

INTO THE ROS

ADVANCED ROS NETWORK INTROSPECTION

Praxis der Softwareentwicklung
Sommersemester 2014

P f l i c h t e n h e f t



Auftraggeber

KIT - Karlsruher Institut für Technologie
Fakultät für Informatik
Institut für Anthropomatik und Robotik (IAR)
Intelligente Prozessautomation und Robotik (IPR)

Betreuer: Andreas Bihlmaier
andreas.bihlmaier@gmx.net

Auftragnehmer

Name	E-Mail-Adresse
Alex Weber	alex.weber3@gmx.net
Matthias Hadlich	matthias.hadlich@student.kit.edu
Matthias Klatte	matthias.klatte@go4more.de
Micha Wetzel	micha.wetzel@student.kit.edu
Sebastian Kneipp	sebastian.kneipp@gmx.net

Karlsruhe, 07.06.2014

Inhaltsverzeichnis

1	Einleitung	4
2	Zielbestimmung	5
2.1	Musskriterien	5
2.2	Wunschkriterien	5
2.3	Abgrenzungskriterien	6
3	Produkteinsatz	7
3.1	Anwendungsbereiche	7
3.2	Zielgruppen	7
3.3	Betriebsbedingungen	7
4	Produktumgebung	8
4.1	Software	8
4.1.1	Implementierung	8
4.1.2	Dokumentation	8
4.2	Lizenz	8
5	Funktionale Anforderungen	9
5.1	Gesamtsystem	9
5.1.1	Pflicht	9
5.1.2	Optional	9
5.2	Soll-Spezifikation	10
5.2.1	Pflicht	10
5.2.2	Optional	10
5.3	API	10
5.3.1	Pflicht	10
5.3.2	Optional	10
5.4	GUI	11
5.4.1	Pflicht	11
5.4.2	Optional	11
6	Nichtfunktionale Anforderungen	12
6.1	Produktleistungen	12
6.1.1	Pflicht	12
6.1.2	Optional	12
6.2	Qualitätsanforderungen	12

6.2.1	Pflicht	12
6.2.2	Optional	13
7	Produktdaten	14
7.1	Config	14
7.2	Metadaten	14
8	Systemmodell	15
8.1	Szenarien	15
8.2	Anwendungsfälle	16
9	Benutzeroberfläche	18
9.1	Merkmale der graphischen Benutzeroberfläche	18
9.2	Protoypen/Skizzen	19
9.2.1	ÜbersichtsWidget	19
9.2.2	Auswahl Widget	21
9.2.3	Listen Widget	22
9.2.4	Node Graph	23
10	Qualitätsanforderungen	25
10.1	Erweiterbarkeit	25
10.2	Wartbarkeit	25
10.3	Zuverlässigkeit	25
11	Testfälle und Testszenarien	26
11.1	Tests	26
11.2	Beispielszenarien	26
12	Glossar	29
12.1	Allgemein	29
12.2	ROS	29

1 Einleitung

Das Robot Operating System (ROS) ist eine Middleware, um flexible Robotik Software zu schreiben. 2007 begann Willow Garage mit der Entwicklung von ROS, welches heute in der Robotik Community weit verbreitet ist. Es liefert eine Sammlung von Tools, Libraries und Konventionen, welche das Erstellen von komplexer und robuster Robotik Software auf verschiedenen Plattformen erleichtern soll. ROS gliedert die Algorithmen zur Realisierung intelligenter Roboter in einzelne Aufgaben, welche durch jeweils einen Prozess (Node) dargestellt und auf mehrere Rechner (Hosts) verteilt sein können. Die Nodes verbinden sich nach dem Publisher-Subscriber Schema über ein Netzwerk miteinander, um die Datenflüsse herzustellen (ROS Graph).

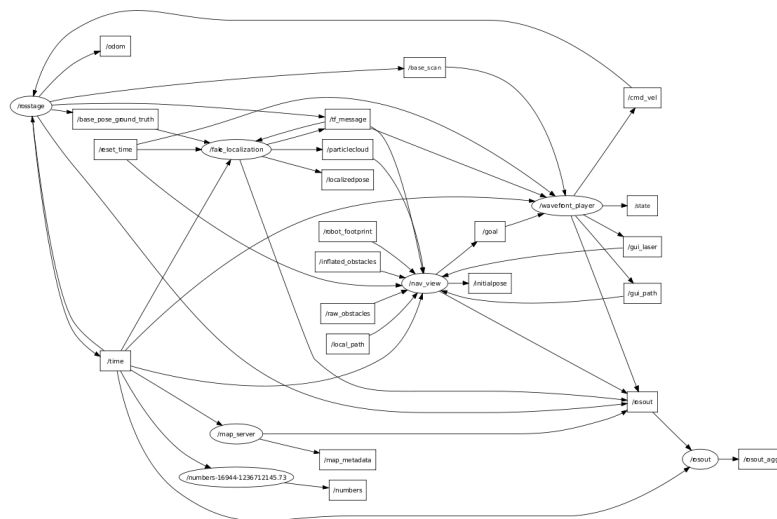


Abbildung 1.1: Beispiel eines ROS Graph. Nodes sind als Ellipsen, Topics als Rechtecke und Datenfluss als Pfeile dargestellt. Quelle: <http://www.willowgarage.com>

Ein Problem stellt die Überwachung dieses Systems dar. Zwar besteht im Moment die Möglichkeit, das Gesamtsystem auf Fehler zu überprüfen, jedoch ist es nicht möglich zu bestimmen, welcher Knoten sich fehlerhaft verhält. Dies kann besonders bei großen System mit hunderten von Knoten zum Problem werden. Um ein schnelles Debuggen zu ermöglichen, bedarf es einer Software zum dezentralen Erfassen der Fehlerursachen.

Unser Ziel als PSE-Team ist es, ein System zur Definition des Soll-Zustandes und zum Überwachen des Ist-Zustandes individueller Knoten zu erstellen.

2 Zielbestimmung

Die zu entwickelnde Softwarelösung soll sich als Überwachungsinstrument in die Robot Operating System (ROS) Middleware einfügen. Dabei soll überwacht werden, ob jeder Prozess, der im ROS-Netzwerk enthalten ist, definierten Sollwerten entsprechend, ordnungsgemäß Daten übermittelt.

