

INTO THE ROS

ADVANCED ROS NETWORK INTROSPECTION

Praxis der Softwareentwicklung
Sommersemester 2014

Q u a l i t ä t s s i c h e r u n g



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, 22.09.2014

Inhaltsverzeichnis

1	Einleitung	3
2	Unittests	4
2.1	Arni_Core	4
2.2	Arni_Processing	6
2.3	Arni_Countermeasure	8
2.4	Arni_Nodeinterface	13
3	Integrationstests	14
3.1	Test 1 - Gleichmäßiges Publizieren	14
3.2	Test 2 - Zu niedrige Frequenz	16
3.3	Test 3 - Variierende Frequenz	18
3.4	Test 4 - Neustarten eines Knotens mit behobenem Fehler	21
3.5	Test 5 - Neustarten eines Knotens mit bestehendem Fehler	24
4	Code-Abdeckung	26
4.1	Nodeinterface	26
4.2	Processing	27
4.3	Countermeasure	28
4.4	GUI	29

1 Einleitung

In diesem Dokument wird die Qualitätssicherung des PSE-Projekts Advanced ROS Network Introspection beschrieben. Durch umfangreiches Testen wird sichergestellt, dass der in der Implementierungsphase geschriebene Code korrekt und stabil ist. Dabei wurde in folgenden Schritten vorgegangen:

- Die im Pflichtenheft aufgestellten Tests und Szenarien wurden verfeinert und hier getestet.
- Für jede Komponente wurden jeweils einzelne Unittests geschrieben, um die isolierte Funktionsweise zu testen.
- Um das Zusammenspiel der einzelnen Komponenten zu testen, wurden Integrationstests entwickelt.
- Es wurden Statistiken zur Code Coverage erhoben. Allerdings sind diese mit Vorsicht zu betrachten, da sie sich nicht durch Unittests erreichen lassen konnten. Somit spiegeln sie nicht die komplette Testabdeckung wieder.

2 Unittests

2.1 Arni_Core

TestSeuid

`test__valid__noarg`

Kein Argument ist keine gültige SEUID.

Bestanden

`test__valid__override`

Die Methode `is_valid()` funktioniert statisch auf einem gültigen SEUID Objekt.

Bestanden

`test__valid__from__class`

Erstellen eines SEUID Objektes mit ungültiger SEUID wirft einen `NameError`.

Die Methode `is_valid()` gibt auf einem gültig initialisiertem SEUID Objekt den richtigen Wert zurück.

Bestanden

`test__valid__from__fn`

Die Methode `is_valid()` funktioniert statisch auf einem SEUID Objekt ohne eigene SEUID.

Bestanden

`test__valid__types`

Überprüft verschiedene gültige und ungültige SEUIDs.

Bestanden

`test__valid__from__serialization`

Testet die Konstruktion mit einem serialisierten SEUID Objekt.

Bestanden

test_fields_node

Testet, ob die öffentlichen Attribute des Objektes mit einer Node-SEUID richtig gesetzt werden.

Bestanden

test_fields_override

Überprüft, ob die öffentlichen Attribute des Objektes zurückgesetzt werden, wenn es mit einem anderen Typ SEUID initialisiert wird.

Bestanden

test_fields_connection

Überprüft die öffentlichen Attribute eines Connection-SEUID Objektes.

Bestanden

test_msg_invalid

Die Konstruktion mit einem Objekt, das kein gültiger Nachrichtentyp ist, wirft einen TypeError.

Bestanden

test_msg_hostmsg

Testet die Konstruktion anhand einer HostMessage.

Bestanden

test_get_seuid

Die Methode `get_seuid()` liefert für eine SEUID falls vorhanden die SEUID des enthaltenen Hosts/Nodes/Publishers/Subscribers.

Bestanden

test_get_seuid_invalid

Die Methode `get_seuid()` wirft einen `AttributeError`, wenn das Objekt nicht mit einer SEUID initialisiert wurde.

Die Methode `get_seuid()` wirft einen `KeyError`, wenn die angefragte SEUID nicht im Objekt vorhanden ist.

Bestanden

2.2 Arni_Processing

TestLoadingSpecifications

test_no_specifications

Der Specificationhandler ist am Anfang leer, wenn er keine Spezifikationen im Namespace findet.

Bestanden

test_load_spec

Testet, ob eine Spezifikation mit der angegebenen SEUID nach dem Laden vorhanden ist. Verwendet das format [seuid1:].

Bestanden

test_load_new_specs

Überprüft, dass auch das format **section:** [seuid1:] geladen wird.

Bestanden

test_reload_spec

Überprüft, dass Spezifikationen zu SEUIDs in den Bestand geladen wurden, nachdem sie auf den Parameterserver geladen und die reload-Methode aufgerufen wurde.

Bestanden

test_invalid_seuid

Überprüft, dass keine Spezifikationen geladen werden, die keine gültige SEUID haben.

Bestanden

test_existing_fields

Überprüft, alle Felder der Definition durch den Parameterserver gekommen sind und im Specification Objekt vorhanden sind.

Bestanden

TestRatingData

test_no_data

Wird versucht None zu bewerten, wird None als Vergleichsergebnis zurückgegeben.

Bestanden

test_invalid_ident

Ist der Identifier der Eingabedaten ungültig, wird None als Vergleichsergebnis zurückgegeben.

Bestanden

test_no_ident

Wird kein Identifier mitgegeben, wird None als Vergleichsergebnis zurückgegeben.

Bestanden

test_no_spec

Ist keine Spezifikation für die angegebene Nachricht geladen, haben die bewerteten Felder den Status 2 (Unknown).

Bestanden

test_spec3

Werden die Spezifikationen erfüllt, wird für die jeweiligen Felder der Status 3 (OK) zurückgegeben.

Bestanden

test_spec2

Sind einzelne Felder nicht spezifiziert, wird für diese der Status 2 (Unknown) zurückgegeben.

Bestanden

test_spec10

Felder, die Werte über ihren Limits liefern, werden mit dem Status 0 (High) bewertet.

Felder, die Werte unter ihren Limits liefern, werden mit dem Status 1 (Low) bewertet.

Bestanden

2.3 Arni_Countermeasure

Test parsing of constraints

test_empty_reaction

Simuliert eine leere Reaction in einem Constraint und erwartet das dies keine Exception verursacht.

Bestanden

test_interval_timeout_not_existing

Testet, ob die Standartwerte von min_reaction_interval und Reaction_timeout geladen werden, wenn ein Constraint diese nicht angibt.

