Components can be connected to *references* of other components (to permit reuse of all parameters)
4. The configuration space can be hierarchically managed
5. The execution platform can be specified in another aspect (not shown in this metamodel, for brevity)

A few comments

The language design bears some minor emphasis in a few points:

1. Connections between components are through strong types of provided/required interfaces
2. Directional associations restrict misconstructions
3. Components can be connected to *references* of other components (to permit reuse of all parameters)
4. The configuration space can be hierarchically managed
5. The execution platform can be specified in another aspect (not shown in this metamodel, for brevity)

A few comments

The language design bears some minor emphasis in a few points:

1. Connections between components are through strong types of provided/required interfaces
2. Directional associations restrict misconstructions
3. Components can be connected to *references* of other components (to permit reuse of all parameters)
4. The configuration space can be hierarchically managed
5. The execution platform can be specified in another aspect (not shown in this metamodel, for brevity)

Benefits of this design

With these points, the following benefits are enabled:

1. Data dependencies can be analyzed at design time
2. System startup order can be computed, rather than a design input
3. Execution platform can be changed without changing component definition/configuration

Benefits of this design

With these points, the following benefits are enabled:

1. Data dependencies can be analyzed at design time
2. System startup order can be computed, rather than a design input
3. Execution platform can be changed without changing component definition/configuration

Benefits of this design

With these points, the following benefits are enabled:

1. Data dependencies can be analyzed at design time
2. System startup order can be computed, rather than a design input
3. Execution platform can be changed without changing component definition/configuration

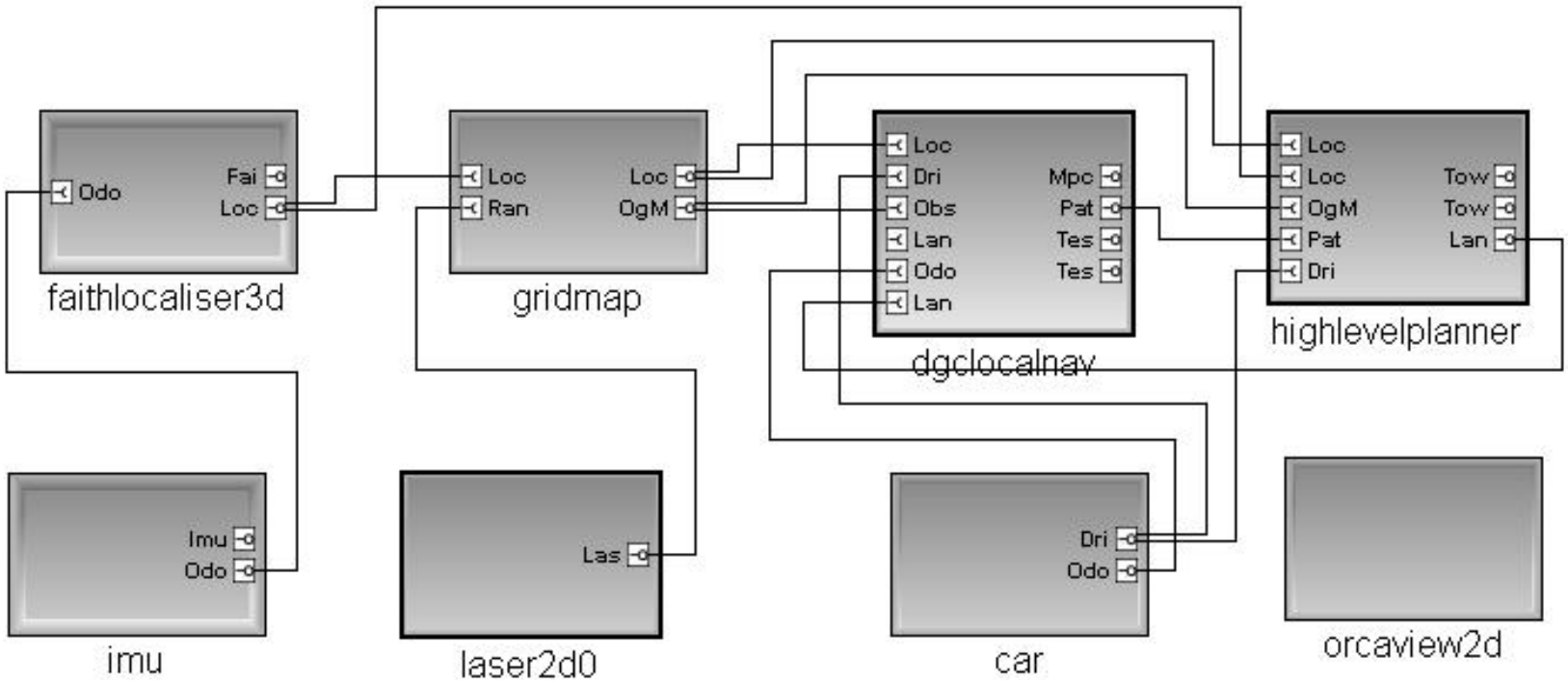


Figure: Configuration of Vehicle Computing Components

We use a two-phase traversal, employing the visitor pattern.

1. For each component, a configuration file is created
   - Name, etc., taken from the graphical model properties
   - Configuration options gathered from attributes of contained models
   - Component and platform interconnection gathered from various connections
2. Data dependencies analyzed, and experiment script generated
   - For circular dependencies, a warning is thrown, and arbitrary order selected
   - Pauses inserted for the physical system simulator startup, to avoid unnecessary errors
   - Loggers, etc., started up and autoconfigured to capture data with a unique output name
   - Middleware core services (registry and cache database) started in the appropriate order, *on the appropriate machine*

We use a two-phase traversal, employing the visitor pattern.

