BaSys

Basissystem Industrie 4.2

D-2.7 Spezifikation Generisches Architekturrahmenwerk für Machbarkeitstests

Dokument-Version	0.1
Datum	11.02.2021
Verbreitungsgrad	Öffentlich
Projekt	BaSys 4.2
Förderkennzeichen	01 IS 190 022
Laufzeit	1.7.2019 – 30.6.2022

GEFÖRDERT VOM

Bundesministerium
für Bildung
und Forschung

Änderungsverzeichnis

Änderung			Geänderte Abschnitte	Autoren	Beschreibung
Nr.	Datum	Version			
1	22.01.21	0.1	Alle	Blohm P.	Erste Version
2	28.01.21	0.2	Alle	Blohm P., Weser M.	Letzte Korrekturen eingearbeitet

Editoren

Dr. Phillip Blohm, KUKA Deutschland GmbH

Michael Weser, KUKA Deutschland GmbH

Autoren

Dr. Phillip Blohm, KUKA Deutschland GmbH

Dr. Jürgen Bock, KUKA Deutschland GmbH

Michael Weser, KUKA Deutschland GmbH

Alexander David, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Sönke Knoch, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

William Motsch, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Tobias Klausmann, Lenze Automation GmbH

Reviewer

Siwara Schmitt, Fraunhofer-Institut für Experimentelles Software Engineering IESE

Torben Miny, RWTH Aachen

Kurzfassung

Mit Fortschreiten von Industrie 4.0 und Digitalisierung entwickelt sich die klassische Fabrik zunehmend in eine Welt der Maschine-Maschine-Kommunikation, in der Maschinen dynamisch kollaborative Aufgaben planen. Um andere Maschinen zu finden, die in solchen Plänen zum Einsatz kommen können, bedarf es eines Prozesses zur Beurteilung der Verwendbarkeit von Ressourcen für bestimmte Aufgaben. In Basys 4.0 wurde zu diesem Zweck eine Ontologie zur semantischen Beschreibung von Fähigkeiten (Capabilities) von Ressourcen erstellt, die verwendet werden kann um symbolisch die Menge an Aufgaben zu modellieren, zu der eine Ressource fähig ist.

In BaSys 4.2 soll dieser Ansatz aufgenommen und erweitert werden. Neben dem Überprüfen von allgemeinen Fähigkeiten sollen auch konkrete Machbarkeitstest (Feasibility Checks) in den Prozess mit aufgenommen werden. Hierbei handelt es sich um umfangreichere Verifikationen, die über die grundsätzliche Fähigkeit eine Aufgabe auszuführen hinausgehen und auch die gegebenen Umstände mit in Betracht ziehen.

Diese Art von Checks kann neben der reinen logischen Inferenz auf symbolischem Level auch komplexere numerische Ansätze und vollständige Simulationen miteinschließen. Basierend auf dem jeweiligen Szenario können die Feasibility Checks sehr unterschiedlich umgesetzt sein und sich auch auf externe Software-Tools stützen, die mit ihren ganz eigenen Anforderungen einhergehen. Um diese Breite an Anforderungen abdecken zu können, wird in diesem Deliverable ein generisches und flexibles Capability- und Feasibility-Checker-Framework entworfen, das die benötigte heterogene Checker-Landschaft hinter einer gemeinsamen Schnittstelle vereint.

Hierfür werden im Folgenden zunächst die Anforderungen an solch ein Framework dargestellt, ehe der Entwurf der Architektur und der unterstützenden Werkzeuge vorgestellt werden und explizit auf die Schnittstellen eingegangen wird.

