Karlsruher Institut für Technologie
Andreas Bihlmaier
andreas.bihlmaier@gmx.de

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

Into the ROS
Advanced ROS Network Introspection

Praxis der Softwareentwicklung SS 2014

Lastenheft

Revision: 1.0

Inhaltsverzeichnis

I Projektbeschreibung

Im Rahmen des Projekts soll eine (Selbst)überwachung für die Robot Operating System (ROS) Middleware erstellt werden. ROS gliedert die Algorithmen zur Realisierung intelligenter Roboter in einzelne Aufgaben, welche durch jeweils einen Prozess („Node“/„Knoten“) dargestellt werden und auf mehrere Rechner („Hosts“) verteilt sein können. Die Nodes verbinden sich nach dem Publisher-Subscriber Schema über ein Netzwerk dynamisch miteinander um die notwendigen Datenflüsse herzustellen („ROS Graph“). Dabei dient eine zentrale Instanz („roscore“) als Namensdienst, welcher die Subscriber über bekannte Namen („Topics“) mit den Publishern zusammenbringt. Die eigentliche Datenübertragung erfolgt direkt zwischen den Knoten nach dem Peer-to-Peer Prinzip.

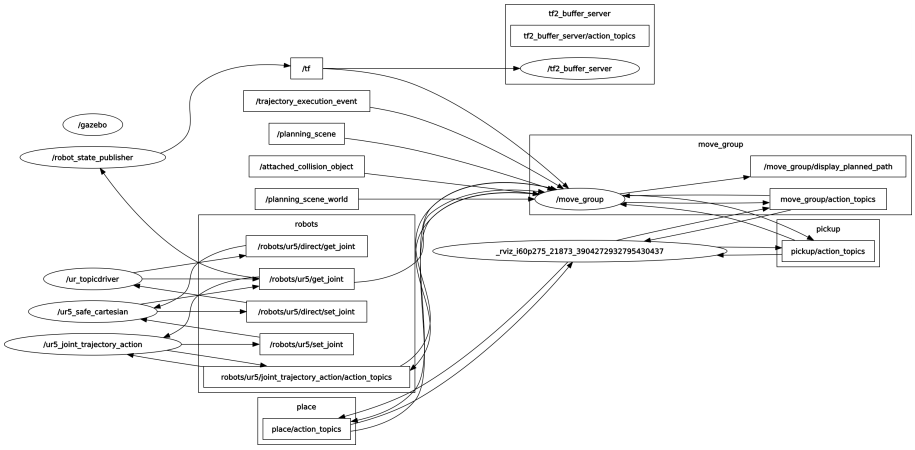

Abbildung 1: Exemplarischer ROS Graph. Nodes sind als Ellipsen dargestellt, Topics durch Rechtecke und der Datenfluss zwischen den Nodes (über die jeweiligen Topics) durch Pfeile.

Als Grundlage des Projekts dienen bestehende ROS Fähigkeiten zur Introspection, wie die Möglichkeit den gesamten aktuellen ROS Graph abzufragen (vgl. Abbildung 1). Außerdem Erkenntnisse und Software aus dem Bereich der Überwachung von komplexer IT Infrastruktur (z.B.

Nagios[1][2]).

I.1 Ausgangsproblemstellung und erwartete Verbesserung

Die hohe Flexibilität einer netzwerktransparenten Middleware ist speziell in der Robotikforschung eine gewünschte Eigenschaft. Jedoch birgt die gesteigerte Komplexität eines verteilten Systems ihre Tücken im Hinblick auf Erfassung und Verstehbarkeit des Ist-Zustands im laufenden Gesamtsystem, Robustheit im Sinne der Betriebssicherheit und die Auffindbarkeit sowie Behebung von Fehlern.

