Reliable Robotics – Diagnostics++

Dominik Kirchner, Daniel Saur

(Distributed System Group, University of Kassel)
Story board

♦ Motivation

♦ Introduction to ROS diagnostics

♦ ROS diagnostics extensions

♦ Takeaways
Robot Failures are facts ...
Robot Failures are facts ... 

But what to do about it?
What can we do about it?

- We have to correct the problems or avoid the failure.

- **Problem:** You need to know the root cause before you can correct the problem

- **Situation (without a diagnostic support):**
  - Limited to visible observations
  - Limited to developer experiences

- **Approach (with ROS diagnostics):**
  - Reduce subjective assessment
  - Provide objective data of the state of the system
  - Filter the data to extract the problem
  - Notify the operator
How ROS diagnostics fits in – A short Overview

Monitoring
- diagnostic_updater
- self_test

Aggregation/Diagnosis
- diagnostic_aggregator
- Analyzer A
- Analyzer B
- Analyzer C

Presentation
- rxconsole
- robot_monitor
- PR2_dashboard

diagnostic

Presentation
How to provide current status data?

- **diagnostic_updater** provides run-time data for the node’s state
- Supports to embed this code in the node
- Unified interface (topic *diagnostics*, weakly typed message fields)
How to diagnose a problem?

- Main purpose is to analyzes diagnostic data for presentation
- Configurable set of analyzers (plug-ins)
- Analyzers are useful for tasks like, grouping, suppressing invalid outputs, ...

```cpp
bool ArmControl::match(const std::string name) {
    if (name == this->nodeToAnalyze) return true;
    return false;
}

bool ArmControl::analyze(const boost::shared_ptr<StatusItem> item) {
    if (item->getName() == this->nodeToAnalyze) {
        if (item->hasKey("internal_state_val")) {
            l = boost::lexical_cast<int>(item->getValue("internal_state_val"));
            // decision rules ...
            if (l > 5) this->ruleViolation[0] = 1;
        } else
            ROS_INFO("internal_state_val not in msg");
    }
    return true;
}

vector<boost::shared_ptr<DiagnosticStatus>> ArmControl::report() {
    vector<boost::shared_ptr<DiagnosticStatus>> output;
    /* create and fill the DiagnosticStatus item */
    output.push_back(ds);
    return output;
}
```
How to present the diagnostic results?

- Present the state of the system on a quick view
- Detailed information are accessable in a few steps
REP 107: Diagnostic System

- ROS diagnostics is proposed:
  - to provide operator awareness,
  - to target hardware drivers only,
  - to have a default update interval (1Hz),
  - not to react to failures.

- The goal is operator awareness
  - Results to correct faults afterwards

- Diagnostics is targeted for hardware drivers only
  - “... adding diagnostics to all software components creates too much noise ...”
  - “... the burden of logging and analyzing goes up significantly.”
Problem solved?

♦ ... at least diagnostics is the right thing to start

♦ Limitations and shortcomings:
  ♦ High integration efforts (hand-coded parts)
  ♦ Limited diagnostic scope (software components)
  ♦ No generic monitoring support (e.g. third-party modules)
  ♦ Static update intervals

♦ For improvement we propose some extensions
  ♦ **Generic monitoring** to extend the monitoring scope
  ♦ **Model based integration support** to limit the hand coded parts
  ♦ **Reactivity** to trigger fail-safe and repair functions
Generic Monitoring: How to extend the scope?

♦ **Robot = Software + Hardware**

♦ **Hardware:**
  ♦ Driver nodes are limited to provided data
  ♦ Some hardware provides diagnostic data
  ♦ ... but many do not

♦ General hardware diagnostic can not address application specific requirements

♦ **Generic hardware diagnostic agent:**
  ♦ sense electrical values (voltage, current)
  ♦ sense physical values (temperature, light, sound, ...)
  ♦ perform (simple) reactions
Generic Monitoring: How to extend the scope?

