



Software Architecture for Planetary & Lunar Robotics

Hans Utz
Research Institute for Advanced Computer Sciences at
Intelligent Robotics Group
NASA Ames Research Center

Terry Fong
Intelligent Robotics Group
NASA Ames Research Center

Issa A.D. Nesnas
Jet Propulsion Laboratory
California Institute of Technology



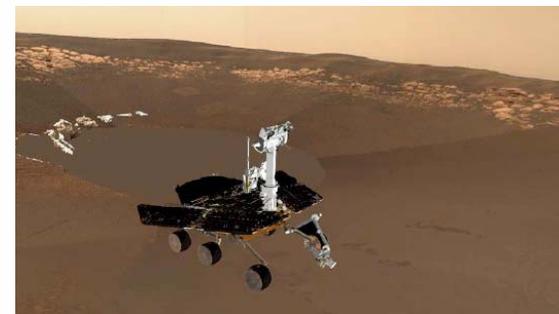
Overview

- The Intelligent Robotics Group
- Software architecture for space robotics
- Challenges in rover interoperability
- The CLARAty approach
- CLARAty examples:
 - Locomotion framework
 - Motor abstraction
- Future development
- Summary and conclusions



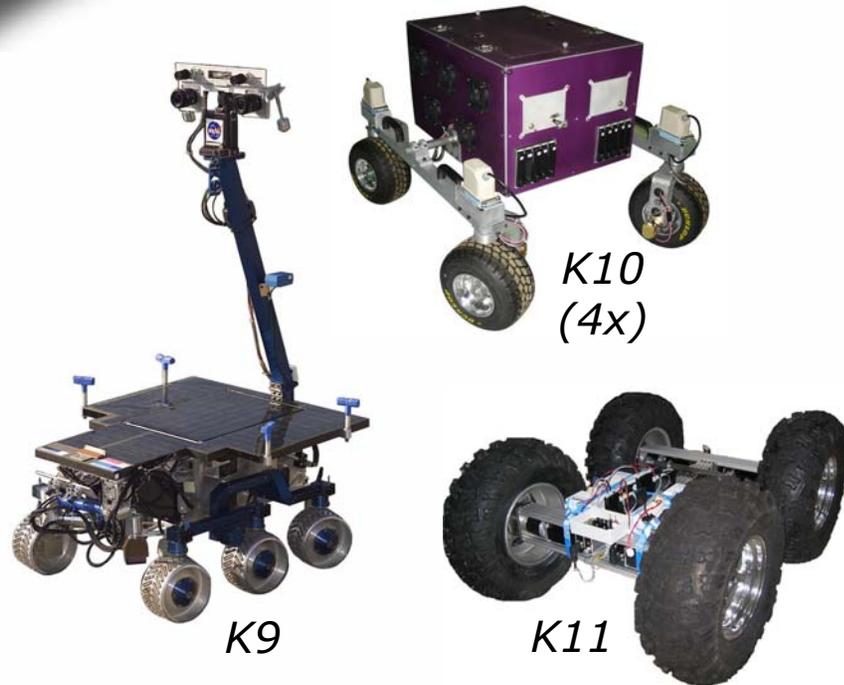
Intelligent Robotics Group

- Areas of expertise
 - Applied computer vision
 - Human-robot interaction
 - Interactive 3D visualization
 - Robot software architectures
- Science-driven exploration
 - Survey, instrument placement, resource mapping
 - Low speed, deliberative operation
- Fieldwork-driven operations
 - Assembly, inspection, maintenance
 - Pre-cursor missions (site preparation, infrastructure emplacement, etc.)
 - Manned missions (human-paced interaction, peer-to-peer assistance)





Robots and Facilities



K9

K10
(4x)

K11



Amtec
Schunk
(2x)



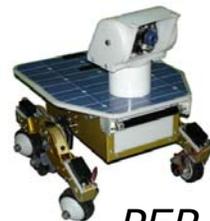
Marscape



Moonscape



Scorpion



PER
(10x)

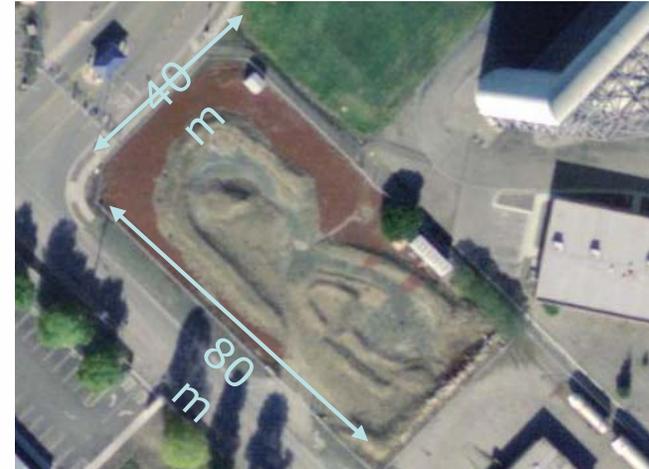


Rover lab



Marscape

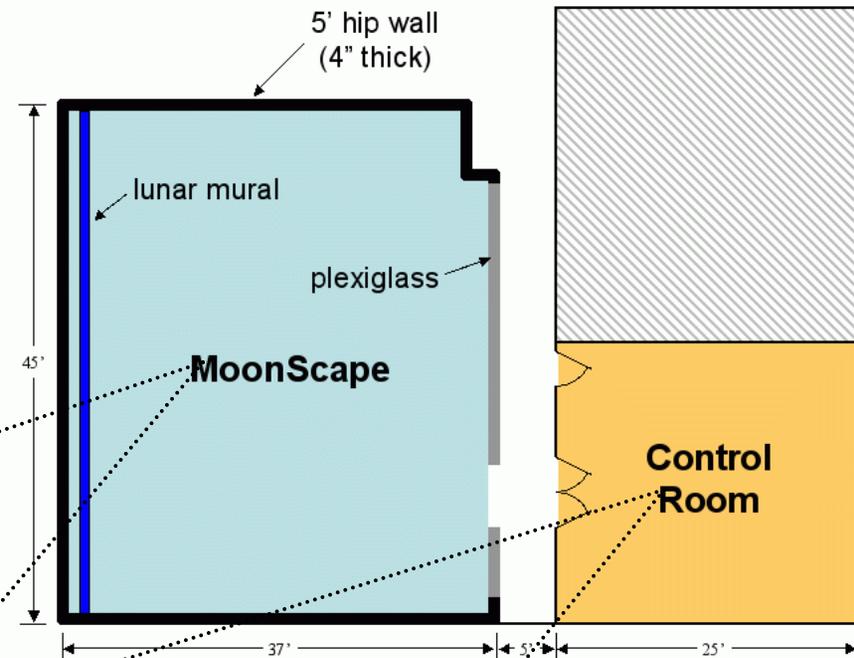
- Outdoor rover test facility
 - 3/4 acre, surveyed site
 - Operations trailer
 - dGPS, wireless LAN, power
- Mars analog
 - Design reflects geology of scientific interest
 - Streambed, delta, lakebed, volcano, chaotic terrain, meteorite impact crater, etc.
 - Traversable + non-traversable regions (with occlusions)





Moonscape

- Indoor rover test facility
 - Human-Robot studies
- Work area
 - 37 x 45 ft with hip wall
 - Lunar mural & floor
 - Optical motion capture
- Control room
 - High-end graphics PC's
 - Plasma touchscreen





K9 Rover

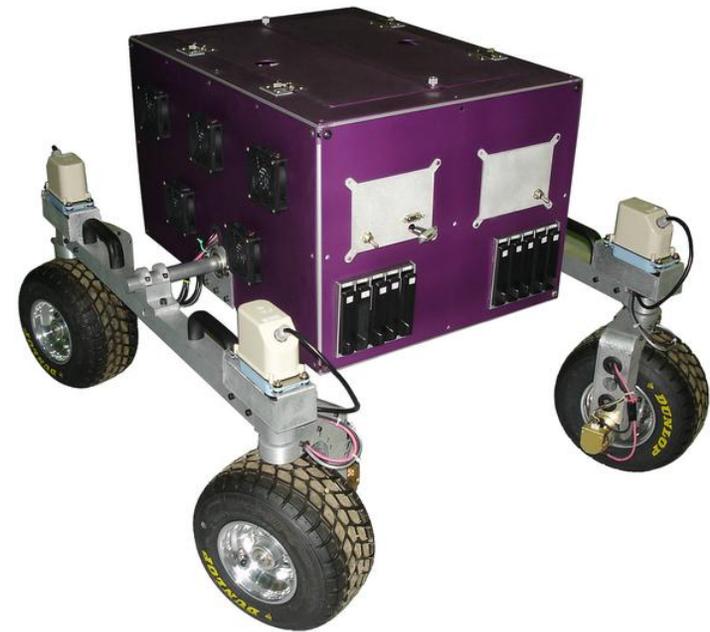
- Planetary science rover
 - Remote autonomous experiments
 - In-situ measurements
- Characteristics
 - FIDO chassis: 6-wheel steer rocker-bogey
 - 5-DOF instrument arm
 - Size: 1.7 x 0.8 x 1m (HxWxL) with mast
 - Speed: 6 cm/s
 - Power: 570 W (Li-Ion batteries)
 - Weight: 70 kg
- Instrumentation
 - CHAMP: Camera, Handlens, and Microscope Probe (Mungus / JPL)
 - 6x Dragonfly cameras (navigation)
 - 2x Basler area scan cameras (science)