1. For each component, a configuration file is created
   - Name, etc., taken from the graphical model properties
   - Configuration options gathered from attributes of contained models
   - Component and platform interconnection gathered from various connections
2. Data dependencies analyzed, and experiment script generated
   - For circular dependencies, a warning is thrown, and arbitrary order selected
   - Pauses inserted for the physical system simulator startup, to avoid unnecessary errors
   - Loggers, etc., started up and autoconfigured to capture data with a unique output name
   - Middleware core services (registry and cache database) started in the appropriate order, *on the appropriate machine*

We use a two-phase traversal, employing the visitor pattern.

1. For each component, a configuration file is created
   - Name, etc., taken from the graphical model properties
   - Configuration options gathered from attributes of contained models
   - Component and platform interconnection gathered from various connections
2. Data dependencies analyzed, and experiment script generated
   - For circular dependencies, a warning is thrown, and arbitrary order selected
   - Pauses inserted for the physical system simulator startup, to avoid unnecessary errors
   - Loggers, etc., started up and autoconfigured to capture data with a unique output name
   - Middleware core services (registry and cache database) started in the appropriate order, *on the appropriate machine*

We use a two-phase traversal, employing the visitor pattern.

1. For each component, a configuration file is created
   - Name, etc., taken from the graphical model properties
   - Configuration options gathered from attributes of contained models
   - Component and platform interconnection gathered from various connections
2. Data dependencies analyzed, and experiment script generated
   - For circular dependencies, a warning is thrown, and arbitrary order selected
   - Pauses inserted for the physical system simulator startup, to avoid unnecessary errors
   - Loggers, etc., started up and autoconfigured to capture data with a unique output name
   - Middleware core services (registry and cache database) started in the appropriate order, *on the appropriate machine*

We use a two-phase traversal, employing the visitor pattern.

1. For each component, a configuration file is created
   - Name, etc., taken from the graphical model properties
   - Configuration options gathered from attributes of contained models
   - Component and platform interconnection gathered from various connections
2. Data dependencies analyzed, and experiment script generated
   - For circular dependencies, a warning is thrown, and arbitrary order selected
   - Pauses inserted for the physical system simulator startup, to avoid unnecessary errors
   - Loggers, etc., started up and autoconfigured to capture data with a unique output name
   - Middleware core services (registry and cache database) started in the appropriate order, *on the appropriate machine*

We use a two-phase traversal, employing the visitor pattern.

1. For each component, a configuration file is created
   - Name, etc., taken from the graphical model properties
   - Configuration options gathered from attributes of contained models
   - Component and platform interconnection gathered from various connections
2. Data dependencies analyzed, and experiment script generated
   - For circular dependencies, a warning is thrown, and arbitrary order selected
   - Pauses inserted for the physical system simulator startup, to avoid unnecessary errors
   - Loggers, etc., started up and autoconfigured to capture data with a unique output name
   - Middleware core services (registry and cache database) started in the appropriate order, *on the appropriate machine*

We use a two-phase traversal, employing the visitor pattern.

1. For each component, a configuration file is created
   - Name, etc., taken from the graphical model properties
   - Configuration options gathered from attributes of contained models
   - Component and platform interconnection gathered from various connections
2. Data dependencies analyzed, and experiment script generated
   - For circular dependencies, a warning is thrown, and arbitrary order selected
   - Pauses inserted for the physical system simulator startup, to avoid unnecessary errors
   - Loggers, etc., started up and autoconfigured to capture data with a unique output name
   - Middleware core services (registry and cache database) started in the appropriate order, *on the appropriate machine*

We use a two-phase traversal, employing the visitor pattern.

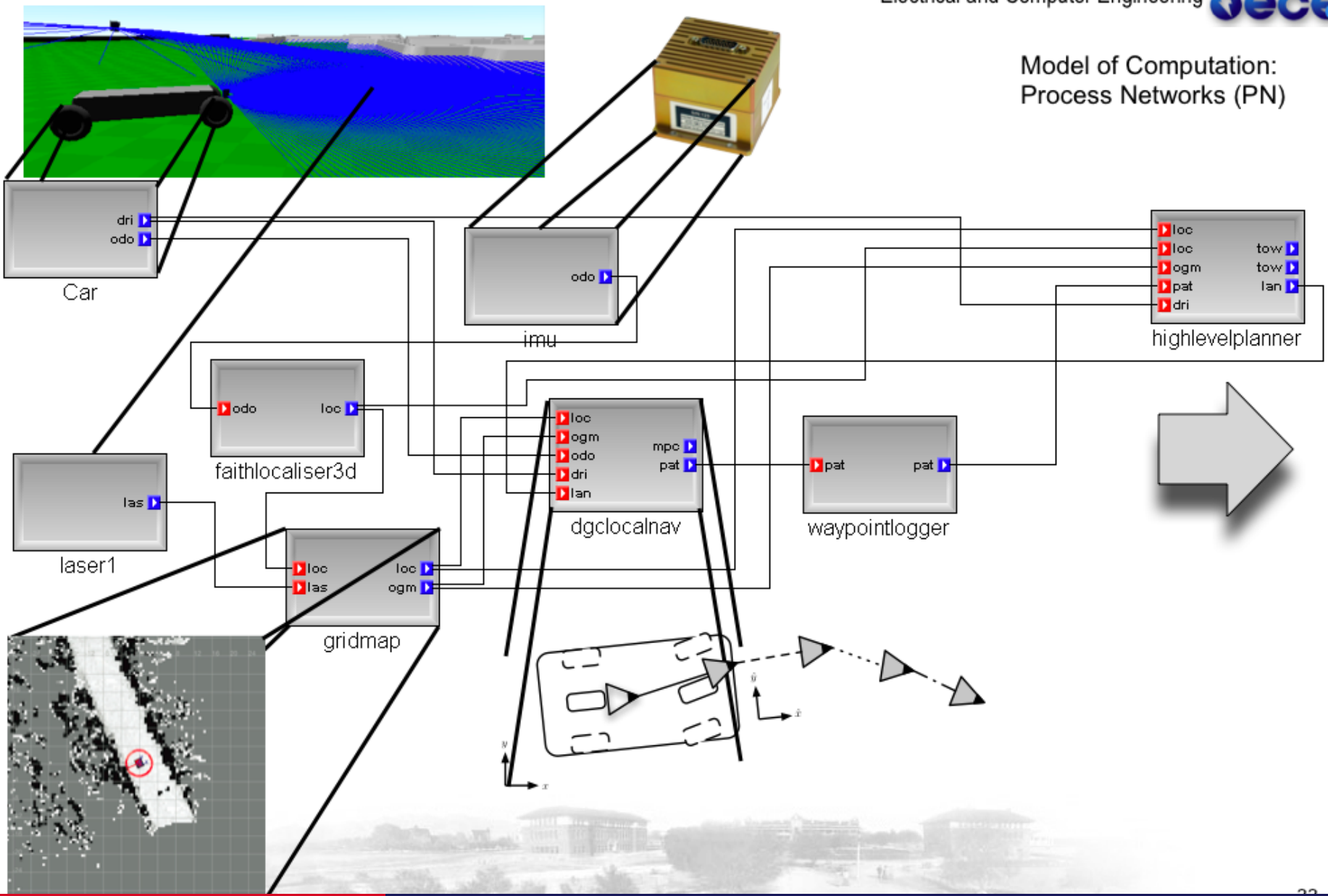
1. For each component, a configuration file is created
   - Name, etc., taken from the graphical model properties
   - Configuration options gathered from attributes of contained models
   - Component and platform interconnection gathered from various connections
2. Data dependencies analyzed, and experiment script generated
   - For circular dependencies, a warning is thrown, and arbitrary order selected
   - Pauses inserted for the physical system simulator startup, to avoid unnecessary errors
   - Loggers, etc., started up and autoconfigured to capture data with a unique output name
   - Middleware core services (registry and cache database) started in the appropriate order, *on the appropriate machine*