Gerade durch die gegebene Problematik, eines auf mehrere Hosts verteilten Systems, bereitet das Erfassen und Verstehen von auftretenden Problemen, zum aktuellen Stand des ROS, Schwierigkeiten. Das vorliegende Projekt erfasst den Ist-Zustand der Prozesse im ROS-Netzwerk, ohne Leistungseinbußen, kontinuierlich und visualisiert diesen im Vergleich zu definierten Soll-Werten. Auftretende Fehler und Abweichungen von den Spezifikationen, werden dem Benutzer deutlich angezeigt. Es lassen sich Gegenmaßnahmen für verschiedene Fehler definieren.

2.1 Musskriterien

- Minimale Erweiterung bestehender Knoten zur Übermittlung von Metadaten
- Publizierung der Metadaten mit definierter Frequenz auf Topics
- Deaktivierung der Datenerfassung einzelner Knoten
- Definition von Soll-Zuständen
- Graphische Übersicht über Abweichungen von Soll- und Ist-Zustand der Knotendaten
- Statuserfassung der Host-Computer

2.2 Wunschkriterien

- Überwachung weiterer ROS Bestandteile: Service, Parameters
- Selbstkontrolle der Knoten
- Ist-Zustand als Soll-Zustand definieren
- Ergänzende Übersichten mit dem ROS Graphen und Plots über den zeitlichen Verlauf

2.3 Abgrenzungskriterien

- Das Projekt wird keine vollständige Automatisierung in Fehlerfällen umsetzen.
- Die Metadatenerfassung umfasst nur Knoten die dafür modifiziert werden.
- Es werden keine ausführlichen Diagnosen¹² aufgrund des Ist-Zustandes gestellt.

¹S. Zaman, G. Steinbauer, J. Maurer, P. Lepej und S. Uran. An Integrated Model-Based Diagnosis and Repair Architecture for ROS-Based Robot Systems. Erschienen in 2013 IEEE ICRA

²V. Monajjemi, J. Wawerla und R. Vaughan. "Drums": a Middleware-Aware Distributed Robot Monitoring System. Erschienen in 2014 Canadian Conference on Computer and Robot Vision

3 Produkteinsatz

3.1 Anwendungsbereiche

Angewandt wird die Software zum Suchen von Fehlern und Überwachen kompletter ROS Netzwerke.

3.2 Zielgruppen

Die Zielgruppe besteht aus Entwicklern und Administratoren von Umgebungen, in denen ROS eingesetzt wird. Es wird technisches Verständnis der Nutzer vorausgesetzt.

3.3 Betriebsbedingungen

- Zur Fehlererkennung wird die Software gestartet, um eine genauere Ursache des Fehlers festzustellen.
- Während des Produktivbetriebs wird die Überwachung zugeschaltet, um Versagen in dem System zu erkennen und automatische Gegenmaßnahmen einzuleiten.

4 Produktumgebung

4.1 Software

4.1.1 Implementierung

- GNU/Linux (Ubuntu 12.04) oder neuer
- ROS Hydro oder neuer
- Python 2.7.3
- QT 4.8
- PyQT 4.9.1

4.1.2 Dokumentation

- Latex
- Pydoc oder Sphinx

4.2 Lizenz

Das Projekt wird BSD-Lizensiert. Sollte das durch den Einsatz bestimmter Bibliotheken nicht möglich sein, wird die nächstmögliche freie Open-Source Lizenz, welche sich mit der Bibliothek vereinbaren lässt, eingesetzt.

5 Funktionale Anforderungen

5.1 Gesamtsystem

5.1.1 Pflicht

/FA0100/ Dezentrale Erfassung von Metadaten: Anzahl Topics und Subscriber, Bandbreite, Frequenz, Latenz, Jitter

/FA0200/ Monitoring Knoten zum zentralen Abgleich des Soll- und Ist-Zustandes

/FA0300/ Countermeasure Knoten um Gegenmaßnahmen bei Fehlern ergreifen zu können

/FA0310/ Warnungen und Fehlernachrichten bei maßgeblichen Abweichungen

/FA0400/ Definition eines ROS Message Types für Metadaten

/FA0500/ Eigenständiger Knoten zur Überwachung der Hardware des Host-Systems (CPU Auslastung, CPU Temperatur, RAM, Festplatten-Speicher)

5.1.2 Optional

/FA0600/ Lebenszeichen von Subscribern und Hosts, auch wenn keine Daten Empfangen werden

/FA0700/ Überwachung weiterer ROS Komponenten wie Services und Parameters

/FA0800/ Überwachung der Publisher

/FA0810/ Erfassung der Anzahl Publisher pro Topic

/FA0820/ Erfassung von Lebenszeichen von Publishern

/FA0900/ Festlegen des Ist-Zustandes als Soll-Definition

/FA0910/ Korrelationen zwischen Empfangs- und Sendeverhalten eines Knotens erkennen

/FA1000/ Anpassung des Sendeverhaltens des Systems an Netzwerkgegebenheiten

/FA1100/ Definition der Netzwerktopologie sowie Darstellung der Auslastung physischer Verbindungen

/FA1200/ Integration mit roswtf

5.2 Soll-Spezifikation

5.2.1 Pflicht

/FA1300/ Parametrisierung für Topics, Hosts und Knoten getrennt

/FA1310/ Obere und untere Schranken für Metadaten definierbar

/FA1400/ Laden der Soll-Spezifikation einmalig bei Start des Überwachungsknotens

/FA1410/ Weitergabe der Soll-Spezifikation an den ROS Parameter Server bei Knotenstart

/FA1500/ Möglichkeit nur Teile des Netzwerkes zu überwachen

/FA1510/ Teilsysteme können sich überlappen

5.2.2 Optional

/FA1600/ Ein Monitoringknoten kann unterschiedliche Soll Spezifikationen verwalten, vorausgesetzt diese widersprechen sich nicht

/FA1700/ Gegenmaßnahmen in der Soll Spezifikation definierbar machen

5.3 API

5.3.1 Pflicht

/FA1800/ Die Metadaten werden durch Ableiten des Subscribers erfasst (Alle Knoten, die auf einem Topic "subscriben" wollen, müssen wiederum von dieser abgeleiteten Klasse erben)