Bestanden

test_interval_valid_values

Testet, ob das Setzen von min_reaction_interval und Reaction_timeout erfolgreich interpretiert wird.

Bestanden

test_interval_wrong_value

Testet, ob bei ungültiger Eingabe für min_reaction_interval auf den Standartwert zurückgegriffen wird.

Bestanden

test_missing_reactions

Simuliert fehlende Reactions in einem Constraint und erwartet, dass dies keine Exception verursacht.

Bestanden

test_reaction_autonomy_level_not_set

Testet, ob eine Reaction, welche kein Autonomy-Level gesetzt hat, erfolgreich interpretiert wird.

Bestanden

test_reaction_multiple_reactions

Testet, ob zwei Reactions in einem Constraint als zwei Reactions interpretiert werden.

Bestanden

test_reaction_parse_empty_reaction

Testet, ob eine leere Reaction interpretiert weiterhin eine leere Reaction ist.

Bestanden

test_reaction_parse_publish_reaction

Testet, ob eine publish-reaction erfolgreich interpretiert wird.

Bestanden

test_reaction_parse_wrong_action

Testet, ob die Interpretation einer Reaction mit falscher action eine leere Reaction zurückgibt.

Bestanden

test_traverse_empty_and

Testet, ob eine leere Und-Verknüpfung erfolgreich interpretiert wird.

Bestanden

test_traverse_simple_and

Testet, ob eine einfach Und-Verknüpfung korrekt interpretiert wird.

Bestanden

test_traverse_simple_and_or

Testet, ob eine Or-Verknüpfung in einer Und-Verknüpfung korrekt interpretiert wird.

Bestanden

test_traverse_simple_not

Testet, ob eine logische Invertierung korrekt interpretiert wird.

Bestanden

test_traverse_wrong_format

Testet, ob eine Verknüpfung bei der Interpretation ignoriert wird, wenn sie von falschem Format ist.

Bestanden

test_traverse_wrong_outcome

Testet, ob ein Constraint-Eintrag, dessen Outcome kein String ist, bei der Interpretation ignoriert wird.

Bestanden

Test Reaction

test_reaction_restart

Testet, ob die Aktion, einen Knoten neuzustarten, erfolgreich ausgeführt wird.

Bestanden

test_restart_no_host

Testet, ob eine Reaction eine korrekte Fehlermeldung ausgibt, falls sie keinen Host zu dem neuzustartenden Knoten findet.

Bestanden

test_restart_no_service

Testet, ob eine Reaction eine korrekte Fehlermeldung ausgibt, falls der Service zum Neustarten eines Knotens nicht verfügbar ist.

Bestanden

test_run_no_host

Testet, ob eine Reaction eine korrekte Fehlermeldung ausgibt, falls sie keinen Host zu dem Knoten, auf dessen Host ein Befehl ausgeführt werden soll, findet.

Bestanden

test_run_no_service

Testet, ob eine Reaction eine korrekte Fehlermeldung ausgibt, falls der Service zum Ausführen eines Befehls nicht verfügbar ist.

Bestanden

test_shutdown_no_host

Testet, ob eine Reaction eine korrekte Fehlermeldung ausgibt, falls sie keinen Host zu dem zu stoppenden Knoten findet.

Bestanden

test_shutdown_no_service

Testet, ob eine Reaction eine korrekte Fehlermeldung ausgibt, falls der Service zum Stoppen eines Knotens nicht verfügbar ist.

Bestanden

Test Scenario

test_constraint_timeout

Testet, ob nach dem Ausführen einer Reaction solange zur nächsten Ausführung gewartet wird, wie der Timeout angibt.

Bestanden

test_high_cpu

Testet, ob eine Reaction in einem Constraint erfolgreich ausgeführt wird, wenn Outcome HIGH für cpu_usage_max über /statistics Rated empfangen wird.

Bestanden

test_high_cpu_too_short

Testet, ob eine Reaction in einem Constraint nicht ausgeführt wird, wenn Outcome HIGH für cpu_usage_max über /statistics Rated nicht lange genug empfangen wird.

Bestanden

test_reaction_autonomy_level_too_high

Testet, ob eine Reaction in einem Constraint nicht ausgeführt wird, wenn das autonomy_level zu hoch ist.

Bestanden

Test Storage

test_add_new_than_old

Testet, ob eine zeitlich ältere Statistik keine zeitlich neueren Statistiken überschreibt, wenn die Ältere später hinzugefügt wird.

Bestanden

test_add_old

Testet, ob das Hinzufügen einer zu alten Statistik verworfen wird.

Bestanden

test_callback_multiple_entries

Testet, ob ein Aufruf mit einem bewerteten Statistik-Eintrag, welcher ein Array enthält, erfolgreich gespeichert wird.

Bestanden

test_callback_multiple_rated_entities

Testet, ob ein Aufruf mit mehreren bewerteten Statistik-Einträgen erfolgreich gespeichert wird.

Bestanden

test_callback_single_entry

Testet, ob ein Aufruf mit einem bewerteten Statistik-Eintrag erfolgreich gespeichert wird.

Bestanden

test_remove_through_timeout

Testet, ob eine hinzugefügte Statistik nach dem Timeout nicht mehr vorhanden ist.

Bestanden

2.4 Arni_Nodeinterface

Test Service

`test_kill`

Testen den Service um einen Node zu stoppen, erwartet das erfolgreiche stoppen des Nodes

Bestanden

`test_restart`

Testet das neustarten eines Nodes. Erwartet den erfolgreichen Neustart

Bestanden

Test statistical Calc

`test_cpu`

Testet ob die statistischen Größen bei gegebenen Werten richtig berechnet werden.

Bestanden

`test_cpu_core`

Testet ob die statistischen Größen bei Listen von Listen richtig berechnet werden.

Bestanden

`test_empty_list`

Testet ob bei einer leeren Listen 0 - werte ausgegeben werden.

Bestanden

3 Integrationstests

3.1 Test 1 - Gleichmäßiges Publizieren

Dieser Test prüft, ob bei stetigem Publizieren die Frequenz korrekt bewertet wird und ob auf die Bewertung korrekt reagiert wird.