K10 Rover

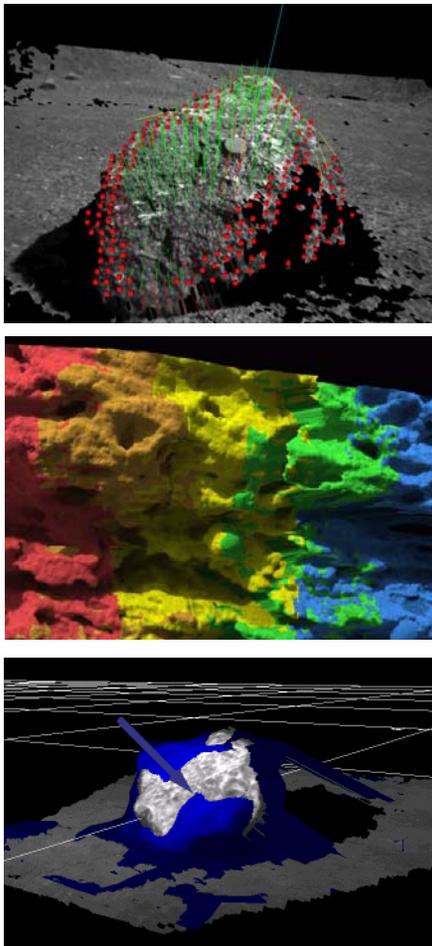
- Field work rover
 - Operational tasks (assembly, inspection, etc.)
 - Human paced operations
 - Same avionics and software as K9
- Characteristics
 - 4-wheel steer rocker chassis
 - Low-cost (COTS parts)
 - Size: 0.6 x 0.7 x 1 m (HxWxL)
 - Speed: 0.8 m/s (10 deg slope)
 - Power: 1900 W (Li-Ion batteries)
 - Weight: 100 kg (30 kg payload)
- Development
 - 2004: initial build (Hsiu / Gogoco LLC)
 - 2005: rev. 2 build
 - 2006: locomotion redesign (Proto Innovations LLC)



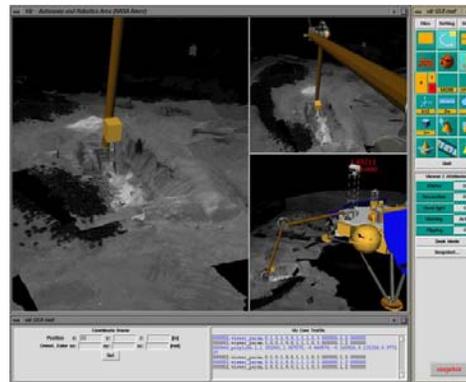


Research Areas

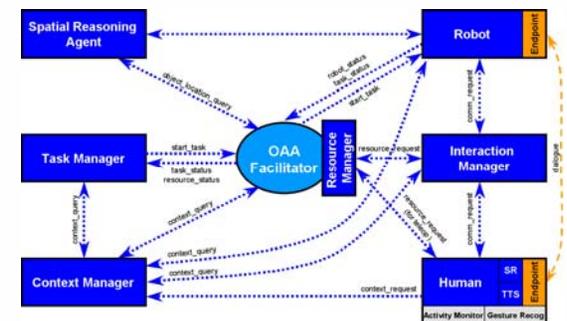
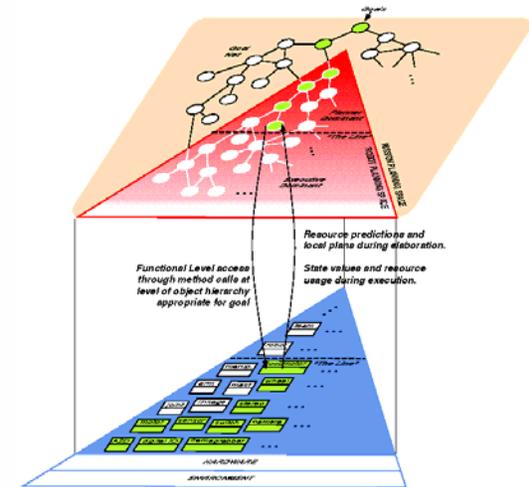
Perception



Interaction



Architecture





Robot Software Architectures

- Almost no SW-reuse in flight software so far
- And also in space robotics research

Rising demand for reusable robotics infrastructure:

- The rising complexity of scenarios does not allow reimplementing from scratch
- Complexity of future applications goes beyond the scope of a single research group
- Code reuse enhances software quality
- Code reuse allows focusing on the not yet solved problems



Why develop reusable software?

- To capture robotic domain knowledge
- To support development of generic algorithms
- To reduce the need for resolving recurring problems for every system
- To simplify integration of new technologies
- To use same framework for various robots
- Increase functionality by leveraging a more mature base



Middleware for Space Robotics?

- Middleware does hardly meet today's requirements of flight software and hardware
 - MHz, MB RAM
 - Software verification: No dynamic memory allocation, no callbacks (virtual methods), no large external libraries
- CLARAty-based demonstrators were flight hardened for use on MER-rovers



CLARAty

Coupled Layer Architecture for Robotic Autonomy

CLARAty is a unified and reusable **robotic software** that provides basic functionality and simplifies the integration of new technologies on various rovers and robotic platforms

- Multi-center project
JPL, ARC, CMU
- Supports various rover platforms
Fido, Rocky 7, Rocky 8, K9, K10, (K11)

<http://claraty.jpl.nasa.gov>

Slides in co-operation with Issa Nesnas, JPL

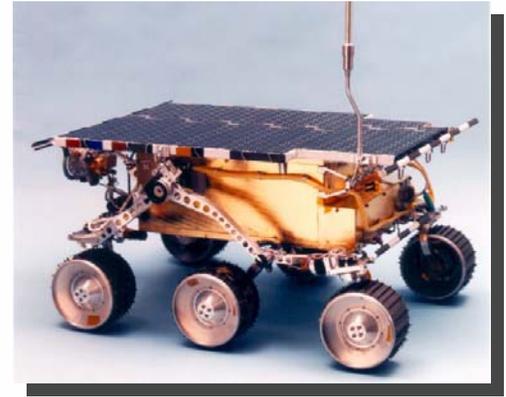
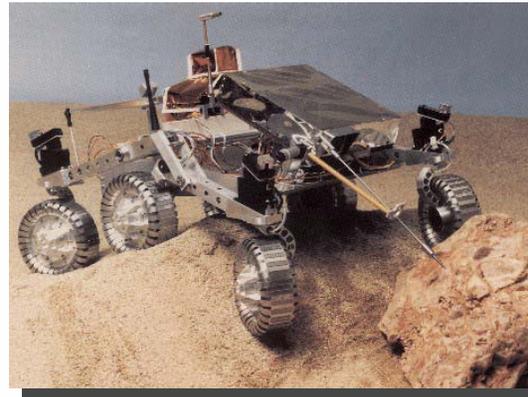
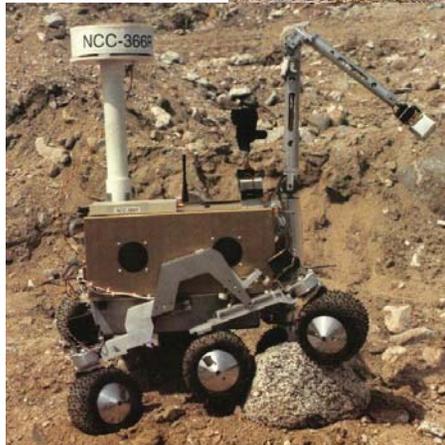


NASA Develops Various Rovers

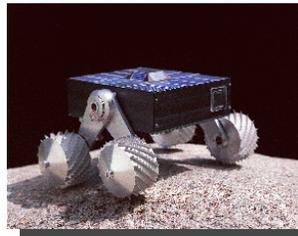
Large



Medium



Small



For research & flight



Would like to support ...