/FA1900/ Die Metadaten werden auf einem Topic mit definierter Frequenz publiziert

/FA2000/ Die Metadatenerfassung lässt sich über Parameter deaktivieren

5.3.2 Optional

/FA2100/ Knoten sind in der Lage, sich anhand ihrer Soll-Metadaten selber zu überwachen

/FA2200/ Das System wird auch für C++ Knoten implementiert

/FA2300/ Metadaten können auch durch einen Funktionsaufruf erfasst werden

5.4 GUI

5.4.1 Pflicht

/FA2400/ Die grafische Benutzeroberfläche bietet eine Visualisierung des Soll-Ist-Vergleichs von Knoten als auch Hostsystemen

5.4.2 Optional

/FA2500/ Geeignete Metadaten werden im Knoten-Graphen visualisiert (verfügbare Bandbreite, Datenrate, Latenz) und regelmäßig aktualisiert

/FA2600/ Geeignete Metadaten werden in den Widgets der graphischen Oberfläche dargestellt

/FA2700/ Es wird ein zeitlicher Verlauf der Metadaten aufgezeichnet und grafisch dargestellt

/FA2800/ Knoten werden (beispielsweise im Listen Widget) nach ihrem Host gruppiert angezeigt

/FA2900/ Knoten werden als Gruppen der unterschiedlichen Soll-Spezifikationen angezeigt

6 Nichtfunktionale Anforderungen

6.1 Produktleistungen

6.1.1 Pflicht

/NF0100/ Einfaches einbinden der Metadatenerfassung in bereits vorhandene Knoten

/NF0200/ Geringen zusätzlichen Leistungsbedarf durch die Metadatenerfassung

/NF0210/ Vernachlässigbarer Leistungsbedarf wenn die Erfassung deaktiviert ist

/NF0300/ Stufenweise Selbstüberwachung des Gesamtsystems

/NF0310/ Selbstüberwachung anhand von Knoten und ihren Verbindungen

/NF0320/ Selbstüberwachung durch die Metadaten der Knoten

/NF0330/ Selbstüberwachung unter Berücksichtigung von Hosts und Netzwerktopologie

6.1.2 Optional

/NF0400/ Flexibles GUI Layout

/NF0500/ Möglichkeit der Lokalisierung der GUI - durch Nutzen der Qt Linguist Schnittstelle eine Übersetzung vereinfachen

6.2 Qualitätsanforderungen

6.2.1 Pflicht

/NF0600/ Möglichkeit die Metadaten zu erweitern

/NF0700/ Modularität der GUI

/NF0800/ Einhalten der ROS und Python Code Konventionen zur verbesserten Wartbarkeit

/NF0900/ Ausführliche Dokumentationen in Englisch

/NF0910/ Dokumentation der API

/NF0920/ Dokumentation der GUI

/NF0930/ Dokumentation des Python-Codes mittels PyDoc oder Sphinx

6.2.2 Optional

/NF1000/ Tutorial zur Nutzung des Programmes in Englisch

/NF1100/ Weitgehende Abdeckung durch Tests

7 Produktdaten

7.1 Config

- Konfigurationsdaten werden generell an den Parameterserver von ROS gesendet um diese Zentral zur Verfügung zu stellen.
- Soll-Spezifizierungen werden vom Benutzer an den Parameterserver gereicht. Dort holt der Monitoring Knoten sich die Spezifizierungen ab.

7.2 Metadaten

- Es wird ein eigener Metadatentyp zum Übertragen der Metadaten entworfen. Dazu gibt es noch einen eigenen Typ zur Übertragung der aufbereiteten Metadaten an die GUI.
- Eingehende Metadaten werden vom Monitoring Knoten erfasst und gespeichert.
- Der Monitoring Knoten verfügt über einen kurzzeitigen Speicher, um die empfangenen Metadaten aufzubereiten und zu publizieren.

8 Systemmodell

8.1 Szenarien

Definition des Soll-Zustandes

- Der Benutzer möchte entscheiden, welche Daten überwacht werden sollen.
- Er definiert eine Datei, welche den Namen und Typ der zu speichernden Daten beschreibt.
- Er bestimmt einen Bereich, in dem die Werte liegen sollen.
- Die Metadaten werden über Publisher-Subscriber an einen Monitor-Knoten geschickt.

Darstellung von Fehlern

Fehlerfeststellung durch den Benutzer

- Einem Benutzer ist ein Fehler aufgefallen: Der Roboter verhält sich nicht mehr, wie es erwartet wird.
- Er öffnet das PSE Introspection Plugin in der rqt GUI.
- In einer Knotenübersicht wird der fehlerhafte Knoten farblich hervorgehoben und lässt sich leicht erkennen.
- Er öffnet die betroffenen Knoten in der Detailansicht und sieht, in welchen Punkten das Knotenverhalten von den Spezifikationen abweicht.

Fehlerfeststellung über die GUI

- Um den Status des ROS-Netzwerkes im Auge zu behalten, hat der Benutzer das PSE Introspection Plugin in der rqt GUI geöffnet.
- Er stellt fest, dass die Statusanzeige nicht mehr in Grün volle Funktionalität signalisiert.
- Mit einem Klick öffnet er das Fehlerlog, in dem er Einträge findet, welche Knoten von ihren Spezifikationen abweichend arbeiten und die Fehlermeldung ausgelöst haben.
- Nun hat der Anwender die Möglichkeit, die fehlerhaften Knoten aus einer Liste zu suchen und sie in einer Detailansicht zu öffnen, um sich weitere Details über die ausgefallenen Komponenten anzeigen zu lassen.

Knotendetail-Ansicht

Auswahl eines Knotens

- Der Benutzer möchte sich die Metadaten Informationen zu einen Knoten anzeigen.
- Er wählt sich den Knoten, der ihn interessiert, aus einer Liste.
- Dazu besteht die Möglichkeit, diesen in einer Liste zu finden, welche sich auch über ein Suchfeld filtern lässt.
- Wählt er einen Knoten aus, öffnet sich ein kleines Fenster mit den Knoteninformationen und reiht sich in die Übersicht der geöffneten Detailfenster ein.