1. Test-Launchfile starten: Mit `roslaunch arni_core test_1_steady.launch` wird das Launchfile gestartet.

Man sieht, dass dabei folgende Knoten gestartet werden:

```
countermeasure (arni_countermeasure/arni_countermeasure)
ninja_turtle (arni_core/predefined_subscriber.py)
node_manager (arni_nodeinterface/arni_nodeinterface)
processing (arni_processing/arni_processing)
steady_tree (arni_core/predefined_publisher.py)
```

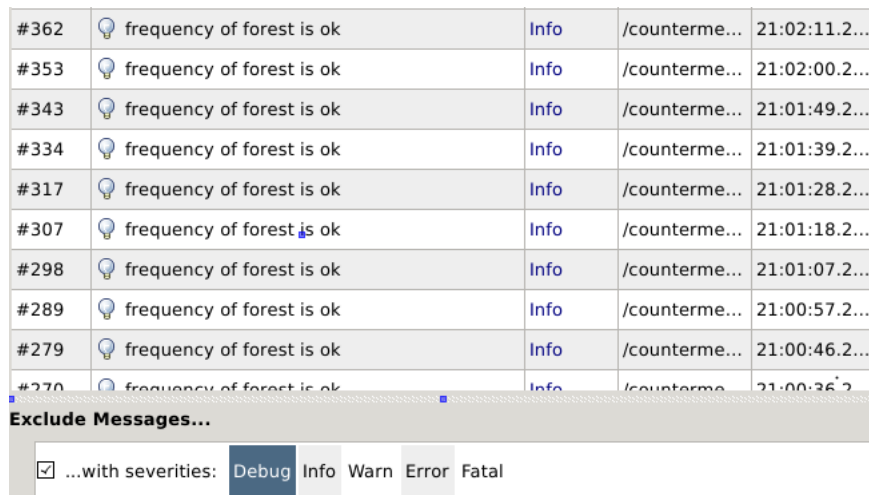
Dabei besteht folgende Verbindung der Knoten (Debugging-Knoten zur Übersichtlichkeit ausgenommen):



Abbildung 3.1: `steady_tree` publiziert mit 100Hz auf `/forest`, `ninja_turtle` abonniert `forest`.

Die Frequenz von `/forest` wird unter 80Hz als LOW und über 120Hz als HIGH bewertet. Der Countermeasure-Knoten hat das Constraint alle 10 Sekunden *frequency of forest is ok* auszugeben, falls die Frequenz mit OK bewertet wurde.

2. Öffnen der GUI: In die Konsole wird `roslaunch rqt_gui rqt_gui` eingegeben und ausgeführt.
3. Öffnen des Console Widgets: Auswählen des Widgets *Logging > Console*, Debug Messages ausblenden



#	Severity	Message	Topic	Time
#362	Info	frequency of forest is ok	/counterme...	21:02:11.2...
#353	Info	frequency of forest is ok	/counterme...	21:02:00.2...
#343	Info	frequency of forest is ok	/counterme...	21:01:49.2...
#334	Info	frequency of forest is ok	/counterme...	21:01:39.2...
#317	Info	frequency of forest is ok	/counterme...	21:01:28.2...
#307	Info	frequency of forest is ok	/counterme...	21:01:18.2...
#298	Info	frequency of forest is ok	/counterme...	21:01:07.2...
#289	Info	frequency of forest is ok	/counterme...	21:00:57.2...
#279	Info	frequency of forest is ok	/counterme...	21:00:46.2...
#270	Info	frequency of forest is ok	/counterme...	21:00:36.2...

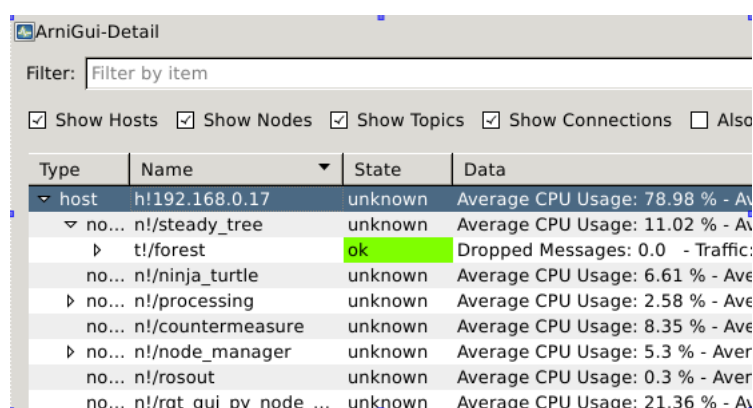
Exclude Messages...

☒ ...with severities: **Debug** Info Warn Error Fatal

Abbildung 3.2: steady_tree publiziert mit 100Hz auf forest. ninja_turtle hört zu.

Es ist zu sehen, dass die Nachricht des Countermeasure-Knotens alle 10 Sekunden publiziert wird.

4. Öffnen des Arni-Detail Widgets: Auswahl des Widgets *Introspection > Arni-Detail*



Type	Name	State	Data
host	h!192.168.0.17	unknown	Average CPU Usage: 78.98 % - Av
no...	n!/steady_tree	unknown	Average CPU Usage: 11.02 % - Av
no...	n!/forest	ok	Dropped Messages: 0.0 - Traffic:
no...	n!/ninja_turtle	unknown	Average CPU Usage: 6.61 % - Av
no...	n!/processing	unknown	Average CPU Usage: 2.58 % - Av
no...	n!/countermeasure	unknown	Average CPU Usage: 8.35 % - Av
no...	n!/node_manager	unknown	Average CPU Usage: 5.3 % - Aver
no...	n!/rosout	unknown	Average CPU Usage: 0.3 % - Aver
no...	n!/rqt_gui	unknown	Average CPU Usage: 21.36 % - Av

Abbildung 3.3: Topic forest ist OK, da es eine Spezifikation für das Topic gibt welche mit OK bewertet wird.

3.2 Test 2 - Zu niedrige Frequenz

Dieser Test prüft das Verhalten beim Publizieren mit geringerer Frequenz als durch die Spezifikationen erwartet wird und ob auf die Bewertung korrekt reagiert wird.

1. Test-Launchfile starten: Mit `roslaunch arni_core test_2_steady_low.launch` wird das Launchfile gestartet.

Man sieht, dass dabei folgende Knoten gestartet werden:

```
countermeasure (arni_countermeasure/arni_countermeasure)
leopard_seal (arni_core/predefined_subscriber.py)
node_manager (arni_nodeinterface/arni_nodeinterface)
processing (arni_processing/arni_processing)
breathing_penguin (arni_core/predefined_publisher.py)
```

Dabei besteht folgende Verbindung der Knoten (Debugging-Knoten zur Übersichtlichkeit ausgenommen):

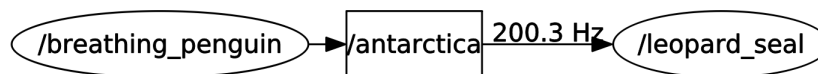


Abbildung 3.4: `breathing_penguin` publiziert mit 200Hz auf `/antarctica`, `leopard_seal` abonniert `antarctica`.