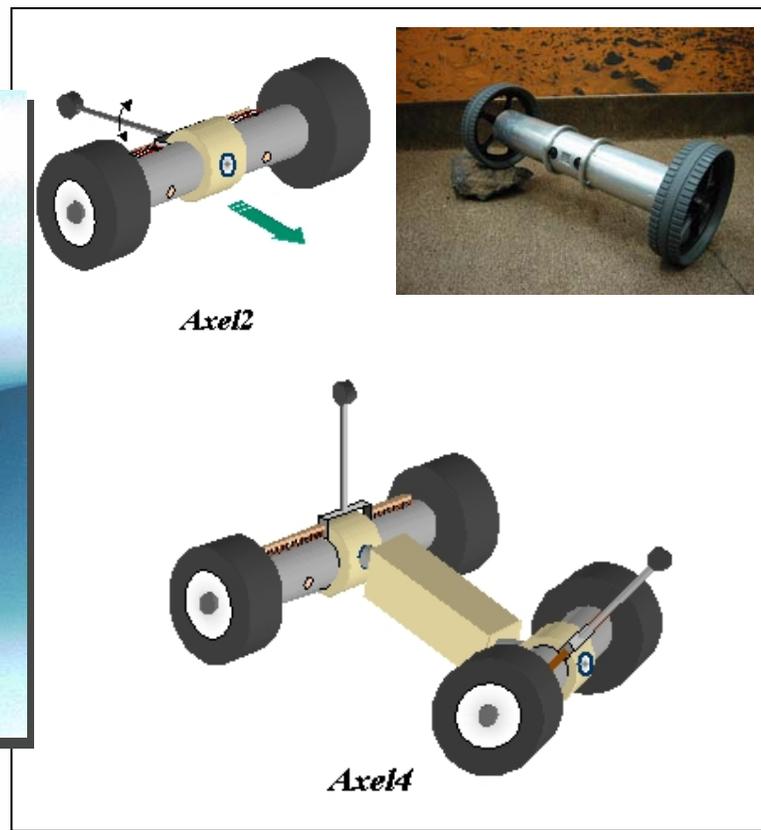
Custom Rovers



COTS Systems



Manipulators



Reconfigurable Robots



Problem and Approach

- Problem:
 - Difficult to share software/algorithms across systems
 - Different hardware/software infrastructure
 - No standard protocols and APIs
 - No flexible code base of robotic capabilities
- Objectives
 - Unify robotic infrastructure and framework
 - Capture and integrate legacy algorithms
 - Simplify integration of new technology
 - Operate heterogeneous robots
 - Mediate between research and flight requirements



Why is robotic software “hard”?

- Software:
 - Software is large and complex
 - Has lots of diverse functionality
 - Integrates many disciplines
 - Requires real-time runtime models
 - Has limited hardware resources - efficiency
 - Talks to hardware
- Hardware:
 - Physical and mechanics vary
 - Electrical hardware architecture changes
 - Hardware component capabilities vary



How?

- Study several robotic system implementations
- Study interactions of elements in various systems
- Identify reusable elements in robotic systems
- Identify implicit assumptions made
- Project potential advances to these elements
- Design a generic/flexible implementation of these elements
- Adapt to a number of robotic systems
- Test and study the limitations of the design
- Go back to design and iterate
- Modify/extend/redesign to address limitations and variability across systems

Your generic base is reusable



Approach

- Domain knowledge guides design
- Layers of abstraction help master complexity
- Abstractions also provide a classification of various technology elements
- Information hiding protects implementation variability
- Small modular components are more reusable than monolithic blocks
- Interfaces define behavior of various elements



Things to be aware of

- Over-generalizing leads to ineffectiveness
 - More general -> less functionality -> more work for results
 - Number of abstractions vs. complex hierarchies
 - Modular elements with strongly typed interfaces
 - Algorithm generality influences abstraction design
- Runtime models vary across systems
 - Challenges in combining hardware/firmware/software architectures in most effective manner
 - Need for both cooperative and pre-emptive scheduling



Goals

- Capture and integrate a wide range of technologies
- Leverage existing tools
- Leverage experience and tools of the larger software development community
- Apply appropriate design patterns to the domain
- Provide an infrastructure that enables rapid robotic development
- Capture experience of technologists implementations



Challenges in Interoperability

- Mechanisms and Sensors
- Hardware Architecture



Different Mobility Mechanisms

with different sensors

From wheeled Rocker-bogies with different steering

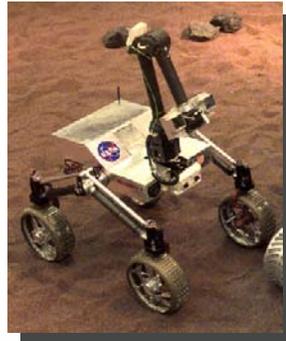
To wheels on articulated links

To inflatable wheels

From three wheelers

To four, six and even eight

From wheeled to legged



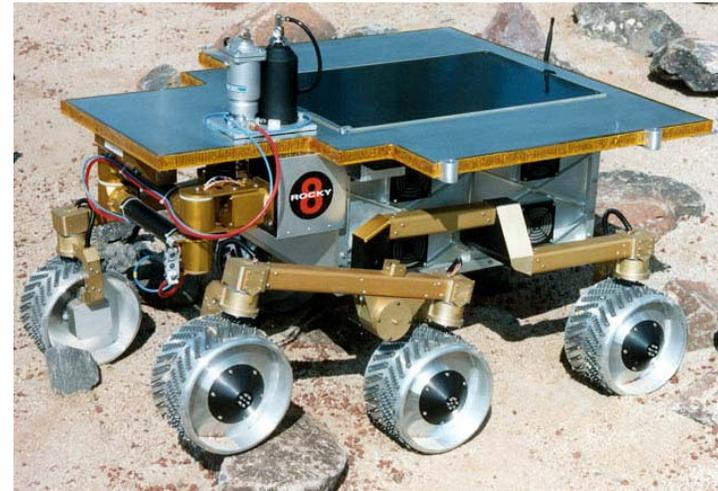


For Example: Wheeled Locomotion



Rocky 7

QuickTime™ and a
None decompressor
are needed to see this picture.



Rocky 8

QuickTime™ and a
Video decompressor
are needed to see this picture.

QuickTime™ and a
None decompressor
are needed to see this picture.



Reusable Wheeled Locomotion Algorithms

General flat terrain algorithms and specialized full DOF algorithms



ATRV (a)



Sojourner (d)



Rocky 7 (d)



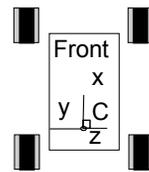
FIDO (e)



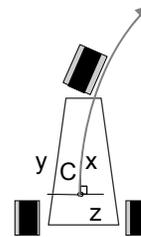
Rocky 8 (e)



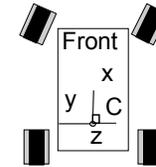
K9 (e)



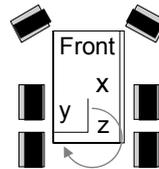
(a)
Skid Steering
(no steering wheels)



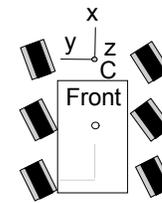
(b)
Tricycle
(one steering wheel)



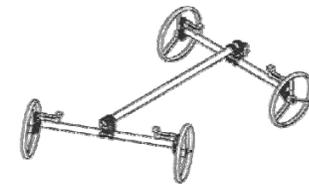
(c)
Two-wheel steering



(d)
Partially Steerable
(e.g. Sojourner,
Rocky 7)



(e)
All wheel steering
(e.g. MER, Rocky8,
Fido, K9)

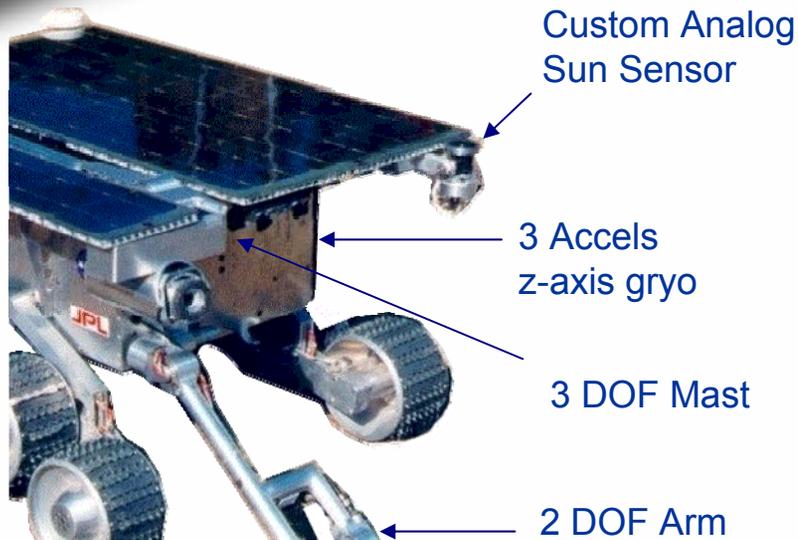


(f)
Steerable Axle
(e.g. Hyperion)

...