We use a two-phase traversal, employing the visitor pattern.

1. For each component, a configuration file is created
   - Name, etc., taken from the graphical model properties
   - Configuration options gathered from attributes of contained models
   - Component and platform interconnection gathered from various connections
2. Data dependencies analyzed, and experiment script generated
   - For circular dependencies, a warning is thrown, and arbitrary order selected
   - Pauses inserted for the physical system simulator startup, to avoid unnecessary errors
   - Loggers, etc., started up and autoconfigured to capture data with a unique output name
   - Middleware core services (registry and cache database) started in the appropriate order, *on the appropriate machine*

Electrical and Computer Engineering

Model of Computation:
Process Networks (PN)



Outline I

Outline II

- Runtime Types

4 Future and Ongoing Work
- UA Autonomous Ground Vehicles

startsim.sh

```
1
2   #!/bin/bash
3   # This script launches a particular demonstration
4   # The script should only be run in the $demo/sys directory
5   #  where $demo is your preferred directory (see the
6   #  orca2 examples for why this directory is chosen)
7
8   # first, allow processes to dump (up to 1Gb)
9   ulimit -c 2097152
10
11  # DIRECTORY
12  DIRECTORY=basicsim
13
14  # CONFIG Files
15  # Insert them IN THE ORDER You would like to start them!!!!!
16  CONFIG_FILES="car imu faithlocaliser3d laser2d gridmap dgclocalnav w
17
18
```

```
19  PLAYER_CMD=player
20  PLAYER_WORLD=gazebocar.cfg
21
22  # use this instead to get 'graphical' gazebo
23  #SIMULATOR_CMD=wxgazebo
24  SIMULATOR_CMD=gazebo
25
26  # use this instead to get obstacles
27  #SIMULATOR_WORLD=carblankWithObstacles.world
28  SIMULATOR_WORLD=carblank-norender.world
29
30  if [ -e ${DIRECTORY}/core ] ; then
31   echo "You already have a core file in this directory - removing it
32   sleep 2
33   rm ${DIRECTORY}/core*
34  fi
35
36  if [ -e /tmp/gazebo-${USER}-0 ]; then
37   echo "You already have a gazebo dir in /tmp/gazebo-${USER}-0 . I w
```

```
38    rm -r /tmp/gazebo-${USER}-0
39   fi
40
41   PIDFILE=.pidfile
42
43   # getpid.sh opens a konsole window and returns its pid
44   ./getpid.sh > $PIDFILE
45   sleep 4
46   PID=‘head $PIDFILE‘
47   echo "Using PID=${PID}"
48
49   sleep 1
50
51   if [ ${SIMULATOR_WORLD} != "" ]; then
52    echo "Simulating using ${SIMULATOR_CMD}..."
53    ./dosimulator.sh ${DIRECTORY} ${SIMULATOR_WORLD} ${SIMULATOR_CMD} $
54   fi
55
56   sleep 2
```

```
57
58  # do it again for player
59  if [ ${PLAYER_WORLD} != "" ]; then
60   echo "Starting up ${PLAYER_CMD} now..."
61   ./dosimulator.sh ${DIRECTORY} ${PLAYER_WORLD} ${PLAYER_CMD} ${PID}
62  fi
63
64  # uncomment if you want to run your ice stuff locally
65  # make sure that you have a proper ~/.orcarc file, regardles...
66  ./doice.sh ${PID}
67
68  ./doshells.sh ${DIRECTORY} ${PID} ${CONFIG_FILES}
69
70  # we remove the pidfile later, so as to allow
71  # someone else to kill us off...
72  #rm -f $PIDFILE
73
```