Umschalten der Detailansicht am Beispiel der Topic Liste

- Das Detailfenster bietet für verschiedene Knotendaten Links, um die Ansicht zu erweitern.
- Der Benutzer bekommt ein Topic angezeigt, mit dem der Knoten in einem Publisher-/Subscriber-Verhältnis steht.
- Klickt der Benutzer auf eine Schaltfläche neben der Anzeige, wird die Anzeige der sonstigen Metadaten durch eine vollständige Auflistung der relevanten Topics ersetzt.
- Intuitiv positioniert findet der Anwender eine Schaltfläche, um zur vorigen Ansicht zurückzukehren.

8.2 Anwendungsfälle

Überwachung eines Knoten

- Hauptakteur: Monitor
- Level: User Goal
- Scope: Monitor System

Beteiligte und Interessen

- Benutzer: Definition eines Soll-Zustandes für einen Knoten. Schnelles Treffen von Gegenmaßnahmen bei fehlerhaftem Knoten.
- Monitor: Herstellen einer Verbindung mit dem zu überwachenden Knoten. Empfangen des Ist-Zustandes des zu überwachenden Knotens. Abgleichen von Ist- und Soll-Zustand um Abweichungen zu finden. Information über den fehlerhaften Knoten an das rqt Widget senden.

Erfolgsgarantie

- Fehlerzustand wurde erkannt und angezeigt.
- Angemessene Gegenmaßnahmen konnten getroffen werden.

Haupterfolgsszenario

1. Benutzer definiert Soll-Zustand
2. Verbindung zwischen Knoten und Monitor wird hergestellt
3. Monitor empfängt Meta-Daten
4. Monitor gleicht den Soll-Zustand mit dem Ist-Zustand ab
5. Diskrepanz wird festgestellt
6. Fehlerhafter Knoten wird auf der GUI angezeigt
7. Benutzer kann Gegenmaßnahme treffen

Alternative Szenarien

1. Der Monitor empfängt keine Daten. Fehlende Verbindung wird auf der GUI angezeigt, damit der Benutzer Gegenmaßnahmen treffen kann.
2. Es wird keine Abweichung festgestellt.

Häufigkeit

Tritt häufig auf.

9 Benutzeroberfläche

Um die Daten des Monitoring Knotens sinnvoll darstellen zu können, soll eine graphische Oberfläche existieren, welche Diese übersichtlich darstellt. Hierbei gilt es, die bereits vorhandene rqt GUI um weitere Funktionalität zu ergänzen, sei es in Form von neuen Widgets oder durch Modifikationen bereits Existierender.

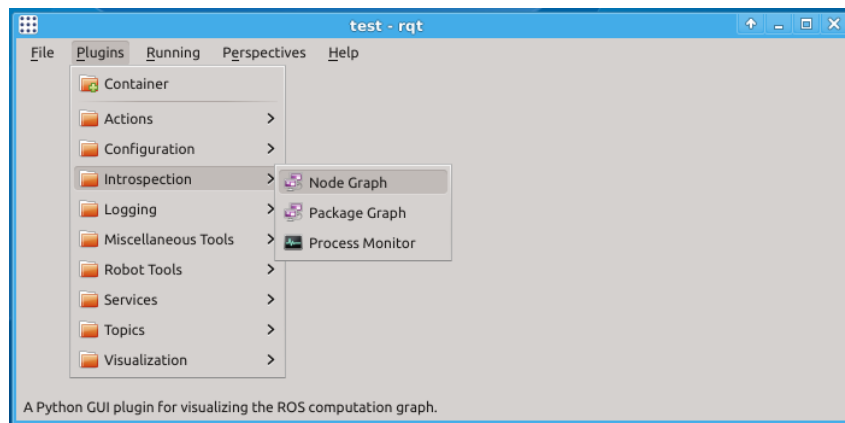


Abbildung 9.1: Die rqt GUI - Öffnen eines Widgets

Im Bild zu sehen ist das Öffnen eines Widgets in rqt. Durch ihre sehr modulare Bauweise können beliebig viele Widgets gestartet und gemeinsam, auf einer Oberfläche oder auch über mehrere Bildschirme verteilt, angezeigt werden. Alle nachfolgenden graphischen Elemente können mit Hilfe der hier gezeigten Funktionalität gestartet und auch gestoppt werden. Widgets sind im Allgemeinen der Grundbaustein einer graphischen Oberfläche mit Qt und können sowohl alleine als auch zusammen mit anderen Widgets in einem Fenster angezeigt werden.

9.1 Merkmale der graphischen Benutzeroberfläche

- Implementiert als rqt¹ Widget, d.h. basierend auf PyQt² bzw. PySide³
- Modularer Aufbau der Benutzeroberfläche in Form von Widgets, welche in die bereits bestehende rqt-Oberfläche integriert werden können
- Erweiterung der Funktionalität einiger rqt Widgets, um zusätzliche Informationen anzeigen zu können

¹<http://wiki.ros.org/rqt>

²<http://www.riverbankcomputing.co.uk/software/pyqt>

³<http://qt-project.org/wiki/pyside>

- Verwendung geeigneter graphischer Bibliotheken, um eine gute Übersichtlichkeit der Daten zu gewährleisten. Erwähnenswert ist hier insbesondere `qt_dotgraph` für den Graphen der Hosts/Nodes und `Matplotlib`⁴ für die Darstellung des zeitlichen Verlaufs von beispielsweise des Netzwerk-Traffics oder der CPU-Auslastung
- Modellierung der GUI in Form des Model/View(/Controller) Konzepts

9.2 Prototypen/Skizzen

9.2.1 ÜbersichtsWidget

Den Kern der graphischen Oberfläche bildet das ÜbersichtsWidget, welches einen Großteil der relevanten Informationen auf einer kleinen und übersichtlichen Oberfläche zusammenfasst. Hierunter fallen unter anderem Information zum aktuellen Status aller Knoten, Fehlermeldungen und Grafiken zum zeitlichen Verlauf der Auslastung unterschiedlicher Komponenten.