- Only driver nodes are proposed to be monitored (REP 107)
- But the system consists of further components
  - Non-driver ROS nodes
  - Third party nodes (e.g. nodes with libraries)
  - Communication links (e.g. depended failures [1])
- Extend the monitoring scope
  - Generic OS information for black-box nodes
  - Data flow monitoring of depended nodes
- Reduce the monitoring overhead
Adaptive Monitoring: How to limit the overhead?

- Overhead increases with the extended scope
- **Individual priority levels** to control monitoring behavior for each node
- **Adaptation** of the monitoring
  - update rates, history length, mean, variance, ...
  - Limited to a fixed range
- **Properties**
  - Types: minimum, maximum, range, histogram
  - Parameters: e.g. the optimal values
- Derivation metric defines the relative distance from the optimum
- Significant reduction of the overhead
Generic Monitoring: How to realize the passive monitoring?

- External component for passive monitoring
- Poll generic process characteristics for *black-box* nodes
- Sniff communication for data flow analysis
- Design overview
  - Characteristics to manage the monitoring
  - Observer to poll the process information
  - Properties to specify expected values
  - Model to set up the needed characteristics
Integration Support

♦ ROS diagnostic relies on manual code and configuration parameters

♦ Generic monitoring introduced even more parameters

♦ Data flow monitoring needs an architectural model

♦ Central **robot model** to unify architectural information and configuration settings

♦ Tree presentation of the architectural composition
  ♦ **Robot**: for global settings
  ♦ **Capability**: a semantic grouping for a task relevant robot features
  ♦ **Functionality**: a system node
  ♦ **Channel**: a topic between system nodes

![Diagram of architectural composition](image)
Integration Support: Robot Model

```xml
<?xml version="1.0" encoding="utf-8"?>
<robot name="tasmania" type="Gen10">
  <cap name="grip" id="2" prio="5" used_in_role="WM09/WM09_Attacker:1">
    <chan name="GripControl" channel="gripCmd">
      <func name="ArmControl" prio="2" working_dir="%ES_ROOT%/ArmControl/bin"
        filename="ArmControl" arguments="">
        <charac name="cpuLoad" type="CpuLoad"
          proptype="Range" minvalue="0" maxvalue="200"/>
        <charac name="memUsage" type="MemUsage"
          proptype="Range" minvalue="7000" maxvalue="7500"></charac>
        <charac type="ThreadUsage"
          proptype="Range" minvalue="4" maxvalue="10"></charac>
      </func>
    </chan>
  </cap>
  ...
</robot>
```

- Intuitive prioritization through hierarchical model structure
- Arbitrary configuration blocks, like a generic monitoring configuration
- XML description as proof of concept
Integration Support: soft diagnosis

- Application specific hand-coded rules in the analyzers
- But often we do not know exactly these rules
  - Uncertainty
  - Lack of knowledge
- Reduce the burden of expert knowledge by methods of Soft Computing (Bayesian Networks)
- Dynamic Bayesian Networks to introduce temporal behavior
- Graphical modeling support through the SMILE/GENIE framework [2]
Autonomous Reactions: An Outlook

♦ Complement operator awareness with autonomous reactions

♦ MAPE-K cycle as the decision cycle

♦ Work in progress:
  ♦ Generic monitoring
  ♦ Diagnostic: Analyzers using soft diagnosis
  ♦ Knowledge: robot model to ease the integration and configuration

♦ Future work:
  ♦ Recovery planning based an diagnostic results
  ♦ Repair execution

Takeaways

♦ ROS diagnostics is a great tool, use it!

♦ We identified some shortcomings:
  ♦ Limited monitoring scope
  ♦ Integration support

♦ We proposed some extensions to overcome these limits
  ♦ Adaptive monitoring with individual priorities
  ♦ Model support to ease the integration
  ♦ Dynamic Bayesian Networks for generic analysis

♦ We presented an outlook of a autonomous reactive system
Thank you
References