car.cfg

```
1   # Orca version 2.2.0+
2
3   # Component
4   Car.Platform=local
5   Car.Component=car
6
7   # Provided Interfaces
8   Car.Provides.DriveBicycle.Name=drivebicycle
9   Car.Provides.Odometry2d.Name=odometry2d
10
11  # Configuration Options
12  # Options are { 'rav4', 'playerclient', 'fake' }
13  Car.Config.ControlDriver=playerclient
14  # Options are { 'rav4', 'fake' }
15  Car.Config.DataDriver=playerclient
16  Car.Config.EnableMotion=1
17  Car.Config.Odometry2dPublishInterval=0.1
18  Car.Config.PlayerClient.Host=localhost
```

```
19  Car.Config.PlayerClient.Port=6665
20  Car.Config.VehicleDescription.Control.Type=VelocityBicycle
21  Car.Config.VehicleDescription.Control.VelocityBicycle.MaxForwardAcce
22  Car.Config.VehicleDescription.Control.VelocityBicycle.MaxForwardSpee
23  Car.Config.VehicleDescription.Control.VelocityBicycle.MaxReverseAcce
24  Car.Config.VehicleDescription.Control.VelocityBicycle.MaxReverseSpee
25  Car.Config.VehicleDescription.Control.VelocityBicycle.MaxSteerAngle=
26  Car.Config.VehicleDescription.Control.VelocityBicycle.MaxSteerAngleA
27  Car.Config.VehicleDescription.Control.VelocityBicycle.MaxSteerAngleR
28  Car.Config.VehicleDescription.Geometry.Cuboid.Size=2.667 1.57 1.5
29  # Car.Config.VehicleDescription.Geometry.Cylindrical.VehicleToGeomet
30  Car.Config.VehicleDescription.Geometry.Type=Cuboid
31  Car.Config.VehicleDescription.PlatformToVehicleTransform=0.0 0.0 0.0
32
```

laser2d.cfg

```
1   # Orca version dgc
2
3   # Component
4   Laser2d.Platform=local
5   Laser2d.Component=laser2d0
6
7   # Provided Interfaces
8   Laser2d.Provides.LaserScanner2d.Name=laserscanner2d
9
10  # Configuration Options
11  Laser2d.Config.AllowRollCompensation=1
12  # Valid values are at least: { 'libOrcaLaser2dSickCarmen.so', 'libOr
13  #Laser2d.Config.DriverLib=libOrcaLaser2dSickCarmen.so
14  Laser2d.Config.DriverLib=libOrcaLaser2dPlayerClient.so
15  Laser2d.Config.FieldOfView=180.0
16  Laser2d.Config.MaxRange=80.0
17  Laser2d.Config.MinRange=0.0
18  Laser2d.Config.NumberOfSamples=181
```

```
19  # x[m] y[m] z[m] roll[deg] pitch[deg] yaw[deg]
20  #Laser2d.Config.Offset=0.0 0.0 0.0  0.0 0.0 0.0
21  Laser2d.Config.Offset=1.27 0.0 1.5  0.0 0.0 0.0
22  Laser2d.Config.PlayerClient.Device=0
23  # Valid values: { "sicklms200", "stage", "gazebo", "urglaser" }
24  #Laser2d.Config.PlayerClient.Driver=sicklms200
25  Laser2d.Config.PlayerClient.Driver=stage
26  Laser2d.Config.PlayerClient.Host=localhost
27  Laser2d.Config.PlayerClient.Port=6665
28  # Valid values: { 9600, 19200, 38400, 500000 }
29  Laser2d.Config.SickAcfr.Baudrate=38400
30  Laser2d.Config.SickAcfr.Device=/dev/ser1
31  # Valid values: { 9600, 19200, 38400, 500000 }
32  Laser2d.Config.SickCarmen.Baudrate=38400
33  Laser2d.Config.SickCarmen.Device=/dev/ttyS0
34  Laser2d.Config.SickCarmen.LaserType=LMS
35  # length[m] width[m] height[m]
36  Laser2d.Config.Size=0.155 0.155 0.185
37
```

imu.cfg

```
1   # Orca version 2.1.1+
2
3   # Component
4   Imu.Platform=local
5   Imu.Component=imu
6   Imu.Endpoints=tcp -t 5000
7
8   # Provided Interfaces
9   Imu.Provides.Imu.Name=imu
10  Imu.Provides.Odometry3d.Name=odometry3d
11
12  # Configuration Options
13  Imu.Config.Baud=4800
14  Imu.Config.Device=/dev/ttyS0
15  # Options are { 'playerclientodometry3d', 'fake' }
16  Imu.Config.Driver=playerclientodometry3d
17  # x[m] y[m] z[m]
18  Imu.Config.frameOffset=0.0 0.0 0.0 # no IMU offset
```

```
19  #Imu.Config.frameOffset=-1.27 0.0 0.0 # put the IMU over the rear ax
20  #Imu.Config.frameOffset=1.27 0.0 0.0 # put the IMU over the front ax
21  Imu.Config.PlayerClient.Device=0
22  Imu.Config.PlayerClient.Host=localhost
23  Imu.Config.PlayerClient.Port=6665
24  Imu.Config.StartEnabled=1
25
```