Durch das vorliegende Projekt soll sich in den Punkten Erfassbarkeit, Verstehbarkeit und Zuverlässigkeit eine Verbesserung ergeben. Zunächst muss es möglich sein alle relevanten Parameter des Ist-Zustandes ohne Beeinträchtigung des laufenden Systems kontinuierlich zu erfassen. Im nächsten Schritt müssen diese Rohdaten abstrahiert und auf einem höheren semantischen Niveau zusammengefasst werden. Die aggegierten Informationen sind auf geeignete Weise zu visualisieren, d.h. auch ein komplexer Ist-Zustand, im Besonderen ein darin enthaltener Fehler, muss vom Benutzer schnell verstanden werden können. Zuletzt ist das Ziel dem System eine weitgehende Selbstüberwachung dadurch zu ermöglichen, dass der Soll-Zustand für eine gegebene Situation definiert wird und vom System selbst gegen den Ist-Zustand geprüft wird. Bei Abweichungen kann das System entweder Gegenmaßnahmen treffen, z.B. einen Knoten neustarten, oder zumindest das System in einen ungefährlichen Zustand überführen und den Benutzer durch einen sinnvollen Fehlerbericht warnen.

I.2 ROS

Das Robot Operating System (ROS) ist ein flexibles Framework zur Programmierung von Robotiksoftware. Es definiert einerseits eine netzwerktransparente Middleware mit nützlichen Kommandozeilen[3] und GUI

[1] Wikipedia: *Nagios — Wikipedia, The Free Encyclopedia.*
[2] *Nagios.*
[3] `http://wiki.ros.org/ROS/CommandLineTools`

Tools. Andererseits bringt ROS eine Vielzahl von implementierten Robotikalgorithmen, welche durch Verwendung der Middleware und ihrer Nachrichtentypen („message types“) zueinander kompatibel sind. Dies beinhaltet sowohl grundlegende Funktionalität in der Robotik, wie die Umrechnung zwischen verschiedenen Koordinatensystemen[4], als auch sehr komplexe Fähigkeiten, beispielsweise simultane Lokalisierung und Kartenerstellung (SLAM)[5]. Das heute wichtigste Merkmal von ROS ist jedoch seine große Nutzer- und Entwicklercommunity. ROS ist sowohl im akademischen wie auch im industriellen[6] Forschungsumfeld im Einsatz und erfährt von dort kontinuierlich Verbesserungen und Erweiterungen. [7][8][9]

I.3 Qt und PyQt/PySide

Qt ist eine C++-Klassenbibliothek für die plattformübergreifende Programmierung grafischer Benutzeroberflächen. Neben der Entwicklung grafischer Benutzeroberflächen bietet Qt umfangreiche Funktionen zur Internationalisierung, Netzwerk- und Interprozesskommunikation, Datenbankfunktionen sowie XML-Unterstützung an und ist für verschiedene Betriebssysteme bzw. Grafikplattformen erhältlich. Qt verwendet einen Präprozessor, genannt MOC (meta object compiler), womit C++ um Fähigkeiten bereichert wird, die im Sprachstandard nicht enthalten sind, beispielsweise Signale und Slots sowie Introspektion. Der so erzeugte Code folgt dem C++-Standard, so dass er mit handelsüblichen Compilern übersetzt werden kann. Neuere Qt Versionen stehen unter der LGPL.[10][11]

Qt kann über language bindings auch direkt in Python Programmen verwendet werden. Es existierten derzeit zwei verschiedene – weitge-

[4] `http://wiki.ros.org/tf2`
[5] `http://wiki.ros.org/navigation`
[6] rosindustrial.org
[7] `http://www.ros.org/about-ros/`
[8] *The Robot Operating System (ROS).*
[9] `http://wiki.ros.org/`
[10] Wikipedia: *Qt (Bibliothek) — Wikipedia, Die freie Enzyklopädie.*
[11] *Qt Project.*

hend kompatible – bindings: PyQt[12] und PySide[13]. Im Rahmen von ROS gibt es eine Abstraktionsschicht[14], welche transparent eines der beiden bindings verwendet und für das vorliegende Projekt genutzt werden soll.