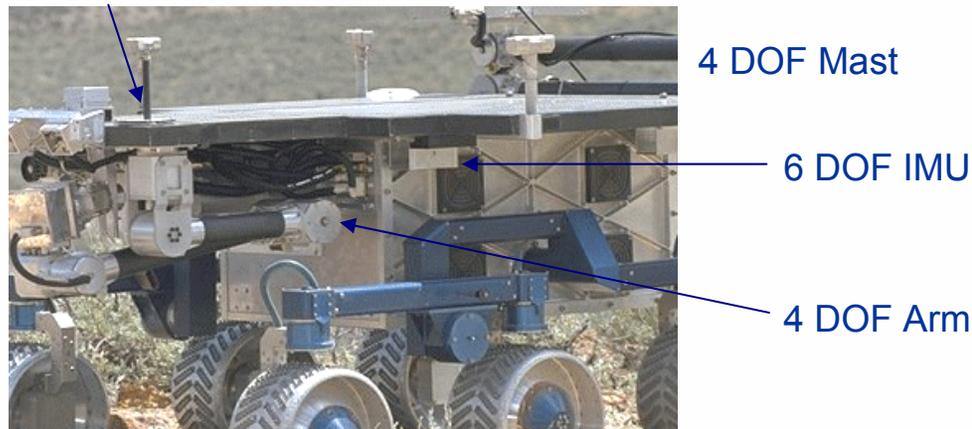


Manipulators and Sensor Suites



QuickTime™ and a None decompressor are needed to see this picture.

Camera Sun Sensor



- Given different capabilities, how much reuse can be achieved?

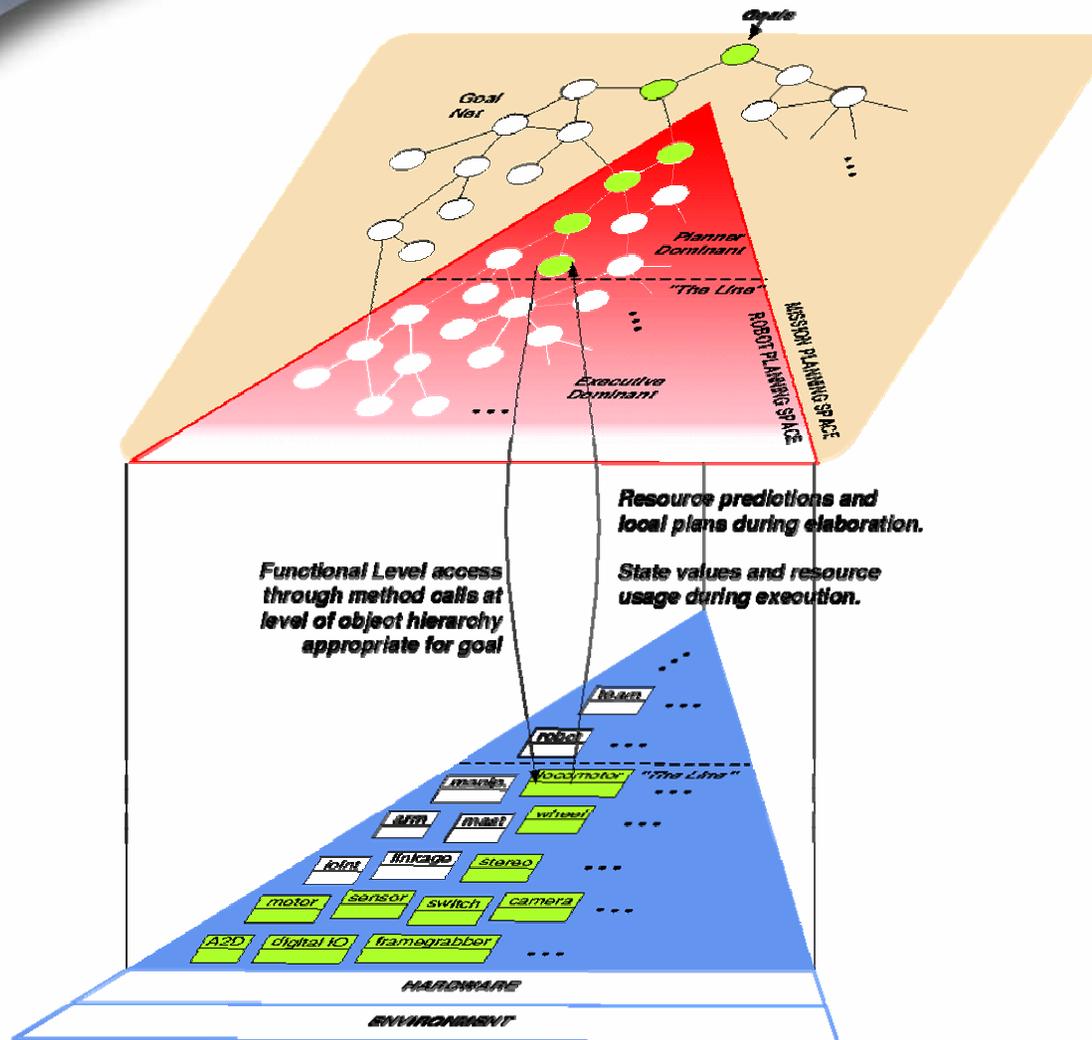


CLARAty Architecture



A Two-Layered Architecture

CLARAty = Coupled Layer Architecture for Robotic Autonomy



THE DECISION LAYER:

Declarative model-based
Mission and system constraints
Global planning

INTERFACE:

Access to various levels
Commanding and updates

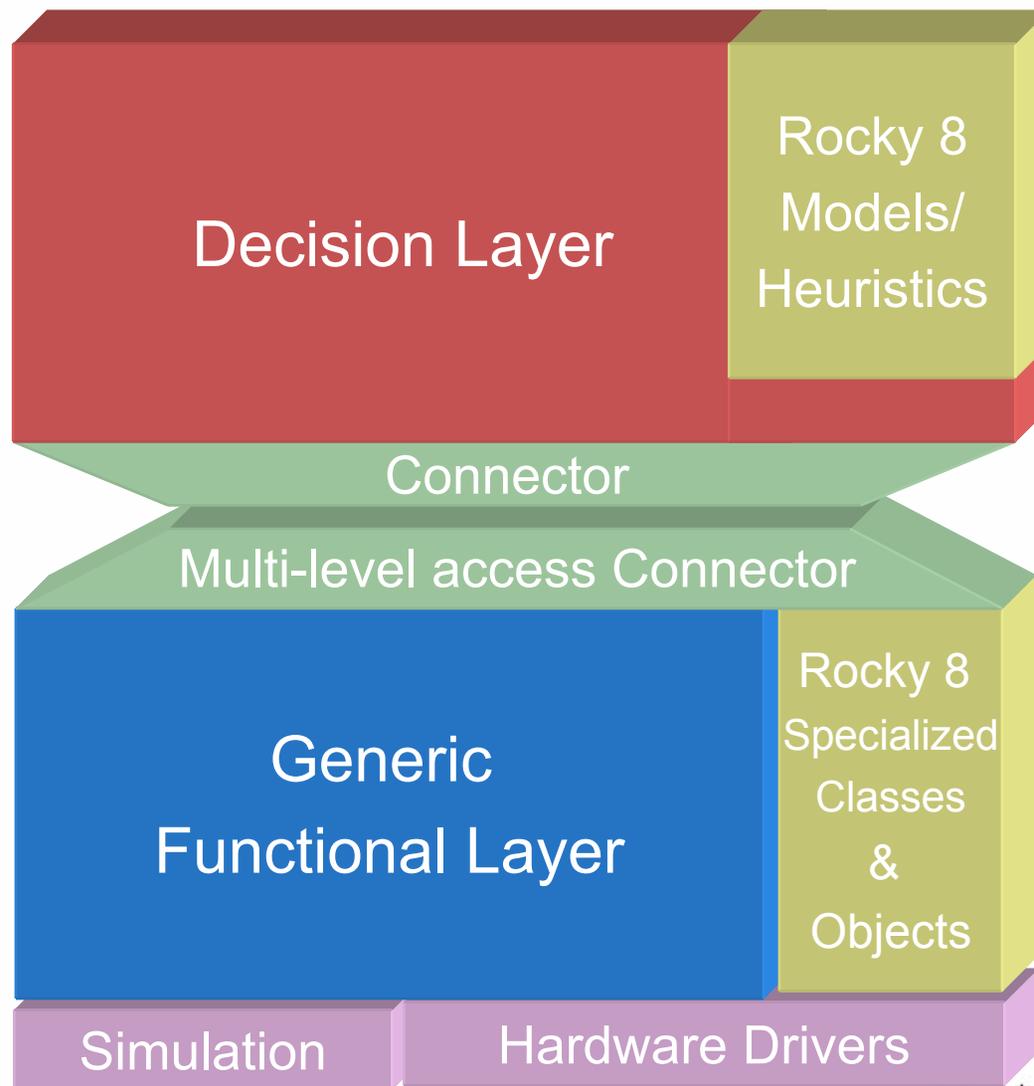
THE FUNCTIONAL LAYER:

Object-oriented abstractions
Autonomous behavior
Basic system functionality

Adaptation to a system



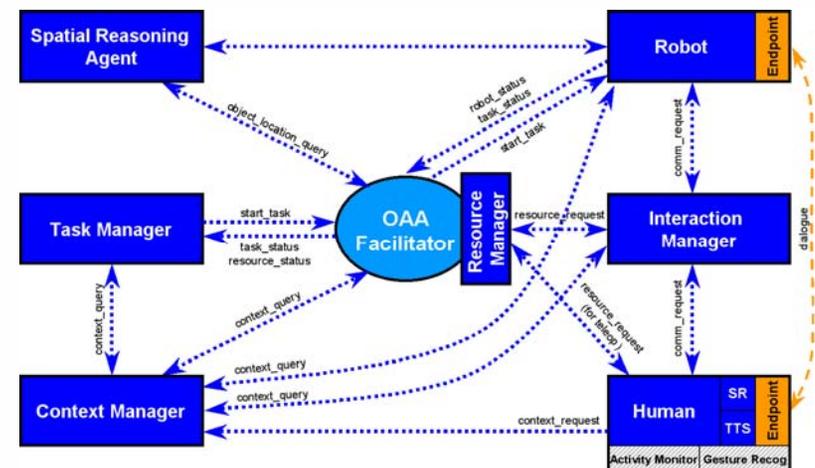
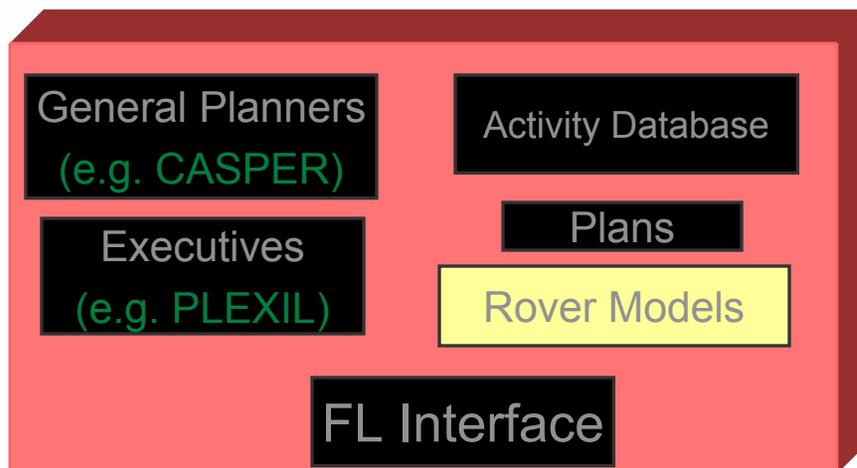
Adapting to a Rover





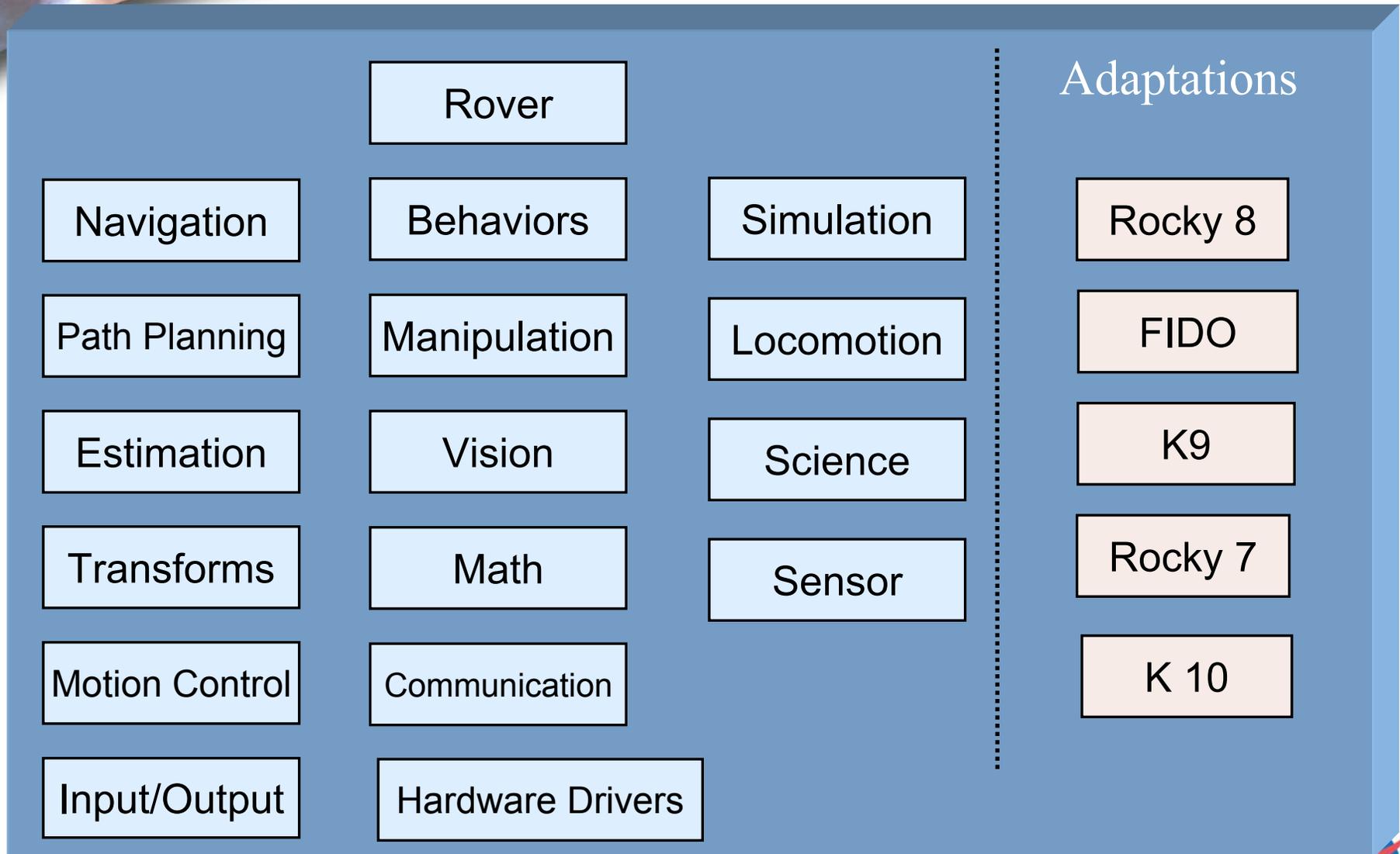
The Decision Layer

- The different application scenarios provide a high degree of variability in the design of the decision layer
- Plan centric vs. interactivity centered architectures have different requirements on the functional layer
- The DL/FL interfacing is therefore a critical part of the robot architecture



