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
- Rover lab
- K10
  (4x)
- K11
- PER
  (10x)
- Scorpion
- Amtec
- Schunk
  (2x)
- Marscape
- Moonscape
- 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 reimplementation 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)


*Slides in co-operation with Issa Nesnas, JPL*
NASA Develops Various Rovers

Large

Medium

Small

For research & flight
Would like to support ...

- Custom Rovers
- Manipulators
- COTS Systems
- 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.
Reusable Wheeled Locomotion Algorithms

General flat terrain algorithms and specialized full DOF algorithms

(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)

ATRV (a) Sojourner (d)
Rocky 7 (d) FIDO (e)
Rocky 8 (e) K9 (e)
Manipulators and Sensor Suites

- Custom Analog Sun Sensor
- 3 Accels z-axis gyro
- 3 DOF Mast
- 2 DOF Arm

- Camera Sun Sensor
- 4 DOF Mast
- 6 DOF IMU
- 4 DOF Arm

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

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

- Decision Layer
  - Rocky 8 Models/Heuristics

- Connector

- Multi-level access Connector
  - Rocky 8 Specialized Classes & Objects

- Generic Functional Layer
  - Rocky 8 Specialized Classes & Objects

- Simulation
  - Hardware Drivers
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

- Navigation
- Path Planning
- Estimation
- Transforms
- Motion Control
- Input/Output
- Rover
- Behaviors
- Manipulation
- Vision
- Math
- Communication
- Hardware Drivers
- Simulation
- Locomotion
- Science
- Sensor

Adaptations
- Rocky 8
- FIDO
- K9
- Rocky 7
- K 10
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, ...

(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)
R7 Specific Rover Implementation

- CoordMotionSystem
  - Locomotor
  - Manipulator
  - LeggedLoc
  - WheeledLoc
  - Arm
  - Mast

- Motor
  - ControlledMotor
  - BBMotor
  - LM629Motor
  - LM629Chip
  - Device Drivers

- IO
  - Digital_IO
  - Analog_IO
  - VPAR10Board

- Implement general fwd & inv. kinematics & joint ctrl
  - R7_Mast
  - R7_Arm
  - R7_Locomotor
  - R7_Motor

- Non reusable Code
  - Reusable Code

- Attach 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

- **Locomotor**
  - Attaches proper motors
  - Restricts Steering to 2 wheels

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

- **CoordMotionSystem**
  - Implements general fwd & inv. kinematics & joint ctrl

- **Motor**
  - R8 MOTOR

- **ControlledMotor**
  - BBMotor

- **Arm**
  - R8 Arm

- **Mast**
  - R8 Mast

- **LeggedLoc**
  - R8 Locomotor

- **Wheeled Locomotor**

- **Manipulator**
  - R8 Arm

- **Arm**
  - R8 Arm

- **Mast**
  - R8 Mast

- **Motor**
  - R8 Motor

- **ControlledMotor**
  - BBMotor

- **IO**
  - Digital_IO
  - Analog_IO

- **Widget Board**
  - Widget DIO
  - Widget AIO
  - Widget Motor

- **HCTL 1100 Chip**

- **Non reusable Code**

- **Reusable Code**
## Code Reusability Results

### Analysis of amount of reusable code across implementations:

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

ControlledMotor
Joint
Linear_Axis

Controlled_Motor_Impl

Mz<Type>

Non-Resuable Layer

Fido_Motor
R7_Motor
R8_Motor
Sim_Motor

PID_Servo

LM629_Chip

HCTL_Chip

Widget_Motor

Trajectory

Trajectory_Generator

Widget_Board

R7_MC_Board
### Actual Examples of Code Reusability for Hardware modules:

- Controlled Motor Hierarchies for Rocky 8 and Rocky 7

#### Rocky 8

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

#### Rocky 7

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

**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
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.
Acknowledgements

Thank you very much for your attention!

CLARAty Team (multi-center)

Jet Propulsion Laboratory

Ames Research Center

Carnegie Mellon University