Die Frequenz von `/antarctica` wird unter 400Hz als LOW und über 600Hz als HIGH bewertet. Der Countermeasure-Knoten hat das Constraint alle 5 Sekunden *frequency of antarctica is too low* auszugeben, falls die Frequenz mit LOW bewertet wurde.

2. Öffnen der GUI: In die Konsole wird `roslaunch rqt_gui rqt_gui` eingegeben und ausgeführt.
3. Öffnen der Widgets: Auswählen des Widgets *Logging > Console*, Debug Messages ausblenden

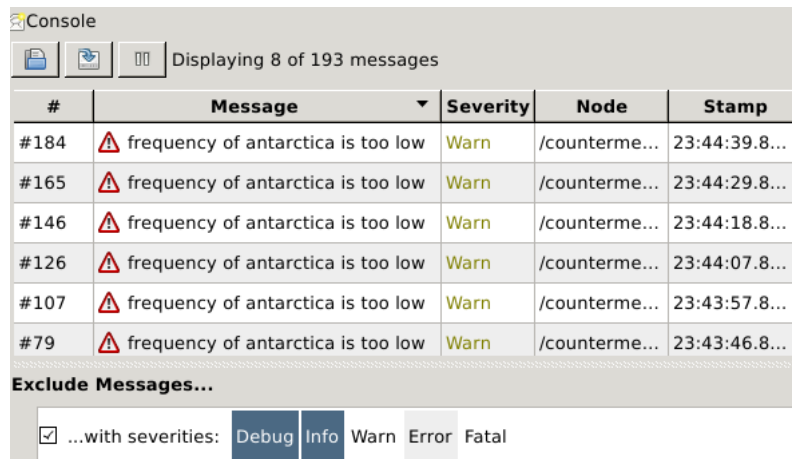


Abbildung 3.5: breathing_penguin publiziert mit 200Hz auf /antarctica. leopard_seal hört zu.

Es ist zu sehen, dass die Nachricht des Countermeasure-Knotens alle 5 Sekunden publiziert wird.

4. Öffnen des Arni-Detail Widgets: Auswahl des Widgets *Introspection > Arni-Detail*

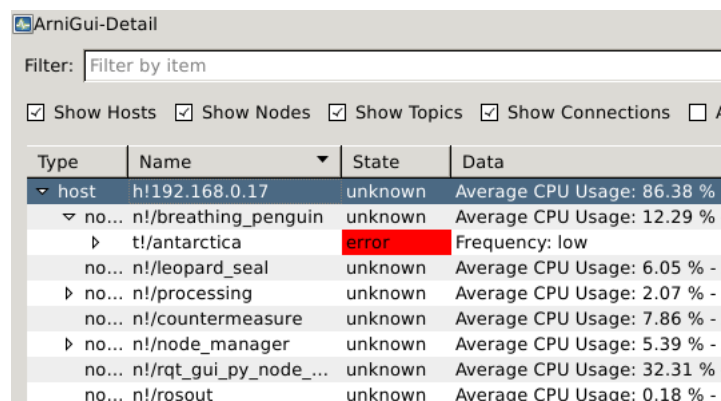


Abbildung 3.6: Topic forest wird als ERROR angezeigt, da es eine Spezifikation für das Topic gibt welche mit LOW bewertet wird.

3.3 Test 3 - Variierende Frequenz

Dieser Test prüft das Verhalten beim Publizieren mit variierender Frequenz in einer Sinuskurve, wobei hohe Werte als zu hoch und niedrige Werte als zu niedrig bewertet werden.

1. Test-Launchfile starten: Mit `roslaunch arni_core test_3_fluctuation.launch` wird das Launchfile gestartet.

Man sieht, dass dabei folgende Knoten gestartet werden:

```
countermeasure (arni_countermeasure/arni_countermeasure)
sailing_boat (arni_core/predefined_subscriber.py)
node_manager (arni_nodeinterface/arni_nodeinterface)
processing (arni_processing/arni_processing)
fluctuation_tide (arni_core/predefined_publisher.py)
```

Dabei besteht folgende Verbindung der Knoten (Debugging-Knoten zur Übersichtlichkeit ausgenommen):

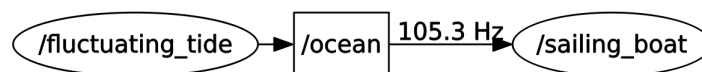
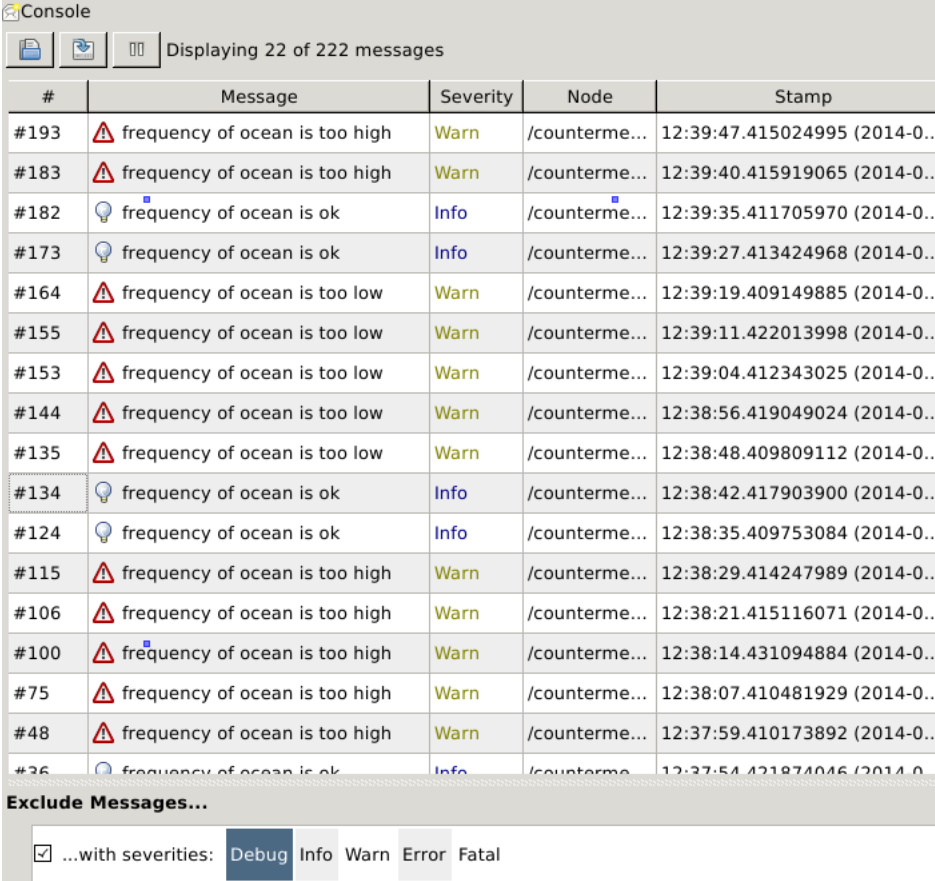


Abbildung 3.7: `fluctuation_tide` publiziert mit einer Frequenz zwischen 10 und 190 auf `/ocean`, `sailing_boat` abonniert `ocean`.