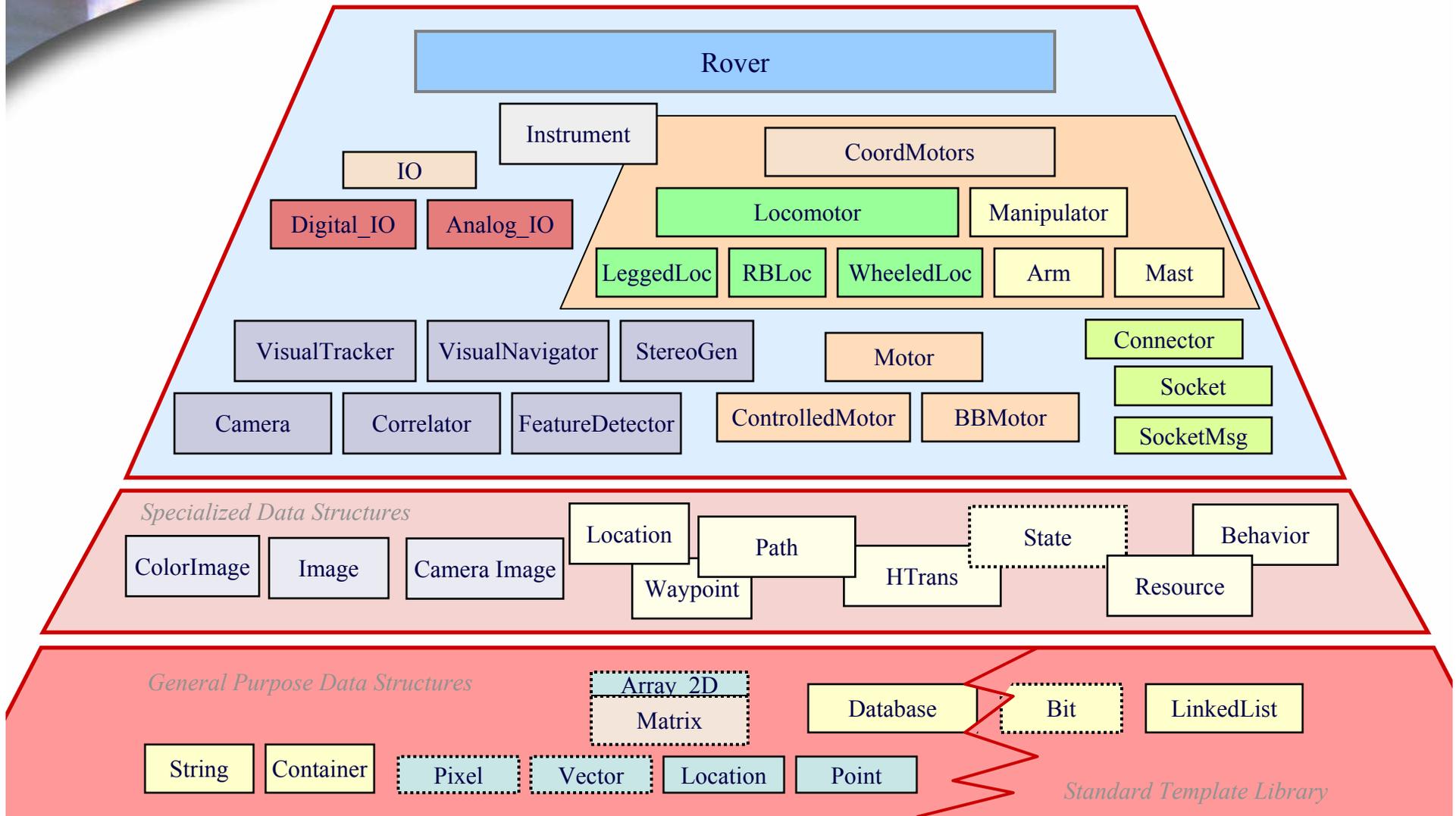


The Functional Layer





Functional Layer Components





Abstraction Models



CLARAty Abstractions

- Generic Physical Components (GPC)
 - Locomotor, Arm, Mast,
- Specialized Physical Components (SPC)
 - K9_Locomotor, K9_Arm, R8_Mast
- Generic Functional Components (GFC)
 - ObjectFinder, VisualNavigator, Stereovision, Localizer
- Specialized Functional Components (SFC)



Generic Technologies & Algorithms

- Technologies that are generic by design should not be constrained by the software architecture & implementation
- Non-generic technologies should be accommodated on the appropriate platforms
 - Example (**Generic**): if you are working in navigation, you would not care about H/W architecture difference among different rovers
 - Example (**Specific**): if you are doing wheel/terrain interaction research, you might require specific hardware which one of the vehicles would support
- Assumptions are made explicit



Wheel Locomotor Example



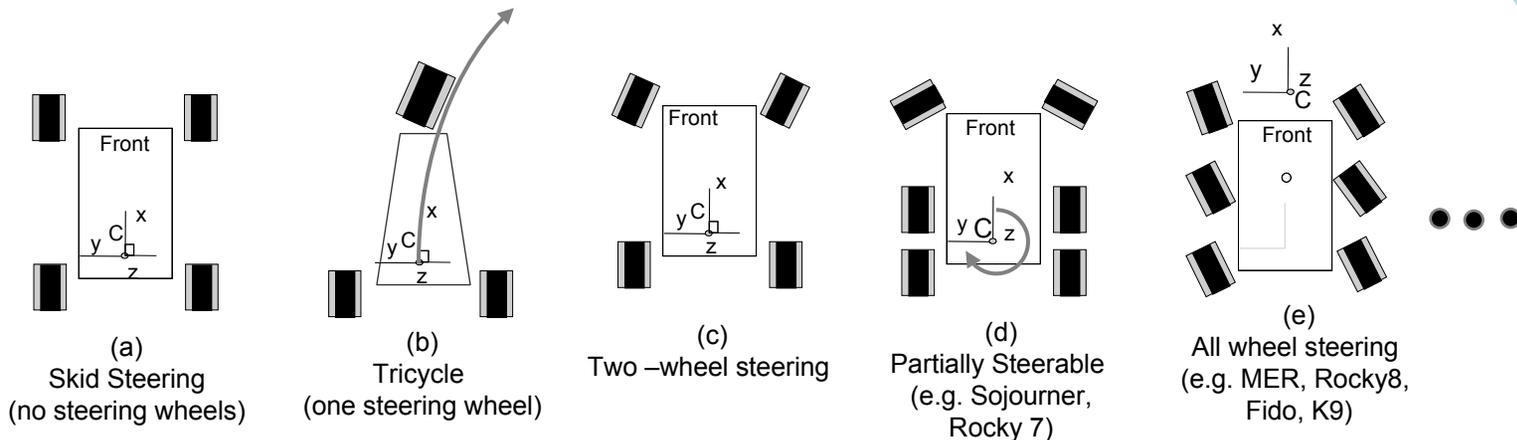
Capabilities of Wheel Locomotor

- Type of maneuvers:
 - Straight line motions (fwd / bkwd)
 - Crab maneuvers
 - Arc maneuvers
 - Arc crab maneuvers
 - Rotate-in-place maneuvers (arc turn $r=0$)
- Driving Operation
 - Non-blocking drive commands
 - Multi-threaded access to the Wheel_Locomotor class – e.g. one task can use Wheel_Locomotor for driving while the other for position queries
 - Querying capabilities during all modes of operation. Examples include position updates and state queries
 - Built-in rudimentary pose estimation that assumes vehicle follows commanded motion



Generic Reusable Algorithms

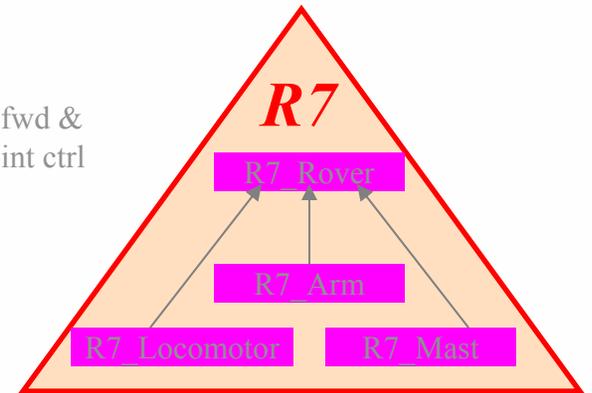
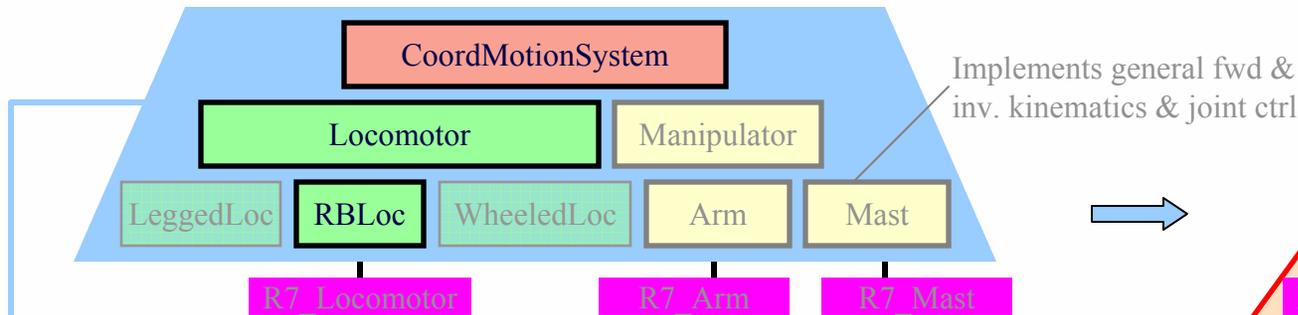
- **Wheeled Locomotion** – works for Rocky 8, Rocky 7, Fido, K9, ...