Inhaltsverzeichnis

1 Glossar

Aufgabe	Eine Aufgabe ist ein Prozess aus Anforderungssicht ohne Vorgabe einer Lösung. Bei einer Aufgabe sind ein Anfangszustand und ein Zielzustand festgelegt. Der Übergang zwischen diesen ist unbekannt. Durch Bedingungen an den Übergang kann der Lösungsraum der Aufgabe eingeschränkt werden.
Broker-Service	Service, bei dem sich Checker an- und abmelden und der eine anfragende Applikation an passende Checker vermittelt.
Capability	Beschreibung von Fähigkeit einer Ressource, die verwendet werden kann um symbolisch die Menge an Aufgaben zu modellieren, zu der diese Ressource fähig ist.
Capability Check	Kontextfreier Fähigkeitstest einer Ressource ob sie eine gegebene Aufgabe ausführen kann.
Capability- & Feasibility-Checker-Framework	Das in diesem Dokument beschriebene Framework bestehend aus Broker-Service, Checker-Services und einem Toolkit zur Erstellung von Checker-Services.
Checker	Siehe Checker-Service.
Checker-Framework	Siehe Capability- & Feasibility-Checker-Framework.
Checker-Service	Service, der Szenarien checkt und Auskunft über Capabilities und Feasibilities gibt.
Closed-World-Assumption	Die Closed-World-Assumption sagt aus, dass alles, was nicht explizit als wahr bewiesen werden kann, als falsch bezeichnet wird.
Feasibility	Umsetzbarkeit eines Szenarios mit einer bestimmten Ressource.
Feasibility Check	Machbarkeitstest eines Szenarios, das über die grundsätzliche Fähigkeit einer Ressource eine Aufgabe auszuführen hinausgeht und auch die gegebenen Umstände mit in Betracht zieht.
Open-World-Assumption	Die Open-World-Assumption sagt aus, dass eine Aussage wahr sein kann unabhängig davon, ob es bekannt ist, dass sie wahr ist.
PPR-Modell	Entsprechend der Definition in [1]. Ein Modell zur Beschreibung von Prozessen, in denen Ressourcen Produkte verarbeiten.
Ressource	Entsprechend der Definition im PPR-Modell [1]. Eine Software- oder Hardware-Komponente, die in einem Prozess ein Produkt verarbeitet. Entspricht einem Asset im Kontext der Verwaltungsschale.
Szenario	Die Summe der übergebenen Informationen an einen Checker, die die PPR- sowie möglicherweise Kontextinformationen enthält.

2 Anforderungen

2.1 A-1 Kompatibilität mit PPR-Modellierung

Zur Formalisierung von Automatisierungssystemen existiert das sogenannte PPR-Modell [1], das zwischen Produkten, Prozessen und Ressourcen unterscheiden. Ressourcen führen hierbei Prozesse aus, um Produkte herzustellen.

Da sich diese Darstellungsweise in der Domäne bewährt hat, hat sich das Konsortium entschieden auch in BaSys 4.2 auf eine entsprechende Modellierung zu setzen und in Arbeitspaket 2.2 entsprechend zu formalisieren. Um mit den restlichen Informationen und Verfahren innerhalb des Projekts kompatibel zu sein, sollte das Checker-Framework ebenfalls auf dem PPR-Modell aufbauen.

Die Ressourcen sollen in diesem Kontext vom Checker-Framework dahingehend überprüft werden, ob sie im gegebenen Szenario in der Lage sind, bestimmte Prozesse durchzuführen, um die entsprechenden Produkte herzustellen.

2.2 A-2 Kompatibilität mit Verwaltungsschale

Zur Modellierung von Software- und Hardware-Komponenten in der Industrie 4.0-Welt hat die Plattform Industrie 4.0 mit der Verwaltungsschale einen Beschreibungs- und Kommunikationsstandard veröffentlicht [2].

Im BaSys Middleware-Konzept wurde auf dem Standard aufgesetzt, um Fähigkeiten als Teilmodelle zu beschreiben. Produkte, Ressourcen und Prozesse können als Assets mit ihren Verwaltungsschalen abgebildet werden, die wiederum Informationsmodelle in Form von Teilmodellen besitzen können. Entsprechend sollte auch das Checker-Framework mit solchen Asset-Beschreibungen und der Tool-Landschaft rund um die Verwaltungsschale, vor allem der Open Source Referenzimplementierung des BaSys-Middleware-Konzept Eclipse BaSyx, kompatibel sein.

Die einzelnen Services, die das Checker-Framework ausmachen, sind im Verwaltungsschalenkontext Infrastruktur-Komponenten, die keine eigene Verwaltungsschale benötigen. Die Ausnahme bilden Checks, die sich exklusiv auf Informationen einer Komponente beziehen und vom Herausgeber der Verwaltungsschale der Komponente bereitgestellt werden. Hier soll es zulässig sein, den zugehörigen Checker als Teilmodell der Komponente zu entwickeln.

2.3 A-3 Selbstbeschreibung von Checkern

Ein Checker überprüft einen Prozess stets anhand eines vereinfachten Modells der Welt, das auf bestimmten Annahmen über diese beruht. So können Checker, die auf semantischem Reasoning beruhen beispielsweise die Open-World-Assumption zugrunde legen, während andere Checker darauf angewiesen sind alle zu betrachtenden Informationen zu erhalten und entsprechend unter der Annahme einer Closed World arbeiten.