Die Frequenz von `/ocean` wird unter 70Hz als LOW und über 130Hz als HIGH bewertet. Unterschreitet die aufgezeichnete Frequenz die Grenze, wird *frequency of ocean is too low* ausgegeben, *frequency of ocean is too high*, wenn die Frequenz 130Hz überschreitet und *frequency of ocean is ok* sonst.

2. Öffnen der GUI: In die Konsole wird `roslaunch rqt_gui rqt_gui` eingegeben und ausgeführt.
3. Öffnen der Widgets: Auswählen des Widgets *Logging > Console*, Debug Messages ausblenden



#	Message	Severity	Node	Stamp
#193	frequency of ocean is too high	Warn	/counterme...	12:39:47.415024995 (2014-0...
#183	frequency of ocean is too high	Warn	/counterme...	12:39:40.415919065 (2014-0...
#182	frequency of ocean is ok	Info	/counterme...	12:39:35.411705970 (2014-0...
#173	frequency of ocean is ok	Info	/counterme...	12:39:27.413424968 (2014-0...
#164	frequency of ocean is too low	Warn	/counterme...	12:39:19.409149885 (2014-0...
#155	frequency of ocean is too low	Warn	/counterme...	12:39:11.422013998 (2014-0...
#153	frequency of ocean is too low	Warn	/counterme...	12:39:04.412343025 (2014-0...
#144	frequency of ocean is too low	Warn	/counterme...	12:38:56.419049024 (2014-0...
#135	frequency of ocean is too low	Warn	/counterme...	12:38:48.409809112 (2014-0...
#134	frequency of ocean is ok	Info	/counterme...	12:38:42.417903900 (2014-0...
#124	frequency of ocean is ok	Info	/counterme...	12:38:35.409753084 (2014-0...
#115	frequency of ocean is too high	Warn	/counterme...	12:38:29.414247989 (2014-0...
#106	frequency of ocean is too high	Warn	/counterme...	12:38:21.415116071 (2014-0...
#100	frequency of ocean is too high	Warn	/counterme...	12:38:14.431094884 (2014-0...
#75	frequency of ocean is too high	Warn	/counterme...	12:38:07.410481929 (2014-0...
#48	frequency of ocean is too high	Warn	/counterme...	12:37:59.410173892 (2014-0...
#36	frequency of ocean is ok	Info	/counterme...	12:37:54.421874046 (2014-0...

Exclude Messages...

☒ ...with severities: **Debug** Info Warn Error Fatal

Abbildung 3.8: fluctuation_tide publiziert mit 10 - 190Hz auf /ocean. sailing_boat hört zu.

Es ist zu sehen, dass sich *[...] is ok*, *[...] is too high*, *[...] is ok*, *[...] is too low*, etc. abwechseln.

4. Öffnen der Arni-Widgets: Auswählen des Widgets *Introspection > Arni-Detail*, bei Filter wird `ocean` eingegeben und die Eingabe bestätigt. Im Baum wird `t!/ocean` ausgewählt. Im Fenster auf der rechten Seite wird der Reiter *Graphs* ausgewählt. Darunter wird Range auf 60 Seconds und Selected auf `delivered_msgs`, `frequency` gestellt. Man sieht eine stufige Sinuskurve, die sich langsam ausbreitet.



Abbildung 3.9: Die Graphen für übertragene Nachrichten und die Frequenz bilden eine Sinuskurve.

3.4 Test 4 - Neustarten eines Knotens mit behobenem Fehler

Der Knoten in diesem Test sendet nach 100 Sekunden mit stark reduzierter Frequenz und wird daraufhin vom Countermeasure Knoten neugestartet.

1. Test-Launchfile starten: Mit `roslaunch arni_core test_4_restarting.launch` wird das Launchfile gestartet.

Man sieht, dass dabei folgende Knoten gestartet werden:

```
countermeasure (arni_countermeasure/arni_countermeasure)
sturbacks (arni_core/predefined_subscriber.py)
node_manager (arni_nodeinterface/arni_nodeinterface)
processing (arni_processing/arni_processing)
jumping_tower (arni_core/predefined_publisher.py)
```

Dabei besteht folgende Verbindung der Knoten (Debugging-Knoten zur Übersichtlichkeit ausgenommen):

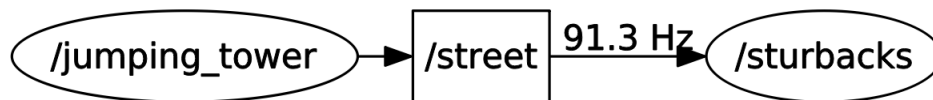


Abbildung 3.10: jumping_tower publiziert für 100 Sekunden mit 100Hz auf /street, sturbacks abonniert street.

Die Frequenz von /street wird unter 80Hz als LOW bewertet. Ist die Frequenz zwischen 80Hz und 120Hz wird alle 7 Sekunden eine Reaction des Countermeasure Knotens ausgelöst und *frequency of street is ok* geloggt. Unterschreitet die aufgezeichnete Frequenz die Grenze, wird *frequency of street is too low - trying to restart* ausgegeben und der Knoten wird neugestartet.

Danach ist die Frequenz wieder Ok und es wird *frequency of street is ok* geloggt.