dgclocalnav.cfg

```
1   # Orca version 2.2.0+
2
3   # Component
4   DgcLocalNav.Platform=local
5   DgcLocalNav.Component=dgclocalnav
6
7   # Provided Interfaces
8   DgcLocalNav.Provides.MpcDebugGraphics.Name=mpcgraphics
9   DgcLocalNav.Provides.PathFollower2d.Name=pathfollower2d
10  DgcLocalNav.Provides.TestLocalise.Name=testlocalise
11  DgcLocalNav.Provides.TestOgMap.Name=testogmap
12
13  # Required Interfaces
14  DgcLocalNav.Requires.Localisation.Proxy=localise2d@local/gridmap
15  DgcLocalNav.Requires.Observations.Proxy=ogmap@local/gridmap
16  DgcLocalNav.Requires.Odometry2d.Proxy=odometry2d@local/car
17  DgcLocalNav.Requires.DriveBicycle.Proxy=drivebicycle@local/car
18  #DgcLocalNav.Requires.Lanes.Proxy=lanes@local/lanedetector
```

```
19  #DgcLocalNav.Requires.Lanes.Proxy=lanes@local/highlevelplanner
20
21  # Configuration Options
22  DgcLocalNav.Config.ConnectToLanes=0
23  DgcLocalNav.Config.TestMode=0
24  Ice.MessageSizeMax=3000
25
26  # FIGHT!
27  #DgcLocalNav.Config.DriverLib=libDgcLocalNavMpc.so
28  DgcLocalNav.Config.DriverLib=libDgcLocalNavMpcUcb.so
29
30  # ACFR Options
31  DgcLocalNav.Config.Mpc.ActDelaySec=0.050
32  DgcLocalNav.Config.Mpc.Control.Type=VelocityBicycle
33  DgcLocalNav.Config.Mpc.Control.VelocityBicycle.MaxForwardAcceleratio
34  DgcLocalNav.Config.Mpc.Control.VelocityBicycle.MaxForwardSpeed=6.0
35  DgcLocalNav.Config.Mpc.Control.VelocityBicycle.MaxReverseAcceleratio
36  DgcLocalNav.Config.Mpc.Control.VelocityBicycle.MaxReverseSpeed=1.0
37  DgcLocalNav.Config.Mpc.Control.VelocityBicycle.MaxSteerAngle=30.0
```

```
38  DgcLocalNav.Config.Mpc.Control.VelocityBicycle.MaxSteerAngleAtMaxSpe
39  DgcLocalNav.Config.Mpc.Control.VelocityBicycle.MaxSteerAngleRate=20.
40  DgcLocalNav.Config.Mpc.DisplayDebugGraphics=1
41  DgcLocalNav.Config.Mpc.NumExpansions=400
42  DgcLocalNav.Config.Mpc.NumPrevPlanPerturbations=10
43  DgcLocalNav.Config.Mpc.SpeedFor1MClearance=2.0
44  DgcLocalNav.Config.Mpc.TimeHorizon=8.0
45
46  # UCB Options
47  # increase this value to get better horizon (increases computational
48  DgcLocalNav.Config.Mpc.Control.mpcStateHorizon=20
49
50  DgcLocalNav.Config.Mpc.Control.obstacleDelta=0.2
51  DgcLocalNav.Config.Mpc.Control.obstacleConstant=200.0
52  DgcLocalNav.Config.Mpc.Control.obstacleActionDistance=12.0
53  DgcLocalNav.Config.Mpc.Control.laneObstacleDelta=0.7
54  DgcLocalNav.Config.Mpc.Control.laneObstacleConstant=20.0
55  DgcLocalNav.Config.Mpc.Control.laneObstacleActionDistance=0.6
56
```

```
57  # increase these values to improve orientation or position
58  DgcLocalNav.Config.Mpc.Control.qWeightXpos=5.0
59  DgcLocalNav.Config.Mpc.Control.qWeightYpos=5.0
60  DgcLocalNav.Config.Mpc.Control.qWeightOrientation=5.0
61  DgcLocalNav.Config.Mpc.Control.qWeightVelocity=10.0
62  DgcLocalNav.Config.Mpc.Control.qWeightWheelAngle=5.0
63
64  DgcLocalNav.Config.Mpc.Control.rWeightAcceleration=2.0
65  DgcLocalNav.Config.Mpc.Control.rWeightWheelAngleRate=2.0
66
67  DgcLocalNav.Config.Mpc.Control.kNormalizeState=1.10
68  DgcLocalNav.Config.Mpc.Control.kNormalizeInputs=1.00
69  DgcLocalNav.Config.Mpc.Control.kNormalizeObstacles=1.00
70  DgcLocalNav.Config.Mpc.Control.kNormalizeWayline=1.50
71  DgcLocalNav.Config.Mpc.Control.kNormalizeNegativeVelocity=3.00
72
73  DgcLocalNav.Config.Mpc.Control.steeringAngle=1.0
74  DgcLocalNav.Config.Mpc.Control.timestep=0.1
75  DgcLocalNav.Config.Mpc.Control.vehicleBaseLength=2.6
```

```
76
77  DgcLocalNav.Config.Mpc.Control.speedMin=-1.0
78  DgcLocalNav.Config.Mpc.Control.speedMax=6.0
79  DgcLocalNav.Config.Mpc.Control.accelerationMin=-1.0
80  DgcLocalNav.Config.Mpc.Control.accelerationMax=1.0
81  DgcLocalNav.Config.Mpc.Control.wheelAngleMin=-0.523598776
82  DgcLocalNav.Config.Mpc.Control.wheelAngleMax=0.523598776
83  DgcLocalNav.Config.Mpc.Control.wheelAngleRateMin=-0.698131701
84  DgcLocalNav.Config.Mpc.Control.wheelAngleRateMax=0.698131701
85
86  DgcLocalNav.Config.Mpc.Control.useObstacles = 0
87
88  # Experimental variable to put in a fake obstacle
89  Config.Mpc.Control.useFakeObstacle.Default=0
90  Config.Mpc.Control.fakeObstacleLat.Default=85
91  Config.Mpc.Control.fakeObstacleLon.Default=-20
92  Config.Mpc.Control.fakeObstacleSize.Default=5
```