I.4 rqt

rqt ist ein auf Qt basierendes Framework zur GUI Entwicklung für ROS[15] mit Python Unterstützung[16]. Es stehen bereits zahlreiche rqt Plugins zur (zweidimensionalen [17] visuellen) Wiedergabe des aktuellen ROS Systemzustandes zur Verfügung. Besonders zu nennen ist rqt_graph, welches auch für Abbildung 1 verwendet wurde.

[12] *PyQt - What is PyQt?*
[13] *PySide - Qt Project*.
[14] `http://wiki.ros.org/python_qt_binding`
[15] `http://wiki.ros.org/rqt`
[16] `http://wiki.ros.org/rqt/Tutorials/WritingaPythonPlugin`
[17] Eine vielfältige 3D Visualisierung bietet rviz

II Systemübersicht

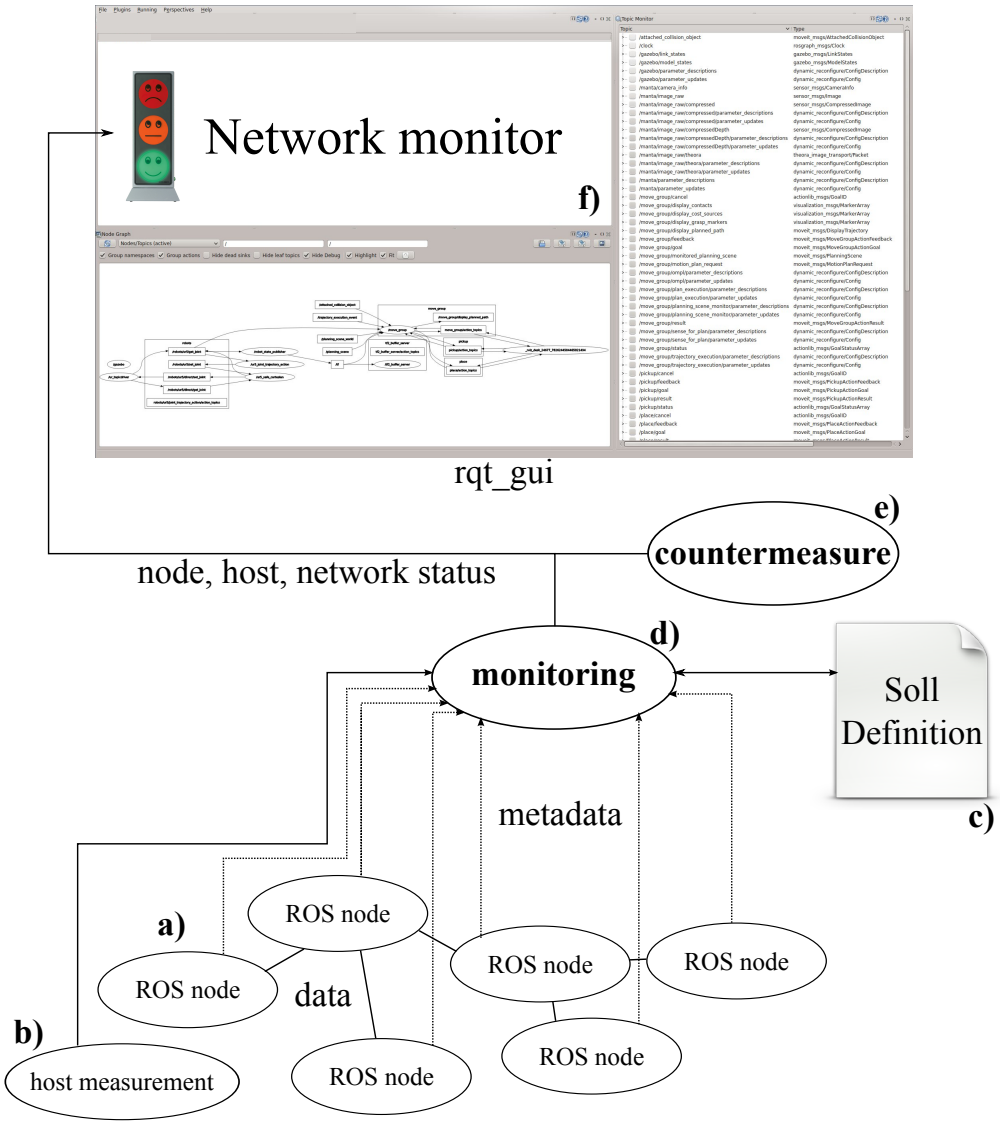

Abbildung 2: Übersicht des zu entwickelnden Systems bestehend aus a) Erweiterung beliebiger bestehender ROS Knoten um Metadatenerzeugung, b) Knoten für Hostüberwachung, c) Definition des Soll-Zustandes, d) Knoten für zentralen Soll-Ist-Vergleich, e) Knoten um Gegenmaßnahmen bei Abweichung zu ergreifen und f) rqt GUI Plugin zur Anzeige des Gesamtstatus.

III Zu entwickelnde Softwarelösung

Die zu entwickelnden ROS Bibliotheken und Knoten soll den Standards für objektorientierte Programmierung (OOP) entsprechen und in Python entwickelt werden. Es sind so weit wie möglich bestehende Funktionalitäten und Bibliotheken anstatt von Eigenentwicklungen zu nutzen. Aufbau und Codekonventionen sollen sich nach denjenigen in ROS und Python, in dieser Prioritätsordnung, richten. Die Zielplattform ist Linux, jedoch soll soweit möglich die plattformunabhängigkeit der zugrundeliegenden Bibliotheken bewahrt werden, d.h. plattformunabhängige Lösungen sind plattformspezifischen vorzuziehen. Das Integrationsziel der entwickelten Software ist ein ROS Stack der Standarddistribution zu werden. Diese Zielvorgabe soll als Leitfaden dienen, muss jedoch nicht tatsächlich erreicht werden. In