Das Checker-Framework soll grundsätzlich verschiedene Checker zulassen. Um für den Anwender jedoch transparent zu halten unter welchen Annahmen die Bewertung des Checkers zustande gekommen ist, soll jeder Checker eine Selbstbeschreibungsfunktion bereitstellen, in der seine Annahmen und möglicherweise andere relevante Aspekte, wie Funktions- und Implementierungsdetails, beschrieben werden.

2.4 A-4 Verlässlichkeit des Checks

Während rein logische Checker stets eindeutige Antworten liefern, ist die Beurteilung durch einen probabilistischen Checker eher eine Vorhersage oder Schätzung, die mit einer bestimmten Konfidenz getroffen wird. Da auch probabilistische Checker nützlich sein können, z.B. in Situationen mit unvollständiger Information und geringen Kosten für eine fehlgeschlagene Aktion, sollen auch diese vom Checker-Framework unterstützt werden.

Um im Falle eines probabilistischen Checkers die Verlässlichkeit der Aussage an den Nutzer zu kommunizieren, soll ein Checker neben seiner Beurteilung stets auch einen Konfidenzwert zurückliefern.

2.5 A-5 Verteiltes Framework

In der Praxis soll das Checker-Framework mit zahlreichen Checkern von Komponenten- und Drittanbietern ausgeführt werden. Um dieses zu unterstützen soll mit dem Checker-Framework, das die Infrastruktur bereitstellt, ein Toolkit entwickelt werden, das bei der Entwicklung von eigenen Checkern unterstützt. Um Checker auffinden zu können, soll ein zentraler Mechanismus entwickelt werden, an dem sich Checker an- und abmelden können. Hier soll eine Liste mit zu angegebenen Szenarien, Fähigkeiten von Checkern oder IDs passenden Checker-Services zurückgeliefert werden. Dieser zentrale Punkt muss stabil entwickelt und redundant ausführbar sein, um Ausfallzeiten, die die gesamte Kommunikation blockieren, möglichst gering zu halten. Außerdem muss das Framework mit Ausfällen von einzelnen Checkern umgehen können.

Um Doppelentwicklungen zu vermeiden und um graduell immer komplexere Checks zu ermöglich, soll es möglich sein, dass Checker Teilaufgaben an andere Checker weiterdelegieren.

In der Industrie 4.0 erhält jede Komponente ihre eigene Verwaltungsschale, mit der gesondert kommuniziert werden kann. Hieraus ergibt sich die Situation, dass bestimmte Informationen verteilt statt an einer zentralen Stelle abfragbar sind. Dieser Umstand muss auch beim Design des Checker-Frameworks beachtet werden.

2.6 A-6 Anforderungen an Rückgabe

Neben der Information, ob der Check erfolgreich war und dem Konfidenzwert, soll die Rückgabe eines Capability- oder Feasibility-Checks im Negativfall optional für den Fall, dass ein Checker diese Information liefern kann, auch eine Begründung zurückliefern, woran der Check gescheitert ist.

2.7 A-7 Kompatibilität mit Planern

Bei den Diskussionen im Konsortium wurde klar, dass das angedachte Checker-Framework einen starken Bezug zur Planungsdomäne hat, in der ebenfalls bestimmte Sachverhalte auf ihren Wahrheitsgehalt oder ihre Erfüllbarkeit geprüft werden. Um voneinander zu profitieren statt Dinge, die sich bewährt haben, neu zu erfinden, soll das Checker-Framework grundsätzlich zusammen mit Planer-Software funktionieren und die Pre- und Post-Condition-Checks in diesen übernehmen können. So lässt die Abbildung der Domäne innerhalb des Planers nur einen gewissen Detailgrad zu. Feasibility-Checks können eingesetzt werden, um den Detailgrad zu erhöhen und die Qualität der Pläne zu erhöhen oder Pläne vorab virtuell zu testen.