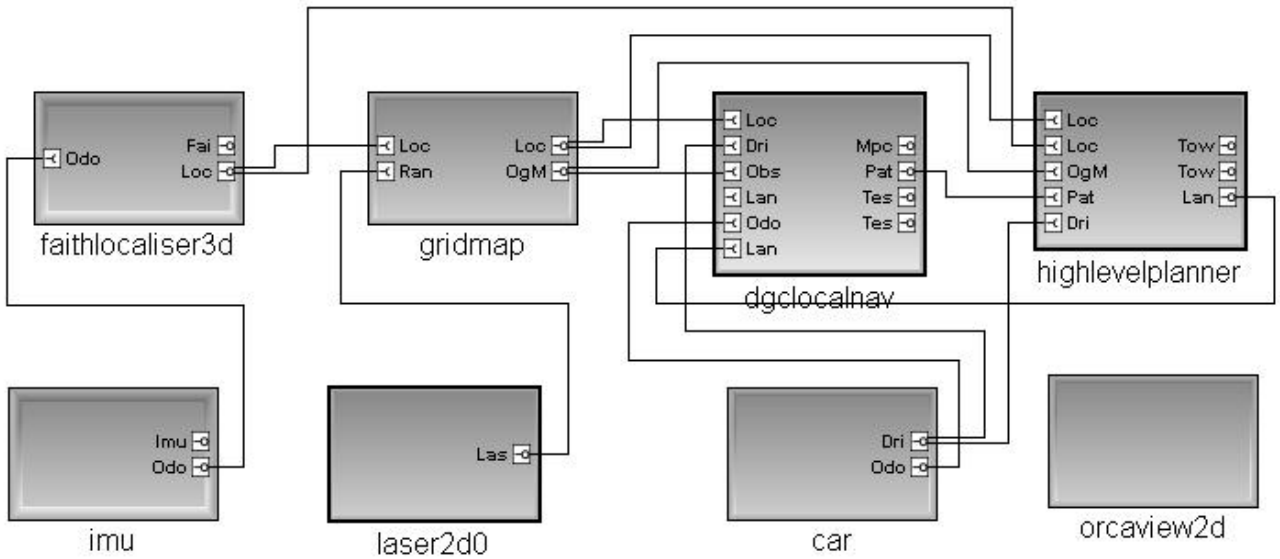


Figure: Running Example

Three different types used:

1. Models created inside *this* experiment (`faithlocaliser3d`)
2. Models included as prototypes of a base class (`gridmap`,`car`)
3. Pointers to models in a library/repository (`highlevelplanner`,`laser2d0`)

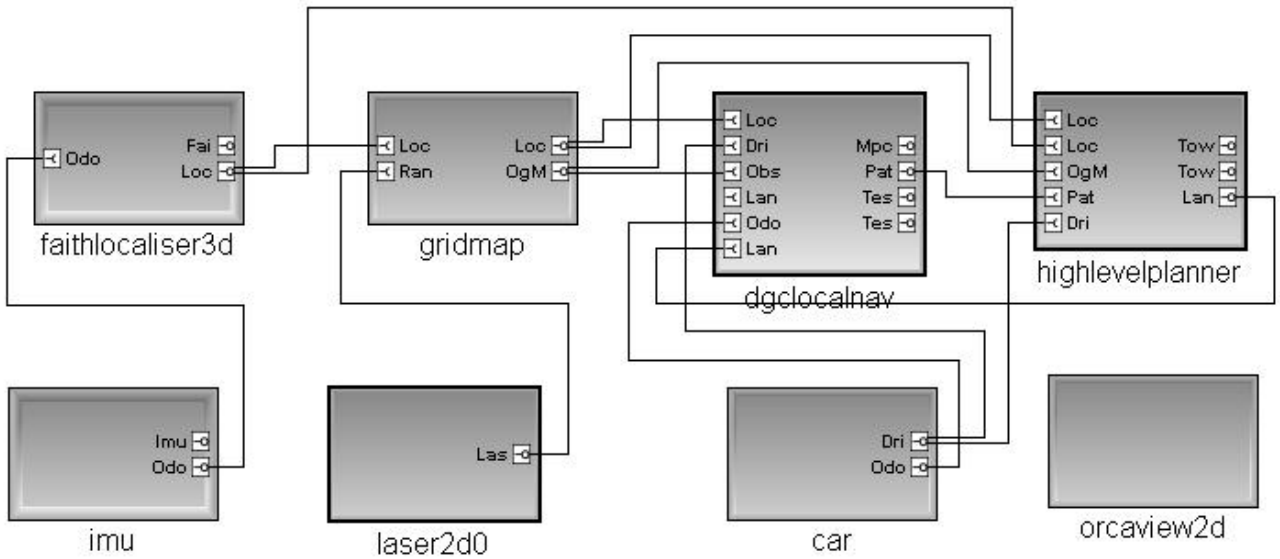


Figure: Running Example

Three different types used:

1. Models created inside *this* experiment (`faithlocaliser3d`)
2. Models included as prototypes of a base class (`gridmap`,`car`)
3. Pointers to models in a library/repository (`highlevelplanner`,`laser2d0`)

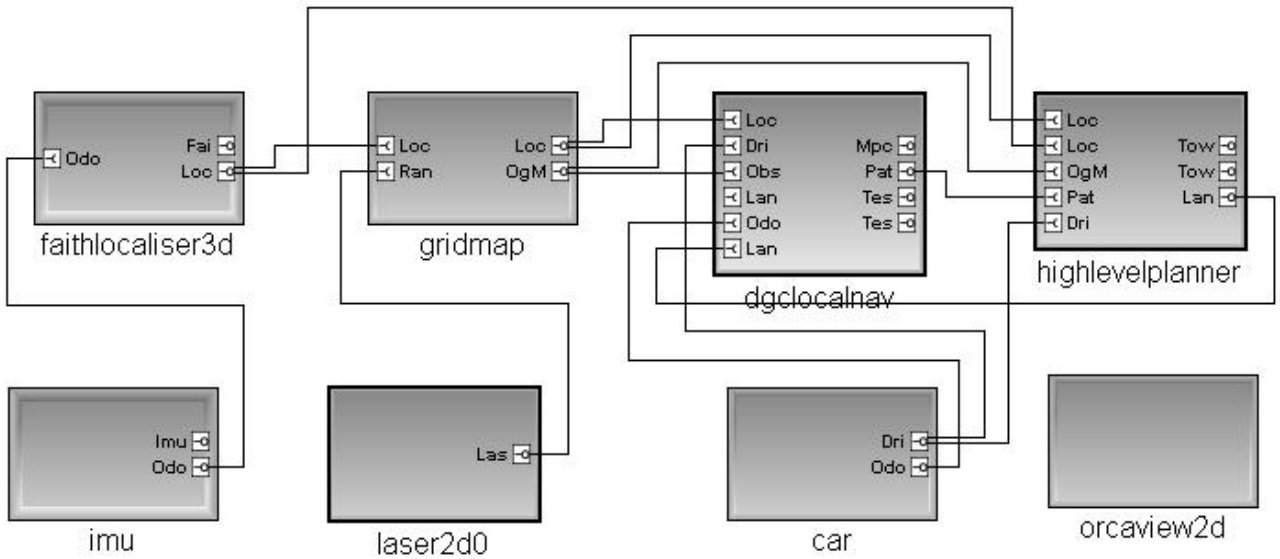


Figure: Running Example

Three different types used:

1. Models created inside *this* experiment (`faithlocaliser3d`)
2. Models included as prototypes of a base class (`gridmap`,`car`)
3. Pointers to models in a library/repository (`highlevelplanner`,`laser2d0`)

Why this kind of component reuse?