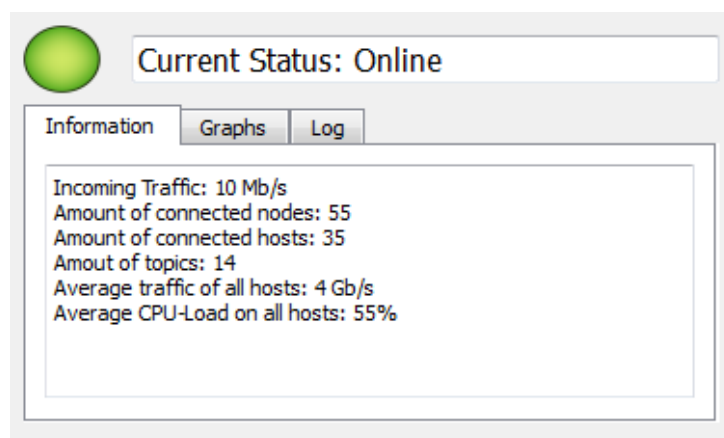


Abbildung 9.2: ÜbersichtsWidget - Reiter Information

Es werden hier allgemeine Informationen, d.h. der Durchschnitt der Auslastung aller Knoten angezeigt und durch die grüne "Ampel" eine korrekte Funktion aller (relevanten) Knoten signalisiert.

⁴<http://matplotlib.org/>

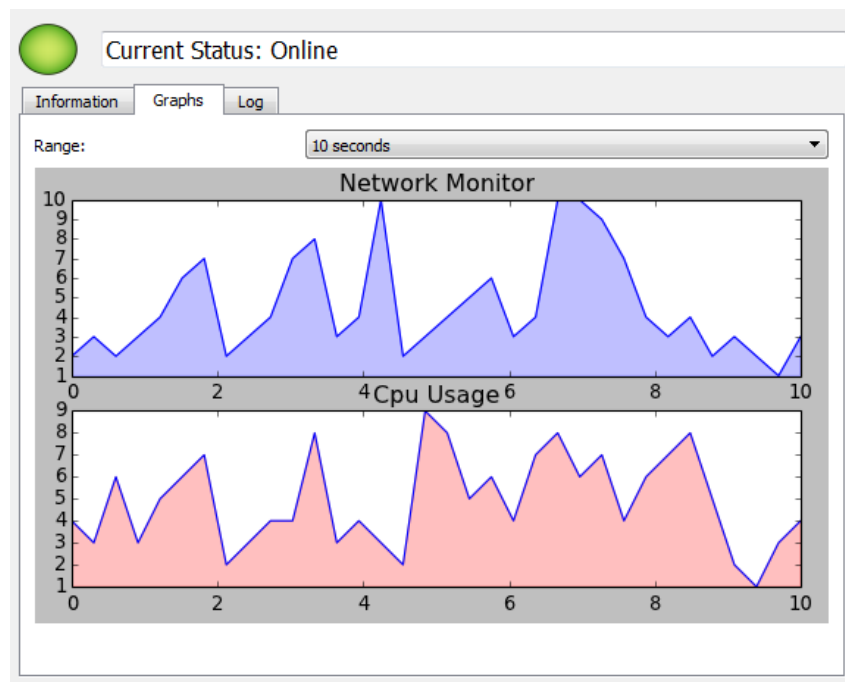


Abbildung 9.3: ÜbersichtsWidget - Reiter Graphs

Die Informationen des Informations-Reiters werden hier in Form von Graphen, d.h. im zeitlichen Verlauf dargestellt. Das betrachtete Intervall kann natürlich verändert werden.

The screenshot shows the 'Overview Widget - Log' tab. At the top, an orange circle indicates 'Current Status: Offline - Error'. Below this are three tabs: 'Information', 'Graphs', and 'Log' (selected). The main area contains a log table with the following data:

Typ	Date	Location	Message
Error	26.05.2014 - 12:43:12	/gazebo	Not responding
Warning	26.05.2014 - 12:40:35	/robots/lwr1/sa...	Failed to send message on /robots/lwr1/set_joints
Error	26.05.2014 - 12:40:28	/gazebo	Not responding
Update	26.05.2014 - 12:41:13	/gazebo	Responds again, normal functionality
Information	26.05.2014 - 12:38:12	/overview_widget	Startet this widget

Abbildung 9.4: ÜbersichtsWidget - Reiter Log

Im Fehlerfall ändert sich die Farbe der "Ampel", sodass der Nutzer unmissverständlich erkennt, dass ein Problem existiert, welches für eine korrekte Funktionsweise behoben werden muss. Im Log-Tab findet der Benutzer dann den Log der letzten Fehler, um so schnell und einfach herauszufinden, wo das Problem liegt.

9.2.2 Auswahl Widget

Dieses Widget zeigt anhand des gerade ausgewählten Elements (z.B. durch Anklicken im Node-Graph oder durch Auswählen im Listen Widget) detaillierte Information zu dem Element an. Element bedeutet in diesem Fall entweder Knoten oder Host. Ähnlich wie das Übersichts-Widget, ist auch dieses Widget in Tabs organisiert, wobei die Tabs Graphs und Log ähnlich funktionieren, wie beim Übersichtswidget, weshalb deren Funktionalität hier nicht erneut betrachtet werden soll.

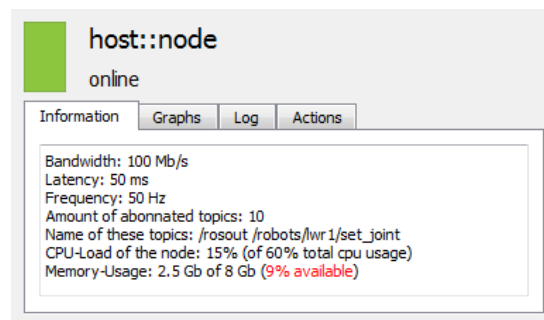


Abbildung 9.5: AuswahlWidget - Reiter Information

Der Reiter Information bietet, wie der Name schon sagt, sehr umfangreiche Information über das spezifische, gerade ausgewählte Element. Diese können je nach Element (Knoten/Host) unterschiedlich sein, umfassen jedoch u.A. Bandbreite, Latenz, Name der Topics auf denen gesendet wird, CPU-Auslastung, Speicherauslastung und Andere.