2.8 A-8 Checker-Services mit Zuständen

In bestimmten Situationen kann es nützlich sein, dass ein Checker einen Zustand vorhält, da z.B. verschiedene Checks auf sehr ähnlichen Szenarien durchgeführt werden und andernfalls mehrfach hohe Initialisierungsaufwände entstehen. So könnte es notwendig sein, verschiedene Bewegungen eines Roboters in einem ansonsten gleichen Szenario in einem Feasibility-Checker, der auf einer Simulationssoftware basiert, zu testen. Wenn das Laden des Szenarios in die Software zeitaufwändig, das Austauschen der Bewegungen aber günstig ist, ist es effizienter, wenn der Simulationssoftware und damit auch dem Feasibility-Checker erlaubt wird, einen Zustand vorzuhalten und so die teure Lade-Operation nur einmal durchzuführen.

2.9 Übersicht Anforderungen

ID	Titel	Anforderung
A-1.1	PPR-Kompatibilität	Das Checker-Framework muss mit den in anderen Deliverables entstehenden, auf dem PPR-Modell basierenden Beschreibungen von Produkten, Ressourcen und Prozessen kompatibel sein. Es muss möglich sein Checker-Services zu implementieren, die derart beschriebene Szenarien verarbeiten können.
A-2.1	Nutzung von Fähigkeiten als Submodels	Es muss möglich sein Checker-Services zu implementieren, die Fähigkeiten, die als Verwaltungsschalen-Submodels beschrieben sind, für ihre Checks nutzen.
A-2.2	Nutzung von Produkten, Ressourcen und Prozessen als Assets	Es muss möglich sein Checker-Services zu implementieren, die Produkte, Ressourcen und Prozesse, die als Verwaltungsschalen-Assets beschrieben sind, für ihre Checks nutzen.
A-2.3	BaSyx-Kompatibilität	Das Checker-Framework muss mit Eclipse BaSyx zusammen betrieben werden können.

A-2.4	Checker als Teilmodell	Für Szenarien, die nur auf Informationen aus einer Verwaltungsschale beruhen, soll es möglich sein Checker als Teilmodelle implementieren zu können.
A-3.1	Check-Annahmen kommunizieren	Es muss möglich sein Checker-Services zu implementieren, die Auskunft über die bei ihrem Check getroffenen Annahmen geben.
A-3.2	Funktions- und Implementierungsdetails kommunizieren	Es muss möglich sein Checker-Services zu implementieren, die Auskunft über Funktions- und Implementierungsdetails geben.
A-4.1	Unterstützung logischer Checks	Es muss möglich sein Checker-Services zu implementieren, die ein Szenario mithilfe von Logik überprüfen.
A-4.2	Unterstützung probabilistischer Checks	Es muss möglich sein Checker-Services zu implementieren, die ein Szenario mithilfe probabilistischer Methoden beurteilen.
A-4.3	Bereitstellung Konfidenzwert	Jeder Checker-Service muss im Ergebnis einen Konfidenzwert zur Beurteilung des gelieferten Ergebnisses mitliefern.
A-5.1	Erstellung von Dritten	Es muss für Drittanbieter möglich sein eigene Checker-Services zu implementieren.
A-5.2	Toolkit zur Implementierung von Checker-Services	Es muss ein Toolkit bereitgestellt werden, dass bei der Implementierung von Checkern unterstützt.
A-5.3	Zentrale An- und Abmeldung	Das Checker-Framework muss einen zentralen Service zur An- und Abmeldung von internen und externen Checker-Services bereitstellen.
A-5.4	Checker-ID	Jeder Checker-Service muss eine ID haben, mit der er eindeutig identifiziert werden kann.
A-5.5	Checker-Fähigkeiten	Es muss möglich sein Checker-Services mit Fähigkeiten (Capabilities) zu versehen, die zum Auffinden des Checkers verwendet werden können.
A-5.6	Checker-Identifikation anhand von Szenario	Es muss möglich sein eine Liste von Checkern zu erhalten, die in der Lage sind ein gegebenes Szenario zu checken.
A-5.7	Robustheit System	Der Service zur An- und Abmeldung von Checker-Services muss ausfallsicher sein. Das Framework muss mit Ausfällen von einzelnen Services umgehen können.
A-5.8	Delegieren von Teilaufgaben	Es muss möglich sein Checker-Services innerhalb von anderen Checker-Services nutzen zu können um Teilaufgaben zu lösen.
A-5.9	Kompatibilität verteilte Informationen in Verwaltungsschale	Das Checker-Framework muss kompatibel mit dem verteilten Datenhaltungsmodell der Verwaltungsschale sein.
A-6.1	Rückgabe Checkergebnis	Jeder Checker-Service muss ein Ergebnis zurückliefern, das die Information enthält, ob der Check „erfolgreich“ oder „nicht erfolgreich“ war.