- Prototypes of base classes
  - Reuse the structure of a component, but change attributes
- Pointers to library classes
  - Force reuse of all attributes of a component, including structure, but permit different interconnection for experimentation

Constraints

In order to enforce the so called *static semantics* of our language we developed constraints, which restrict some (normally allowed) syntax patterns when used in certain contexts. These constraints are expressed in an extension of OCL available in GME.

```
Component.COMP_TAG.trim() <> ""
...
RequiredInterface.connectedFCOs("src")->size <= 1
```

Figure: Constraints

OrcaGUI

Previous releases of the Orca framework provided a GUI to provide a similar method of specification of configuration files.

- Recent Orca releases do not provide this GUI as most users of Orca are power users, and do not need the graphical syntax.
- Our solution provides a more configurable, and also more structured, modeling environment
- Specific differences include:
  - previous experiments can be easily archived
  - future experiments can refer to previous/current experiments
- Differences from CoSMIC
  - Our goal is not a generic middleware specification ADL
  - We focus on *robotics experiment* domain-specific middleware, where parameter values and topological rerouting is common
  - We expect domain experts, but not Orca experts, to become more confident with Orca through using this tool.

OrcaGUI

Previous releases of the Orca framework provided a GUI to provide a similar method of specification of configuration files.

- Recent Orca releases do not provide this GUI as most users of Orca are power users, and do not need the graphical syntax.
- Our solution provides a more configurable, and also more structured, modeling environment
- Specific differences include:
  - previous experiments can be easily archived
  - future experiments can refer to previous/current experiments
- Differences from CoSMIC
  - Our goal is not a generic middleware specification ADL
  - We focus on *robotics experiment* domain-specific middleware, where parameter values and topological rerouting is common
  - We expect domain experts, but not Orca experts, to become more confident with Orca through using this tool.

Outline I

Outline II

Timing can also be important

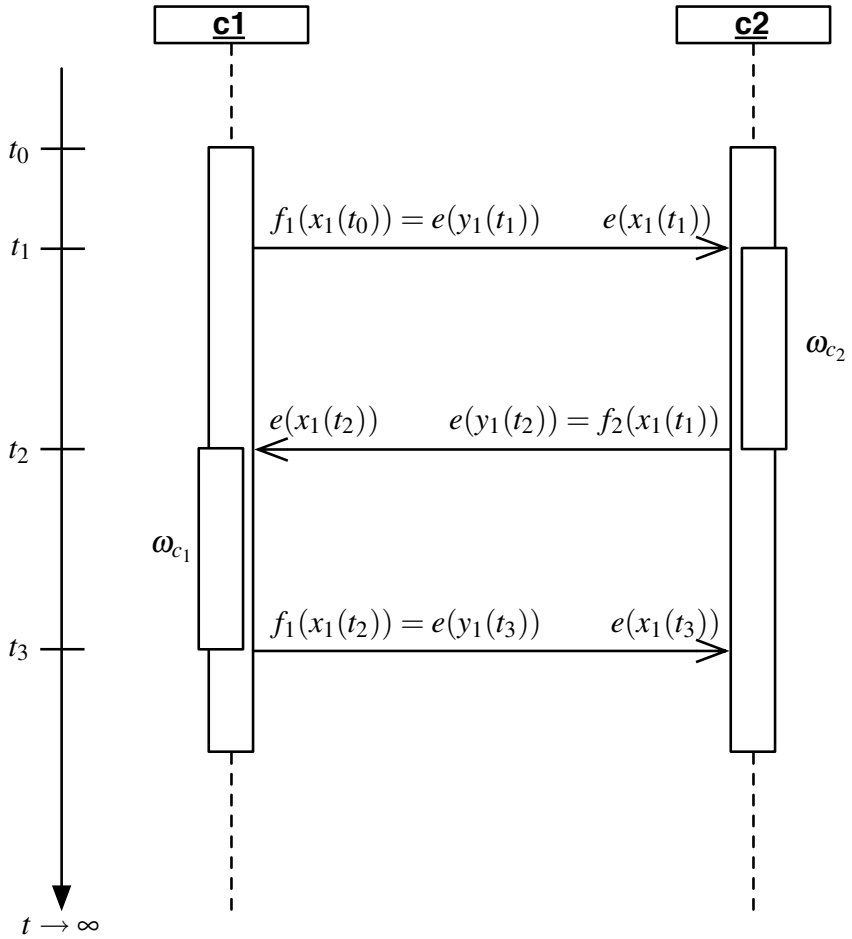


Figure: Proper timing results in expected behavior.