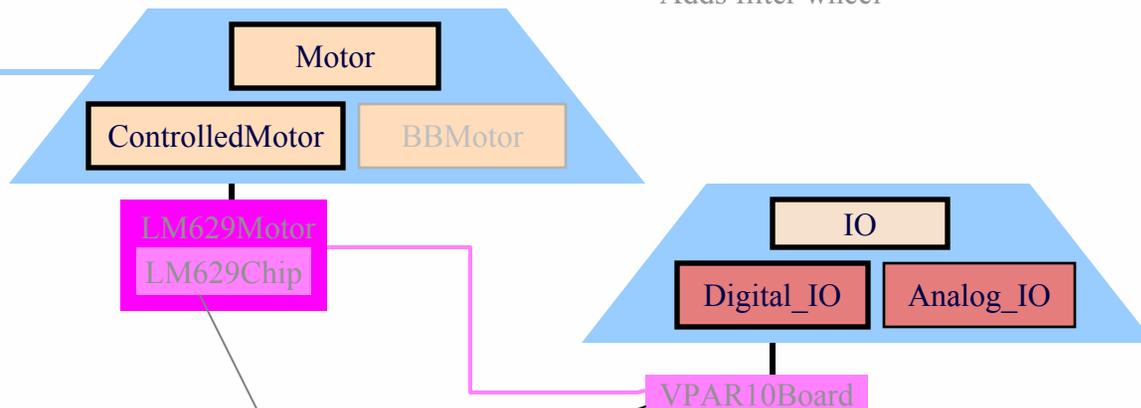
R7 Specific Rover Implementation

Non reusable Code Reusable Code



- Attaches proper motors
- Restricts Steering to 2 wheels

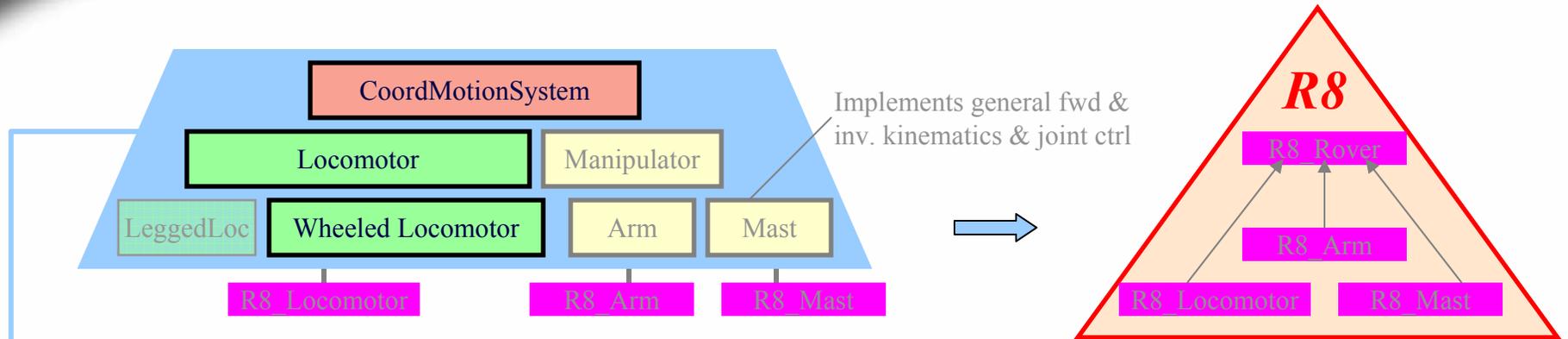
- Specialized inv. Kinematics (overrides default)
- Attaches proper motors
- Attaches proper cameras for mast
- Adds filter wheel





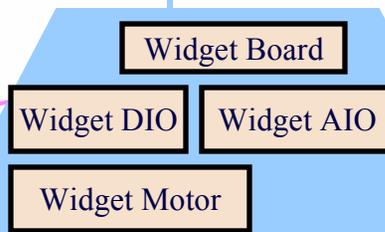
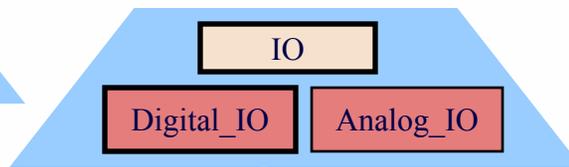
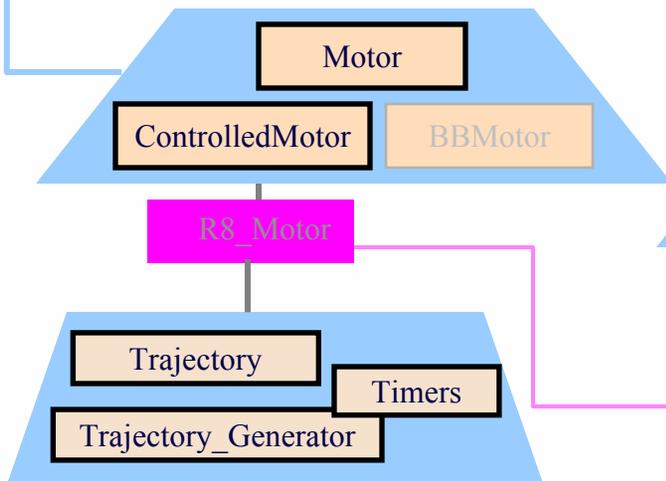
R8 Specific Rover Implementation

Non reusable Code Reusable Code



- Attaches proper motors
- Restricts Steering to 2 wheels

- Specialized inv. Kinematics (overrides default)
- Attaches proper motors
- Attaches proper cameras for mast
- Adds filter wheel



HCTL 1100 Chip





Code Reusability Results

Analysis of amount of reusable code across implementations:

Module	Lines of Code	Status	Depends On
Wheel Locomotor	1445	Reusable	Motion Sequence, 1D Solver, Homogeneous Transforms
Motion Sequence	540	Reusable	Vector
Matrix, Vector, Array	1083	Reusable	-
1D Solver	356	Reusable	-
Location, Homogeneous Transforms	341	Reusable	Rotation Matrix, Point 2D
Rotation Matrices	435	Reusable	-
Point 2D	131	Reusable	-
Controlled Motor	2080	Reusable	
Rocky 8 Locomotor	250	Non-reusable	Rocky 8 Motor
Rocky 8 Motor	334	Non-reusable	Widget Motor, etc...
Total	6995	584 (non-reusable)	
Total Reusable	~92%		

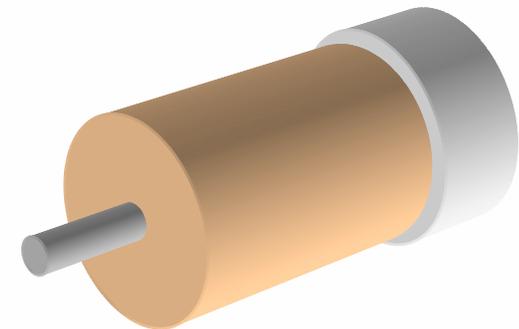
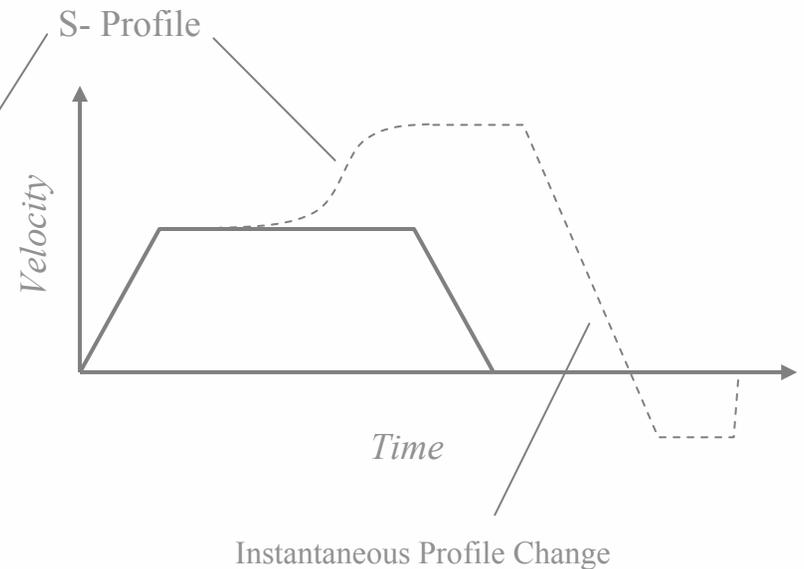