A-6.2	Rückgabe Begründung	Es muss möglich sein einen Checker-Services zu implementieren, der im Fall eines negativen Checks eine Begründung für das Fehlschlagen des Checks mitliefert.
A-7.1	Kompatibilität mit Planer-Software	Das Checker-Framework soll mit mindestens einer Produktionsplanungssoftware kompatibel sein.
A-8.1	Checker-Services mit Zuständen	Es muss möglich sein einen Checker-Services zu implementieren, der zustandsbehaftet ist.

2.10 Abgelehnte Anforderungen

Neben den hier aufgeführten Anforderungen wurden weitere im Konsortium diskutiert und nicht aufgenommen. So wurde angedacht, dass Checker weitere Funktionalitäten von Planern übernehmen sollten, konkret wie das Ermitteln eines Plans, wie etwas umsetzbar ist, und das Iterieren über Varianten von Szenarien. Mit Blick auf eine saubere Trennung von Funktionalitäten (Separation of Concerns) und die angestrebte modulare Architektur wurde entschieden derartige Funktionalitäten separaten Planern, deren Hauptaugenmerk auf einer performanten Umsetzung dieser liegt, zu überlassen und nicht ins Checker-Framework mit aufzunehmen.

Ebenso wurde die Möglichkeit diskutiert die Informationen, die der Checker benötigt, vom Checker selber einzuholen (indem er die beteiligten Ressourcen nach ihren Daten befragt) statt alles vorher in Erfahrung zu bringen und gesammelt an den Checker zu übergeben. Dies hat jedoch den Nachteil, dass es dann, da der Checker immer nur den aktuellen Zustand der erreichbaren Komponenten abfragen würde, nicht möglich ist auch hypothetische Szenarien, die evtl. in der Zukunft auftreten können oder noch nicht hinzugefügte Komponenten enthalten und für einen Planer relevant sind ebenfalls überprüfen zu lassen. Aus diesem Grund wird die Datensammlung nicht vom Checker durchgeführt, sondern stattdessen der aufrufenden Applikation überlassen.

ID	Titel	Anforderung
A-9.1	Produktionsplanung	Das Checker-Framework muss keine Produktionspläne ermitteln.
A-9.2	Szenarien iterieren	Das Checker-Framework muss keine Iteration über Varianten von Produktionsszenarien beinhalten.
A-9.3	Datensammlung aus Verwaltungsschale	Der Checker-Service muss keine Daten aus VWS abrufen.

3 Architektur

3.1 Kombiniertes Framework für Capability- und Feasibility-Checks

Die Trennung zwischen Capability- und Feasibility-Check muss nicht immer klar sein. Manche Machbarkeiten sind kontextunabhängig und rein symbolisch zu checken, sodass in diesem Fall der Capability- und der Feasibility-Check das Gleiche sind. Aufgrund der engen Verwandtschaft zwischen den beiden Checks ist geplant, beide in einem Framework zu kombinieren und die Capability-Checks als Sonderfall der Feasibility-Checks (solche ohne Kontextinformationen) zu betrachten.

3.2 Micro-Service-Architektur

Zur sauberen Kapselung der Funktionalität und zur einfacheren Integration in variable Tool-Landschaften soll das Checker-Framework auf eine Micro-Service-Architektur aufgebaut werden. Hierbei soll das Framework zwei unterschiedliche Arten von Services ermöglichen:

- Einen sogenannten Broker-Service, bei dem sich Checker an- und abmelden und der eine anfragende Applikation an passende Checker vermittelt
- Die eigentlichen Checker-Services, die Szenarien checken und Auskunft über Capabilities und Feasibilities geben

Der Checker-Broker soll dabei als fertige Software-Komponente, die direkt in eine Verwaltungsschalen-Umgebung deployt werden kann, bereitgestellt werden (Anforderungen A-2 & A-5). Für die Entwicklung von einzelnen Checkern soll ein Toolkit zur Verfügung gestellt werden, das im nächsten Abschnitt beschrieben wird (siehe A-5.2).