jedem Fall empfiehlt sich während der Entwicklung informellen Kontakt mit der ROS Entwicklercommunity aufzunehmen[18] um die Randbedingungen und Schwerpunkt der initialen Entwicklung richtig zu setzen, Akzeptanz sowie Nutzen der Community zu steigern und um künftigen Erweiterungen den Weg zu bereiten.

IV Produkteinsatz

Die Software soll zunächst im universitären Forschungsumfeld des beauftragenden Institutes eingesetzt werden. Später kann der Nutzerkreis potentiell auf alle ROS Benutzer ausgedehnt werden.

V Produktfunktionen

V.1 Bewertung der gewünschten Produktfunktionen

! Musskriterien

\+ Wunschkriterien

[18] `http://answers.ros.org/questions/`

V.2 Gewünschte Produktfunktionen

- Gesamtsystem
  - ! Dezentrale Erfassung von
    - ∗ Lebendigkeit (watchdog[19] Funktionalität)
    - ∗ Topic Statistiken (Metadaten): Anzahl Publisher und Subscriber, Bandbreite, Frequenz, Latenz, Jitter

    ohne zusätzliche Subscriber
  - ! Definition eines geeigneten ROS Message Types für Metadaten
  - ! Modulare Definition des Soll-Zustand basierend auf
    - ∗ ROS Graph
    - ∗ Metadaten
  - ! Knoten für zentralen Abgleich des Soll- und Ist-Zustand
    - ! Warnungen- und Fehlernachrichten (rosconsole)
    - + Durchführung definierter Gegenmaßnahmen
  - + Eigenständiger Knoten zur Host Überwachung (Publizieren von CPU Auslastung, Systemtemperaturen, etc)
  - + Zentrale Berechnung und Publizierung von Korrelationen zwischen Topics (d.h. Zusammenhang zwischen Empfangs- und Sendeverhalten eines Knotens), deren Definition und Überwachung
  - + Überwachung weiterer ROS Bestandteile[20]: Services, Parameters
  - + Umwandlung des Ist-Zustandes in Soll-Definition
  - + Definition der Netzwerktopologie und Aggregierung der Metadaten auf Ebene der physischen Verbindungen (z.B. Auslastung eines Gigabit-Ethernet Links)