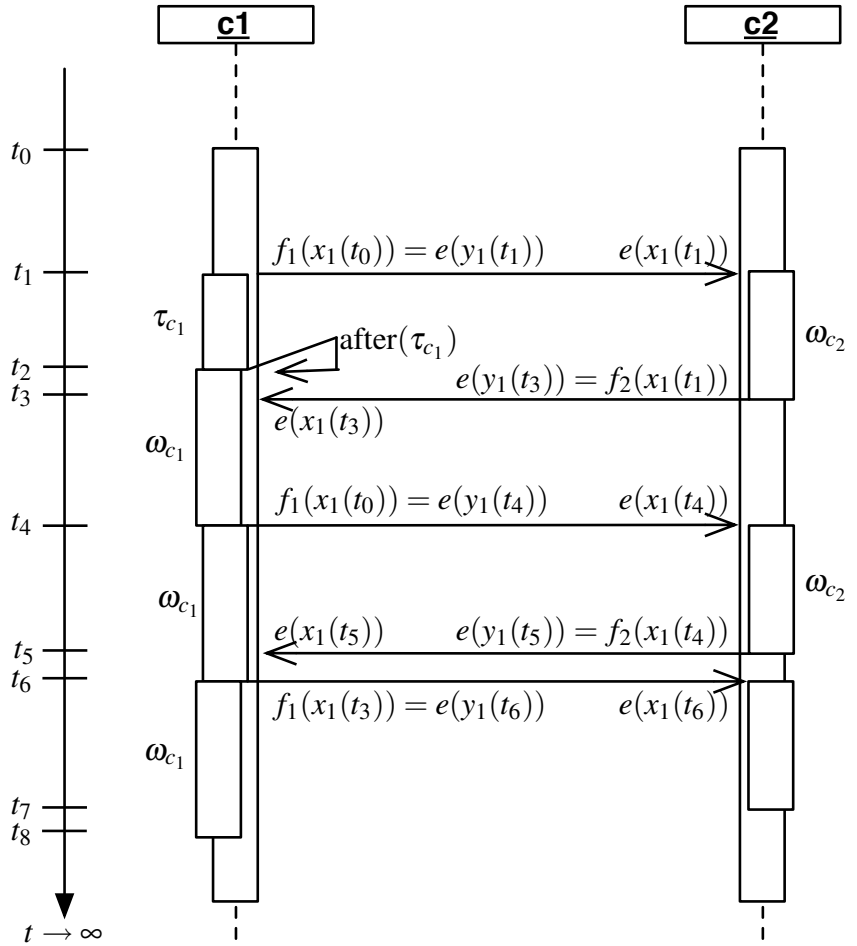


Figure: Improper timing results in "message chattering".

Model Transformations for Timing

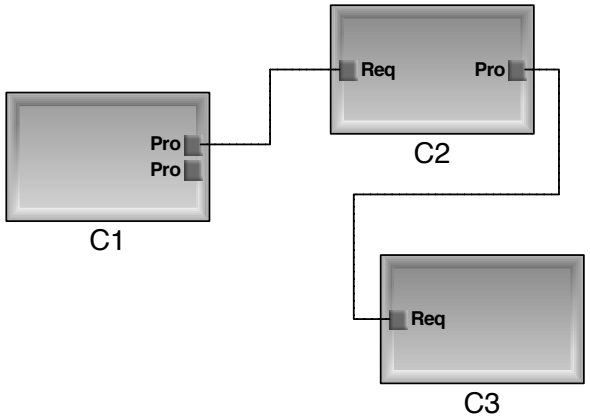


Figure: Example depends on internal timeouts, if no data are received..

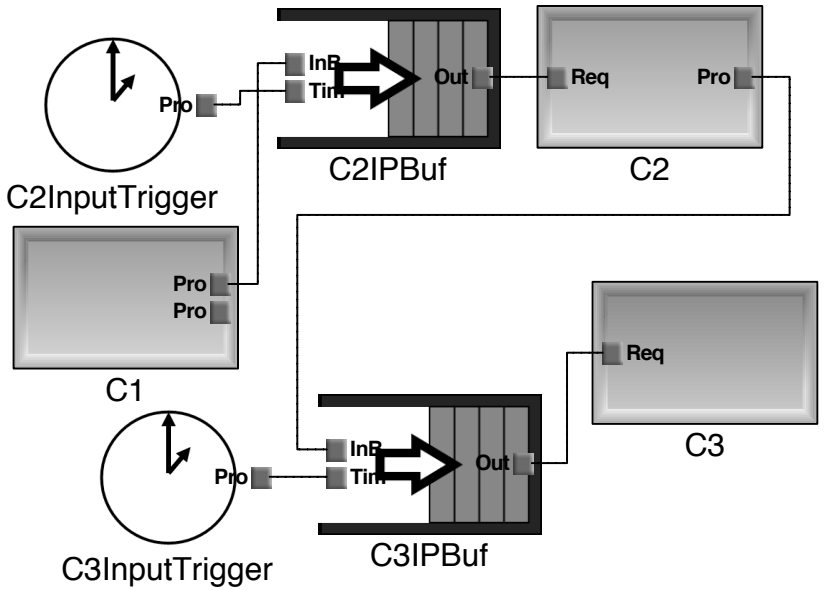


Figure: Time-triggered buffers inserted, instead of timeout values.


4618
STATE OF ARIZONA FOR OFFICIAL USE ONLY


A. Makarenko, A. Brooks, and T. Kaupp, "On the benefits of making robotic software frameworks thin," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'07) Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware* (E. Prassler, K. Nilsson, and A. Shakhimardanov, eds.), November 2007.


J. Sprinkle *et al.*, "Model-based design: a report from the trenches of the DARPA urban challenge," *Software and Systems Modeling*, vol. 8, pp. 551–566, September 2009.


D. Schmidt *et al.*, "CoSMIC: An MDA generative tool for distributed real-time and embedded component middleware and applications," in *OOPSLA 2002 Workshop on Generative Techniques in the Context of Model Driven Architecture, Seattle, WA*, 2002.

ARIZONA

T. Collett, B. MacDonald, and B. Gerkey, "Player 2.0: Toward a practical robot programming framework," in *Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2005)*, 2005.

H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the Orocos project," in *IEEE International Conference on Robotics and Automation*, pp. 2766–2771, 2003.

S. Cherry, "Robots incorporated," *Spectrum, IEEE*, vol. 44, pp. 24–29, Aug. 2007.

A. Makarenko, A. Brooks, and T. Kaupp, "Orca: Components for robotics," in *International Conference on Intelligent Robots and Systems (IROS)*, pp. 163–168, 2006.

ARIZONA


Ákos Lédeczi, Árpad Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai, "Composing domain-specific design environments," *IEEE Computer*, vol. 34, pp. 44–51, November 2001.

We're always looking for good graduate students.

http://ece.arizona.edu/