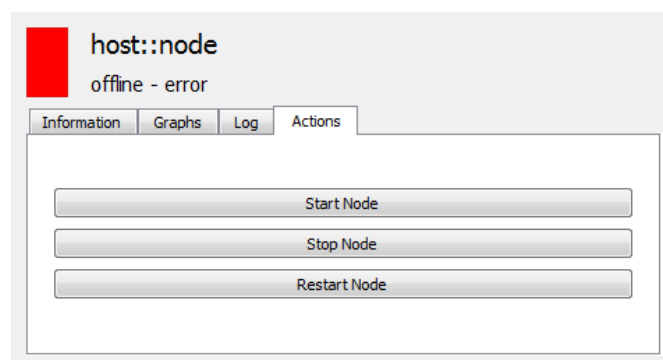
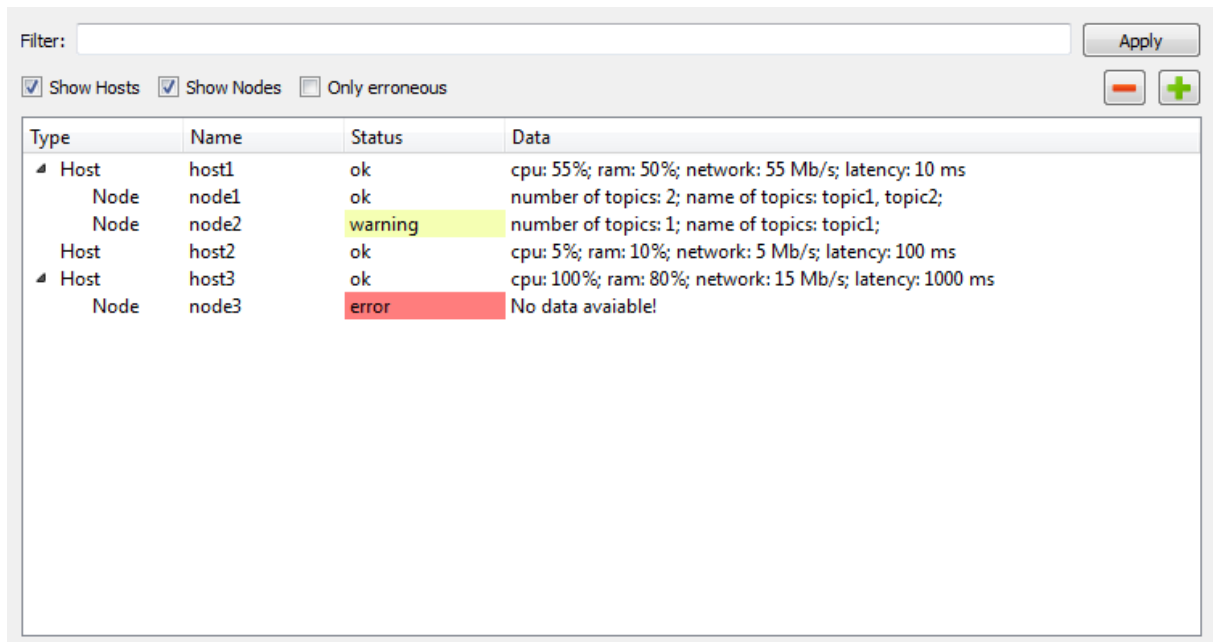


Abbildung 9.6: AuswahlWidget - Reiter Actions

Als optionales Feature können die Knoten im Problemfall neugestartet werden bzw. überhaupt gestartet werden. Dies erleichtert die Problembehebung erheblich, da diese Funktion aus der Ferne ausgeführt werden kann.

9.2.3 Listen Widget

Ein sehr einfaches Widget, welches dazu gedacht ist, die vorhandenen Hosts und Knoten übersichtlich anzuzeigen.

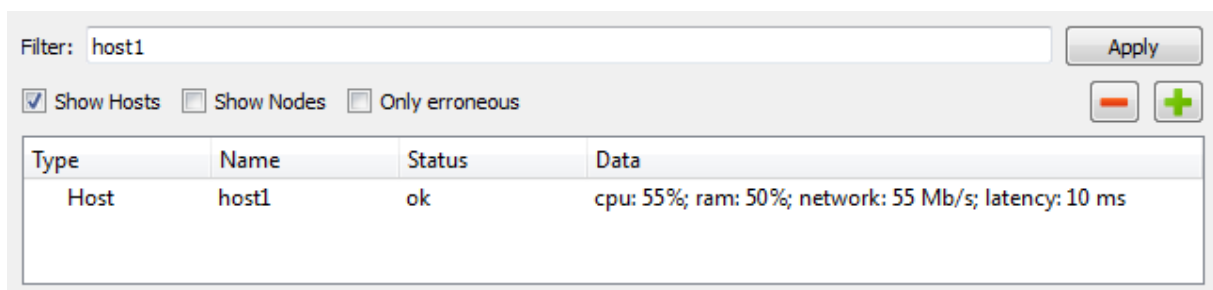


The screenshot shows the Listen Widget interface. At the top, there is a 'Filter:' input field and an 'Apply' button. Below this are three checkboxes: 'Show Hosts' (checked), 'Show Nodes' (checked), and 'Only erroneous' (unchecked). To the right of these checkboxes are two small buttons: a red minus sign and a green plus sign. The main area contains a table with the following data:

Type	Name	Status	Data
Host	host1	ok	cpu: 55%; ram: 50%; network: 55 Mb/s; latency: 10 ms
Node	node1	ok	number of topics: 2; name of topics: topic1, topic2;
Node	node2	warning	number of topics: 1; name of topics: topic1;
Host	host2	ok	cpu: 5%; ram: 10%; network: 5 Mb/s; latency: 100 ms
Host	host3	ok	cpu: 100%; ram: 80%; network: 15 Mb/s; latency: 1000 ms
Node	node3	error	No data available!

Abbildung 9.7: Listen Widget - Hauptansicht

Um die Übersichtlichkeit zu erreichen werden die vorhandenen Daten zunächst hierarchisch angezeigt, was sich durch De-/Aktivieren der Punkte "Show Nodes" und "Show Hosts" modifizieren lässt. Des Weiteren lassen sich die Elemente nach Suchbegriffen filtern und ermöglichen so ein schnelles Finden des gewünschten Elements. Ein weiterer Filter ermöglicht das Anzeigen nur solcher Elemente, die derzeit in einem Fehler-Zustand sind. Eine Kombination der Filter ist natürlich möglich.