[19] `http://wiki.ros.org/watchdog_timer`
[20] `http://wiki.ros.org/ROS/Concepts`

    + Integration mit roswtf

- API zur Erweiterung von Nodes
    ! Erfassung der Metadaten durch Hinzufügen *eines* Funktionsaufrufes zu bestehenden Callbacks
    ! Publizierung der Metadaten auf einem Topic mit definierter Frequenz
    ! (De)aktivierbarkeit der Metadatenerfassung über (private) Parameter
    + Selbstüberwachung eines Knoten anhand Soll-Metadaten
    + auch für C++
- GUI zur zentralen Visualisierung
    ! (Ampel) Visualisierung des Soll-Ist Vergleichs (vgl. Nagios bzw. rqt_robot_monitor) auf der Ebene von
        * Knoten
        * Hosts
    + Ergänzung der ROS Graph Visualisierung durch
        * Metadaten (vgl. rqt_graph)
        * Soll-Ist Vergleich
    + Sinnvolle Visualisierungen des ROS Netzwerkzustandes bzw. der Metadaten, z.B. im zeitlichen Verlauf mit rqt_plot
    + 2D räumliche Zuordnung von logischen Entitäten (Nodes, Hosts) zu Hardware

VI Produktleistungen

! Effiziente Erfassung der Metadaten

! Vernachlässigbarer Leistungsbedarf bei deaktivierter Metadatenerfassung

! Stufenweise Introspektion und (Selbst)Überwachung

  - Nur Knoten und ihre Verbindungen
  - mit Metadaten
  - unter Berücksichtigung von Hosts und Netzwerktopologie

+ Flexibles GUI Layout im Bezug auf Bildschirmfläche

VII Qualitätsanforderungen

! Erweiterbarkeit um zusätzliche Metadaten

! Modularer Aufbau der GUI

! Einhaltung der ROS und Python Konventionen

! Ausführliche Dokumentation der API und des GUI

! Dokumentation des internen Codes mit Python docstrings

+ Erstellung eines Tutorial

+ Testabdeckung

VIII Systemumgebung

- Ubuntu LTS (12.04 oder 14.04)
- ROS Hydro oder neuer
- Python 2.7 oder neuer
- Qt 5.0 oder neuer

Literatur

Blanchette, Jasmin und Mark Summerfield: *C++ GUI Programming with Qt4*, ^{2}2008.

Nagios, [Online; Stand 28. März 2014], 2014, URL: http://www.nagios.org/.

PyQt - What is PyQt?, [Online; Stand 31. März 2014], 2014, URL: http://www.riverbankcomputing.co.uk/software/pyqt/intro.

PySide - Qt Project, [Online; Stand 31. März 2014], 2014, URL: http://qt-project.org/wiki/PySide.

Qt Project, [Online; Stand 4. November 2013], 2013, URL: http://qt-project.org.

ROS-Industrial, [Online; Stand 31. März 2014], 2014, URL: http://www.rosindustrial.org/.

Tanenbaum, Andrew S. und Maarten van Steen: *Verteilte Systeme*, München ^{2}2007.

The Robot Operating System (ROS), [Online; Stand 28. März 2014], 2014, URL: http://www.ros.org/.

Wikipedia: *Nagios — Wikipedia, The Free Encyclopedia*, 2014, URL: http://en.wikipedia.org/w/index.php?title=Nagios&oldid=601461121.

Wikipedia: *Qt (Bibliothek) — Wikipedia, Die freie Enzyklopädie*, [Online; Stand 4. November 2013], 2013, URL: http://de.wikipedia.org/w/index.php?title=Qt_(Bibliothek)&oldid=123854287.