2. Öffnen der GUI: In die Konsole wird `roslaunch rqt_gui rqt_gui` eingegeben und ausgeführt.
3. Öffnen der Widgets: Auswählen des Widgets *Logging > Console*, Debug Messages ausblenden

#	Message	Severity	Node	Stamp
#2515	💡 frequency of street is ok	Info	/counterme...	00:19:15.7...
#2482	⚠ frequency of street is too low - trying to restart	Warn	/counterme...	00:19:02.7...
#2462	💡 frequency of street is ok	Info	/counterme...	00:18:50.7...
#2443	💡 frequency of street is ok	Info	/counterme...	00:18:43.7...
#2432	💡 frequency of street is ok	Info	/counterme...	00:18:35.7...
#2413	💡 frequency of street is ok	Info	/counterme...	00:18:28.7...
#2403	💡 frequency of street is ok	Info	/counterme...	00:18:20.7...
#2384	💡 frequency of street is ok	Info	/counterme...	00:18:13.7...
#2373	💡 frequency of street is ok	Info	/counterme...	00:18:06.7...
#2354	💡 frequency of street is ok	Info	/counterme...	00:17:58.7...
#2344	💡 frequency of street is ok	Info	/counterme...	00:17:50.7...
#2325	💡 frequency of street is ok	Info	/counterme...	00:17:42.7...
#2305	💡 frequency of street is ok	Info	/counterme...	00:17:34.7...
#2295	💡 frequency of street is ok	Info	/counterme...	00:17:26.7...
#2276	💡 frequency of street is ok	Info	/counterme...	00:17:18.7...
#2251	⚠ frequency of street is too low - trying to restart	Warn	/counterme...	00:17:05.7...
#2214	💡 frequency of street is ok	Info	/counterme...	00:16:49.7...
#2204	💡 frequency of street is ok	Info	/counterme...	00:16:42.7...
#2184	💡 frequency of street is ok	Info	/counterme...	00:16:34.7...

Abbildung 3.11: Es ist zu sehen das die Frequenz nach dem neustarten des Knotens wieder Ok ist.

4. Öffnen des Arni-Widgets: Analog zu Test 3.

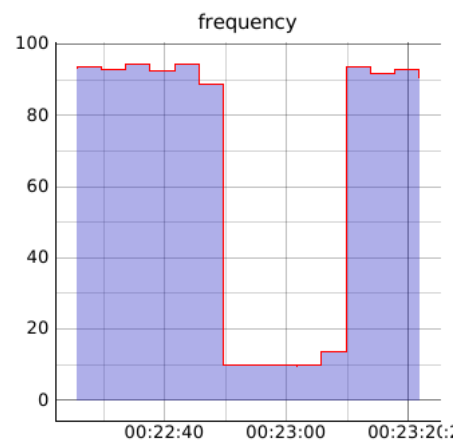


Abbildung 3.12: Es ist zu sehen wie die Frequenz auf 10Hz sinkt und durch den neustart des Knotens wieder auf 90Hz steigt.

3.5 Test 5 - Neustarten eines Knotens mit bestehendem Fehler

Der Knoten in diesem Test sendet mit reduzierter Frequenz und wird daraufhin vom Countermeasure Knoten neugestartet. Daraufhin sendet der Knoten weiterhin mit reduzierter Frequenz.

1. Test-Launchfile starten: Mit `roslaunch test_5_restart_not_helping.launch` wird das Launchfile gestartet.

Man sieht, dass dabei folgende Knoten gestartet werden:

```
airplane (arni_core/predefined_subscriber.py)
countermeasure (arni_countermeasure/arni_countermeasure)
hawk (arni_core/predefined_publisher.py)
node_manager (arni_nodeinterface/arni_nodeinterface)
processing (arni_processing/arni_processing)
```

Dabei besteht folgende Verbindung der Knoten (Debugging-Knoten zur Übersichtlichkeit ausgenommen):



Abbildung 3.13: hawk publiziert mit 476Hz auf /storm, airplane abonniert storm.

Die Frequenz von `/storm` wird unter 800Hz als LOW bewertet. Da auf dem Topic `/storm` mit weniger als 800Hz publiziert wird publiziert der Countermeasure Knoten *frequency of storm is too low*. Alle 60 Sekunden startet der Countermeasure Knoten den Knoten hawk neu (und loggt dies mittels der Nachricht *frequency of storm is too low - trying to restart*). Nach dem neustart publiziert hawk daraufhin weiter mit 500Hz.

2. Öffnen der GUI: In die Konsole wird `roslaunch rqt_gui rqt_gui` eingegeben und ausgeführt.
3. Öffnen der Widgets: Auswählen des Widgets *Logging > Console*, Debug Messages ausblenden




















#	Message	Severity	Node	Stamp
#310	 frequency of storm is low	Warn	/counterme...	00:47:37.6...
#309	 frequency of storm is low	Warn	/counterme...	00:47:24.6...
#307	 frequency of storm is low	Warn	/counterme...	00:47:12.6...
#306	 frequency of storm is low	Warn	/counterme...	00:47:00.6...
#303	 frequency of storm is too low - trying to restart	Warn	/counterme...	00:46:58.6...
#302	 frequency of storm is low	Warn	/counterme...	00:46:47.6...
#300	 frequency of storm is low	Warn	/counterme...	00:46:35.6...
#299	 frequency of storm is low	Warn	/counterme...	00:46:22.6...
#297	 frequency of storm is low	Warn	/counterme...	00:46:09.6...
#284	 frequency of storm is too low - trying to restart	Warn	/counterme...	00:45:58.6...
#283	 frequency of storm is low	Warn	/counterme...	00:45:57.6...
#264	 frequency of storm is low	Warn	/counterme...	00:45:44.6...
#235	 frequency of storm is low	Warn	/counterme...	00:45:32.6...
#216	 frequency of storm is low	Warn	/counterme...	00:45:19.6...
#187	 frequency of storm is low	Warn	/counterme...	00:45:06.6...
#164	 frequency of storm is too low - trying to restart	Warn	/counterme...	00:44:57.6...
#163	 frequency of storm is low	Warn	/counterme...	00:44:54.6...
#134	 frequency of storm is low	Warn	/counterme...	00:44:41.6...
#105	 frequency of storm is low	Warn	/counterme...	00:44:38.6...

Abbildung 3.14: Es ist zu sehen das die Frequenz nach dem neustarten des Knotens weiterhin zu gering ist.

4 Code-Abdeckung

Da sich die Code-Abdeckung unter ROS nur sehr bedingt testen lässt, konnten nur die nachfolgenden Ergebnisse gesammelt werden. Automatische Tests konnten nicht analysiert werden, sodass die Datenerhebung lediglich durch Probieren von Testszenarien stattfinden konnte, was in einigen Bereichen die von Unittests abgedeckten Abschnitte nicht vollständig erreicht.