The screenshot shows the Listen Widget interface with the filter 'host1' applied. The 'Filter:' input field contains 'host1'. The checkboxes are 'Show Hosts' (checked), 'Show Nodes' (unchecked), and 'Only erroneous' (unchecked). The table now displays only the data for 'host1':

Type	Name	Status	Data
Host	host1	ok	cpu: 55%; ram: 50%; network: 55 Mb/s; latency: 10 ms

Abbildung 9.8: Listen Widget - Beispielhafte Anwendung der Filter

9.2.4 Node Graph

Der Node Graph ist ein bereits vorhandenes Widget, das bereits in rqt integriert ist und Informationen zu den Knoten in einem sehr übersichtlichen Graphen zur Verfügung stellt.

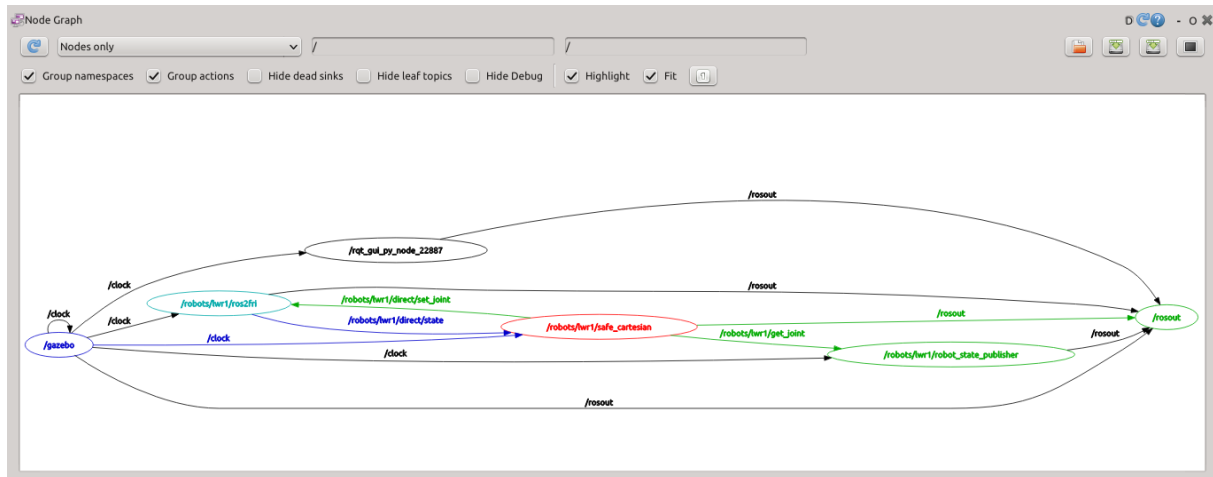


Abbildung 9.9: Node Graph Widget

Erweiterung des Node Graphs

Die Erweiterung des Node Graphs bietet die Funktionalität sich Metadaten von Topics direkt anzeigen zu lassen. Des Weiteren werden die Kanten je nach Soll-Status eingefärbt.

So ist beispielsweise das Topic `/voxelgrid` als rot markiert, da die Latenz außerhalb der Soll-Definition liegt. Das Topic `/camera/depth/points` ist dagegen orange, da die Frequenz vor kurzem außerhalb des Soll-Wertes war, mittlerweile aber wieder im Soll-Bereich ist. Dagegen ist `/camera/rgb/image_rect_color` grün, da alle Werte im vorgegebenen Bereich liegen.

Des weiteren werden die Kanten unterschiedlich breit dargestellt, abhängig von der jeweiligen Datenrate.

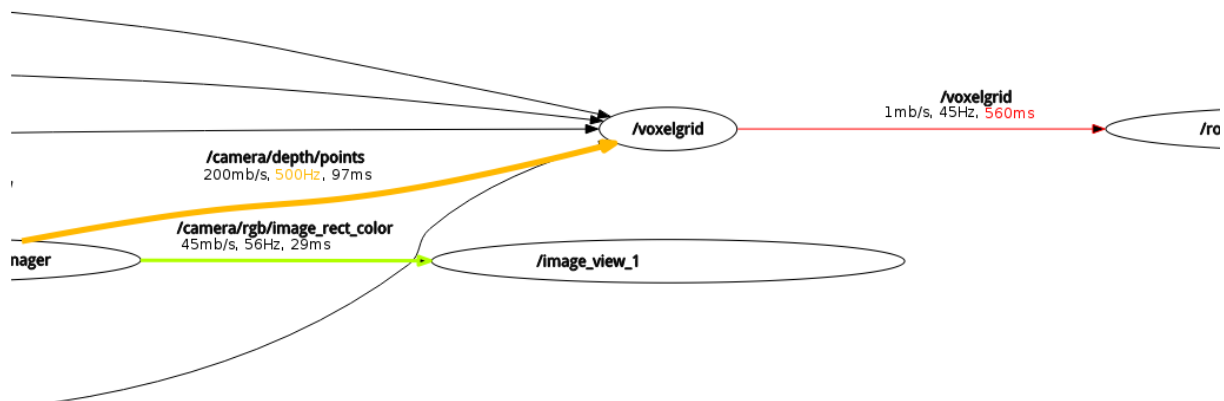


Abbildung 9.10: Erweiterter Node Graph⁵

⁵Dieses Bild ist eine Modifikation von ROS Wiki:rqt_graph, lizenziert unter CC 3.0

10 Qualitätsanforderungen

10.1 Erweiterbarkeit

Um in einer erweiternden Entwicklung den Umfang der übertragenen Metadaten zu erweitern, muss zum einen der Topic-Datentyp erweitert werden, mit dem die Daten zum Monitoringknoten übermittelt werden und zum anderen muss die Funktion zur Erfassung der Daten ergänzt werden.

10.2 Wartbarkeit

Aufgrund einer transparenten Abfolge in der Übertragung der Metadaten, können die einzelnen Stationen stückweise auf Fehler untersucht werden.