Motor Example



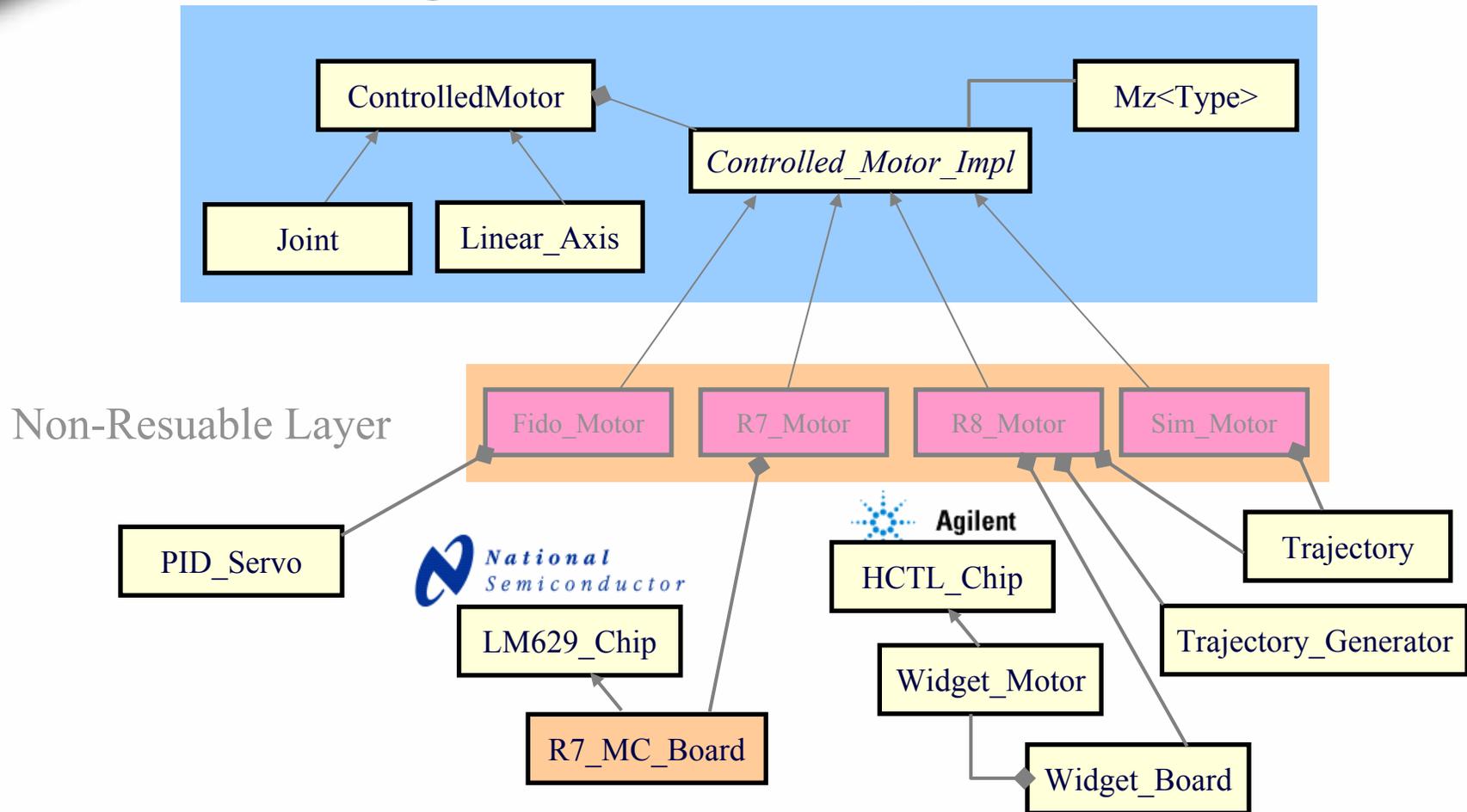
Example: Generic Controlled Motor

- Define generic capabilities independent of hardware
- Provide implementation for generic interfaces to the best capabilities of hardware
- Provide software simulation where hardware support is lacking
- Adapt functionality and interface to particular hardware by specialization inheritance
- Motor Example: public interface command groups:
 - Initialization and Setup
 - Motion and Trajectory
 - Queries
 - Monitors & Diagnostics





Comparing Different Implementations





Modules

- **Actual Examples of Code Reusability for Hardware modules:**
 - **Controlled Motor Hierarchies for Rocky 8 and Rocky7**

Rocky 8

Module	Lines of Code	Status
Controlled Motor	2080	Reusable
Trajectory Generator	338	Reusable
Resources (Timers, etc)	143	Reusable
Bits	756	Reusable
Rocky 8 Motor	334	Non-reusable
Widget Motor	383	Reusable –
Motor Controller HCTL	900	Reusable – HCTL
I2C Master	1446	Reusable – I2C
Total	6380	334 (non-reusable)
Total Reusable	~95%	
Total Reusable – Strict	~52%	

Rocky 7

Module	Lines of Code	Status
Controlled Motor	2080	Reusable
Bits	756	Reusable
Input Output	706	Reusable
Rocky 7 Motor	415	Non-reusable
Rocky 7 I/O Maps	131	Non-reusable
Motor Controller LM629	1014	Reusable – LM629
VPAR10 Parallel I/O	534	Reusable – VPAR10
Total	5636	546 (non-reusable)
Total Reusable	~90%	
Total Reusable – Strict	~63%	



Currently Supported Platforms



Rocky 8

- VxWorks
- Intel x86
- JPL

- Intel x86
- Linux
- Solaris CC
- CMU
- JPL AI



K9

- Linux
- Intel x86
- Ames



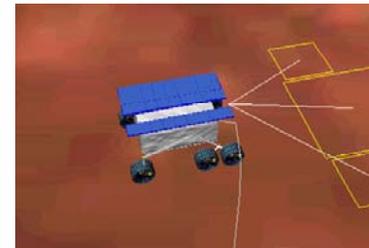
K10

- Linux
- Intel x86
- Ames



Rocky 7

- VxWorks
- Motorola 68K
- JPL



ROAMS

- Solaris/Linux
- JPL



Future Plans

- Open Source Release
- Flexible, freely configurable locomotion system and mechanisms model
- CORBA based functional Layer/Decision Layer interfacing
- Extend the architecture to the requirements of the challenges of Lunar Robotics



Mars Rovers

CLARAty design originates primarily in Marsian scenarios:

- Communication delay requires high amount of autonomy
- Very limited system resources
- No interaction possible

➤ An extremely autonomous mobile rover system

Scenario assumptions:

- Single rover in an otherwise static world
- Limited communication (up/down-link based)
- Operation speed vs. operation safety tradeoff

➤ Limited reactivity



Lunar Challenges

- High Bandwidth, low latency links
 - Extensive communication
 - Limited teleoperation
 - Supervisor feedback augments autonomy
- Multiple robots
 - Team communication
 - Multi-robot cooperation
- Human robot interaction
 - Safety requirements
 - Speed requirements
 - Reactivity requirements
 - Single actor assumption breaks
 - Flexible task allocation



Summary

- Space is a challenging application area for mobile robots
- CLARAty provides a repository of reusable software components at various abstraction levels
- It captures well-known robot technologies in a basic framework for researchers
- It allows researchers to integrate novel technologies at different levels of the architecture
- It is a collaborative effort within the robotics community
- It runs on multiple heterogeneous robots
- The lunar mission provides new challenges for CLARAty



Thank you very much for your attention!

Acknowledgements

CLARATy Team (multi-center)



Jet Propulsion Laboratory



Ames Research Center



Carnegie Mellon University