Die Daten wurden mit dem Tool *coverage.py*¹ in den näher beschriebenen Szenarien erhoben. Dazu wurde das Start-Script der Knoten ausgeführt und abschließend der Report der Line-Coverage auf die relevanten Dateien gekürzt und in Diagrammen dargestellt.

4.1 Nodeinterface

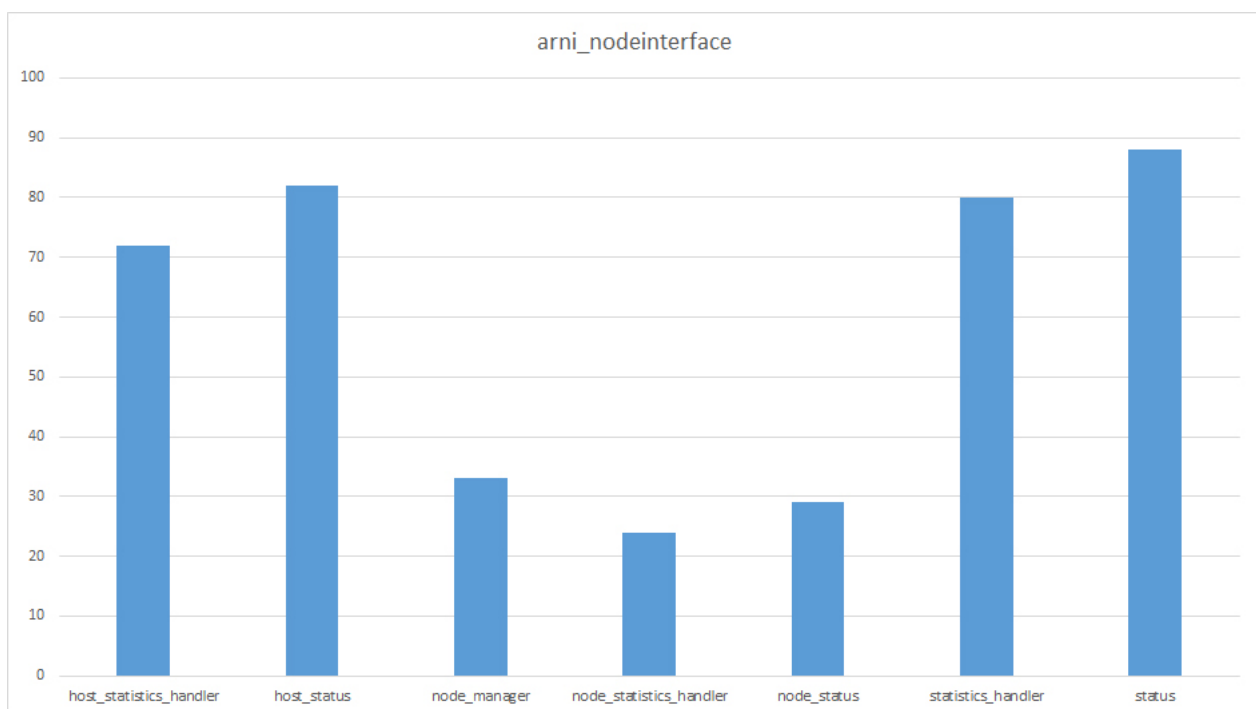


Abbildung 4.1: Die dargestellten Daten wurden im Normalbetrieb des Nodeinterface Knotens erhoben, wobei äußere Zugriffe auf die Services nicht dargestellt werden konnten, weshalb entsprechende Bereiche unterrepräsentiert sind.

¹Coverage lässt sich über `pip install coverage` installieren.

4.2 Processing

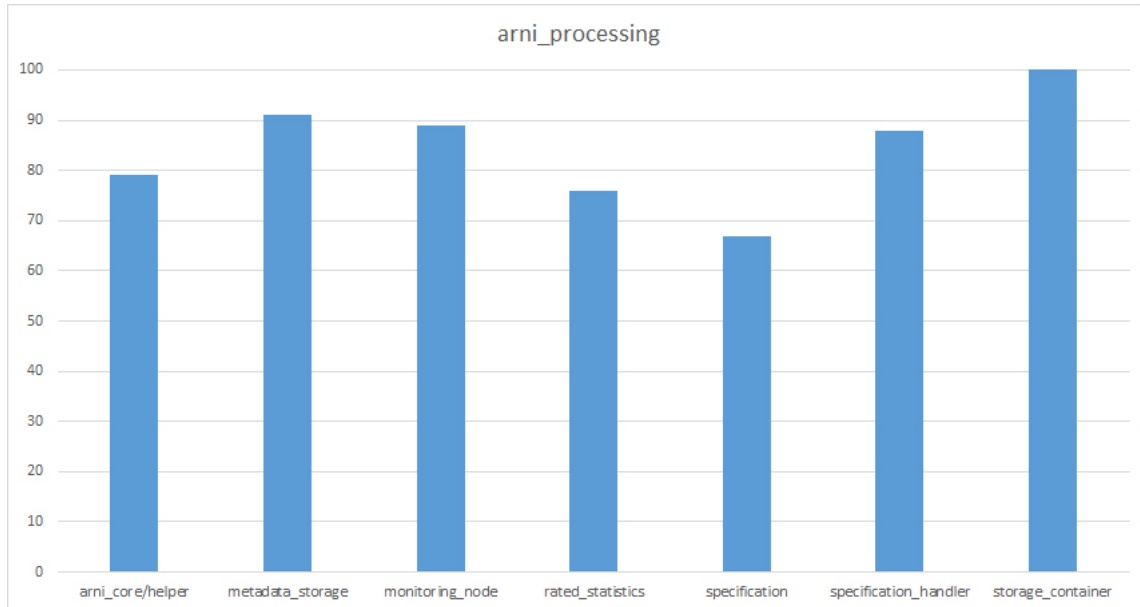


Abbildung 4.2: Zur Erhebung der dargestellten Daten liefen mehrere Knoten (arni_core/tutorial_*) und ein Nodeinterface. Nachträglich wurde die GUI gestartet, die sich Daten über einen Service holte. Außerdem wurden verzögert Spezifikationen geladen, die auch einen Service bnuzten. Nach einer Weile wurde ein Knoten abgeschaltet, woraufhin Nachrichten über seine Fehlfunktion gesendet wurden.