Da das Projekt mit dem ROS Publisher/Subscriber Prinzip die Metadaten auf Topics überträgt, ist es zudem ohne weiteres möglich, den Inhalt der übertragenen Daten mit bestehenden Mitteln einzusehen und zu kontrollieren.

10.3 Zuverlässigkeit

Die Funktionalität des ROS Netzwerkes wird durch unerheblichen Zusatzaufwand zur Verarbeitung der erhobenen Metadaten nicht weiter eingeschränkt.

11 Testfälle und Testszenarien

11.1 Tests

/TF0100/ Der Monitoring-Knoten erkennt abweichendes Verhalten und gibt diese Information an den Countermeasure-Knoten weiter.

/TF0200/ Eine Komponente sendet mit festgelegter Frequenz Metadaten. Sie kommen mit der selben Frequenz vollständig am Monitoring-Knoten an.

/TF0300/ Ein Metadatenpaket überschreitet auffällig das Mittel, nachfolgende Pakete halten sich aber im Rahmen. Das fehlerhafte Paket wird aufgezeichnet aber ignoriert.

/TF0400/ Eingelesene Konfigurationsdaten lassen sich parsen und auslesen.

/TF0500/ Werden keine eigenen Konfigurationsdaten angegeben, werden sinnvolle Standardwerte verwendet.

/TF0600/ Die Hostdaten werden korrekt übermittelt.

/TF0700/ Optional: Ein Knoten erhält Daten von mehreren anderen Knoten. Er informiert, wenn er weniger Quellen hat, als üblich.

/TF0800/ Optional: Funktionsweise des Startens und Stoppens von Knoten unter Netzwerklast prüfen.

11.2 Beispielszenarien

Typische Vorgehensweise

- Der Benutzer startet zunächst roscore, dann neben den von ihm benötigten Knoten den Monitoring Knoten.
- Um den aktuellen Status seiner Knoten und des Netzwerks zu überwachen, startet er nun rqt und öffnet das Listen Widget. Falls nun Fehler auftreten, wird er durch eine farbliche Veränderung auf mögliche Probleme hingewiesen.

Knoten existiert nicht mehr

- Ein ROS-Netzwerk mit beliebig vielen Knoten wird gestartet.
- Der Benutzer öffnet das Plugin in der rqt GUI.
- Nun wird ein Knoten vom Netzwerk getrennt (z.B. durch Programmabsturz).
- Falls dieser Knoten in den Soll-Spezifikationen als wichtig definiert wurde, reagiert die GUI nun darauf und zeigt an, dass der Knoten nicht mehr verfügbar ist und dass ein Fehler aufgetreten ist.
- Optional bietet die GUI nun die Möglichkeit den betroffenen Knoten neuzustarten.

Knoten weicht von seinen Sollwerten ab

- Wieder wird ein ROS-Netzwerk mit beliebig vielen Knoten gestartet.
- Es werden die Soll Spezifikationen festgelegt und der Monitoring Knoten gestartet.
- Der Zustand eines Knotens verschlechtert sich gravierend und er überschreitet die Soll-Spezifikation an mindestens einem Punkt.
- Der Monitoring Knoten erkennt das Überschreiten der Spezifikation und startet ggf. vordefinierte Aktionen.
- Der Benutzer öffnet den Node Graph in der GUI und sieht den betreffenden Knoten aufgrund der farblichen Hervorhebung sofort. Optional kann er sich auch in Form eines Graphen den Verlauf der Daten über bestimmte Zeiträume hinweg anzeigen lassen.
- Der Benutzer hat nun entweder die Möglichkeit den Knoten aus der GUI heraus neuzustarten oder er nimmt manuell Reparaturen vor.

Countermeasure Knoten

- Vorrausgesetzt wird das vorherige Szenario, in dem ein Knoten die Soll-Spezifikation überschreitet.
- Nun reagiert der Countermeasure Knoten auf diese Abweichung und leitet vordefinierte Gegenmaßnahmen ein.
- Z.B. wird der Name des betroffenen Knotens in der Shell ausgegeben.

Dynamische GUI

- Der Benutzer schließt ein rqt Plugin, das er nicht mehr benötigt.
- Daraufhin verschiebt sich das Layout in der rqt GUI und das PSE Introspection Plugin ändert seine Form und Größe.
- Das PSE Introspection Plugin rearrangiert seine Inhalte, um sie im neuen Layout anzuzeigen.

12 Glossar

12.1 Allgemein

API Application Programming Interface. Eine Schnittstelle, über die Programme auf Funktionalität anderer Programme zugreifen können

CPU Central Processing Unit. Das englische Akronym für den Prozessor eines Computers

GUI Graphische Benutzeroberfläche, original aus dem Englischen: Graphical User Interface

Middleware Schicht, die zwischen Anwendungen und der Infrastruktur vermittelt um so die Komplexität der anderen Schichten zu verbergen

Open Source Software die Quelloffen ist und weiterentwickelt sowie weiterverbreitet werden darf

12.2 ROS

Knoten Siehe Node

Master Der Master ermöglicht die Namensauflösung/-registrierung und ist zuständig für die Verknüpfung von Knoten miteinander

Message Nachricht, welche zwischen Knoten versendet wird. Eine Nachricht besteht aus primitiven Datentypen

Node Auch Knoten genannt. Repräsentiert ein Prozess. Ermöglicht verschiedene Aufgaben zu trennen und so das ROS Netzwerk modular zu gestalten.

Parameter Server Der Parameter Server erlaubt das zentrale Speichern von Daten. Alle Knoten können auf ihn zugreifen

Publisher Objekt, das Messages unter bestimmten Topics veröffentlicht

ROS Das Robot Operating System ist eine Open-Source Middleware um flexible Robotik Software zu schreiben

Service Services dienen zur zweiwege Kommunikation in ROS Netzwerken, da das Publisher/Subscriber System nur einwege Kommunikation unterstützt. Services verhalten sich wie entfernte Funktionsaufrufe (RPC)

Subscriber Objekt, das Messages von abonnierten Topics empfängt

Topic Ein Topic ist ein Name um den Inhalt einer Message zu definieren. Publisher können Messages auf bestimmten Topics publizieren, Subscriber können Messages von abonnierten Topics erhalten