4.3 Countermeasure

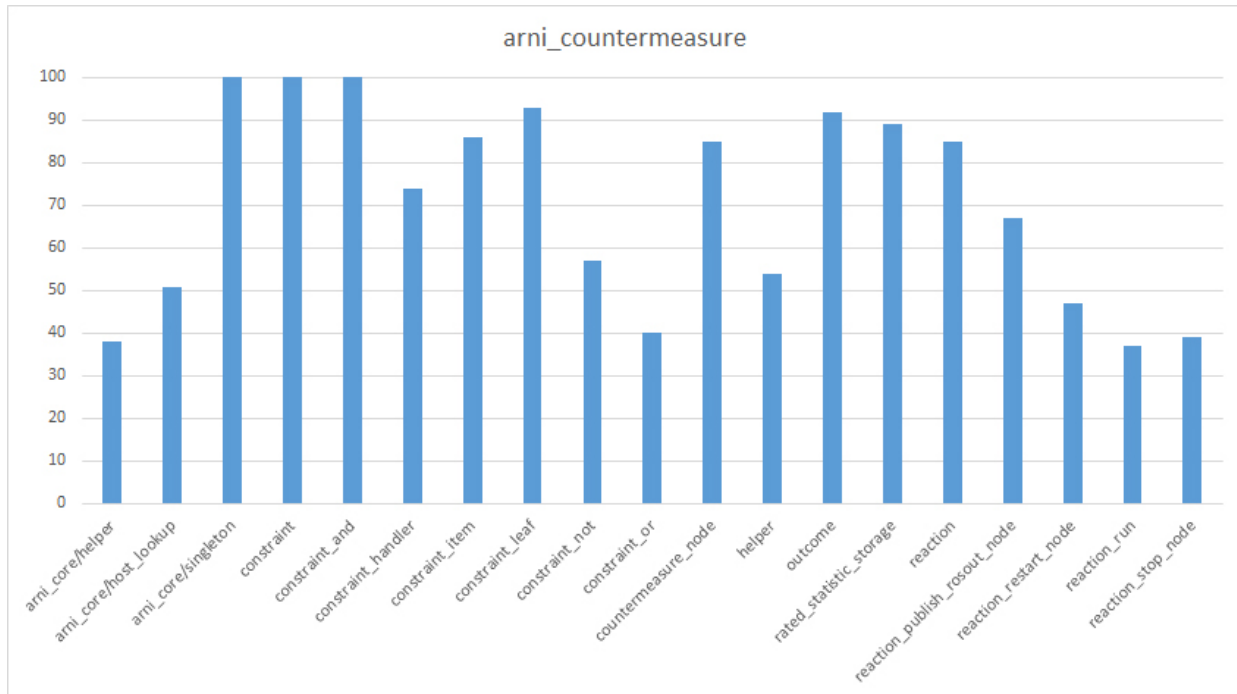


Abbildung 4.3: Die dargestellten Werte wurden erreicht, indem Integrationstest 4 durchgeführt wurde.

4.4 GUI

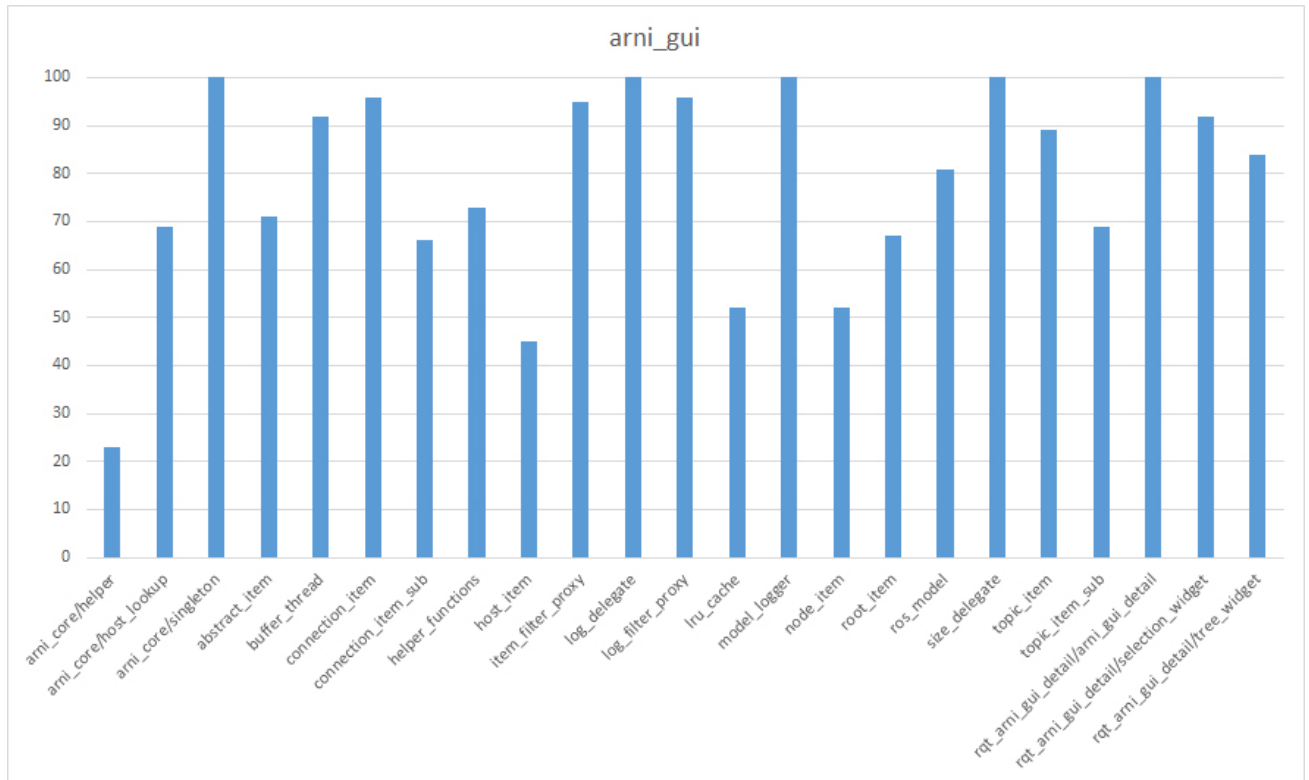


Abbildung 4.4: Die dargestellten Werte wurden erreicht, indem Integrationstest 4 und Arni-Detail betrachtet wurde. Dabei wurden strukturiert alle Features der Übersichtsliste durchgegangen, beinhaltend Filter und Ansichtsoptionen. Anschließend wurden verschiedene Itemtypen ausgewählt. Die einzelnen Tabs der Detailanzeige wurden geöffnet und verfügbare Steuerelemente benutzt.