Das folgende Schaubild gibt einen Überblick über die Micro-Service-Architektur. Der Checker-Broker soll redundant ausgeführt werden (A-5.7), um hier die notwendige Verfügbarkeit zu garantieren. Sowohl Applikationen als auch Checker-Services können Teil der Software-Komponente einer Verwaltungsschale sein und in einem eigenen Teilmodell beschrieben werden (A-2.4). Ein Checker-Service kann Teile seines Szenarios (die Summe der übergebenen Informationen, die die PPR- (Anforderung A-1) sowie Kontextinformationen enthält, wird im Folgenden als Szenario bezeichnet) überprüfen, indem er hierfür andere passende Checker-Services aufruft (A-5.8).

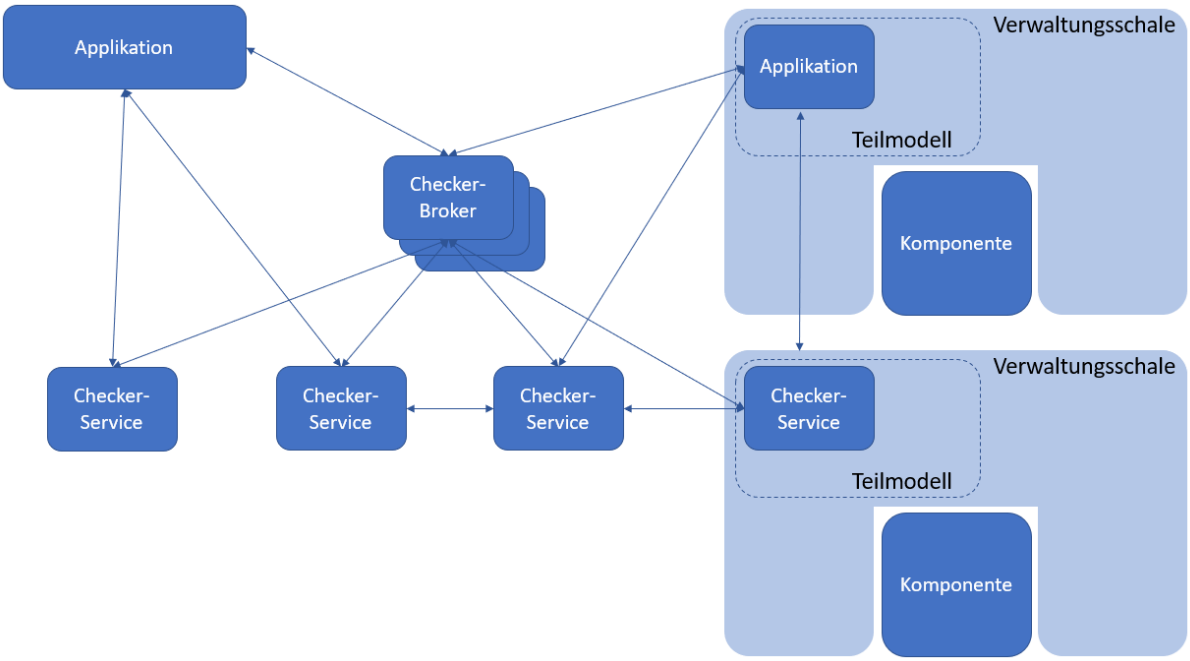

Abbildung 1: Micro-Service-Architektur

Zur Implementierung des Checker-Brokers wurde das Java Framework Spring Boot ausgewählt, das sich im Geschäftsumfeld für die Entwicklung von Micro-Services bewährt hat.

3.3 Kommunikation

Das folgende Schaubild illustriert die Kommunikation zwischen den Komponenten. Beim Hochfahren der Checker-Services registrieren sich diese beim Checker-Broker (A-5.3). Dieser bestätigt die Registrierung und erfragt seinerseits die für ihn relevanten Infos vom Checker, falls er diese nicht bereits kennt. Die Applikation wendet sich mit einer getChecker-Query an den Broker, um eine Liste mit den Adressen von für die Anfrage passenden, aktuell erreichbaren Checkern zu erhalten. Die Applikation sammelt dann alle für den Check notwendigen Informationen zusammen und sendet diese an den von ihr ausgewählten Checker. Der Checker überprüft das Szenario (unter Umständen, indem er Teile des Checks an weitere Checker delegiert (A-5.8)) und sendet das Ergebnis zurück an die Applikation. Wenn der Checker-Service heruntergefahren werden soll, meldet er sich beim Checker-Broker ab (A-5.3) und erhält von diesem eine Bestätigung, dass die Abmeldung erfolgreich war (A-6.1).

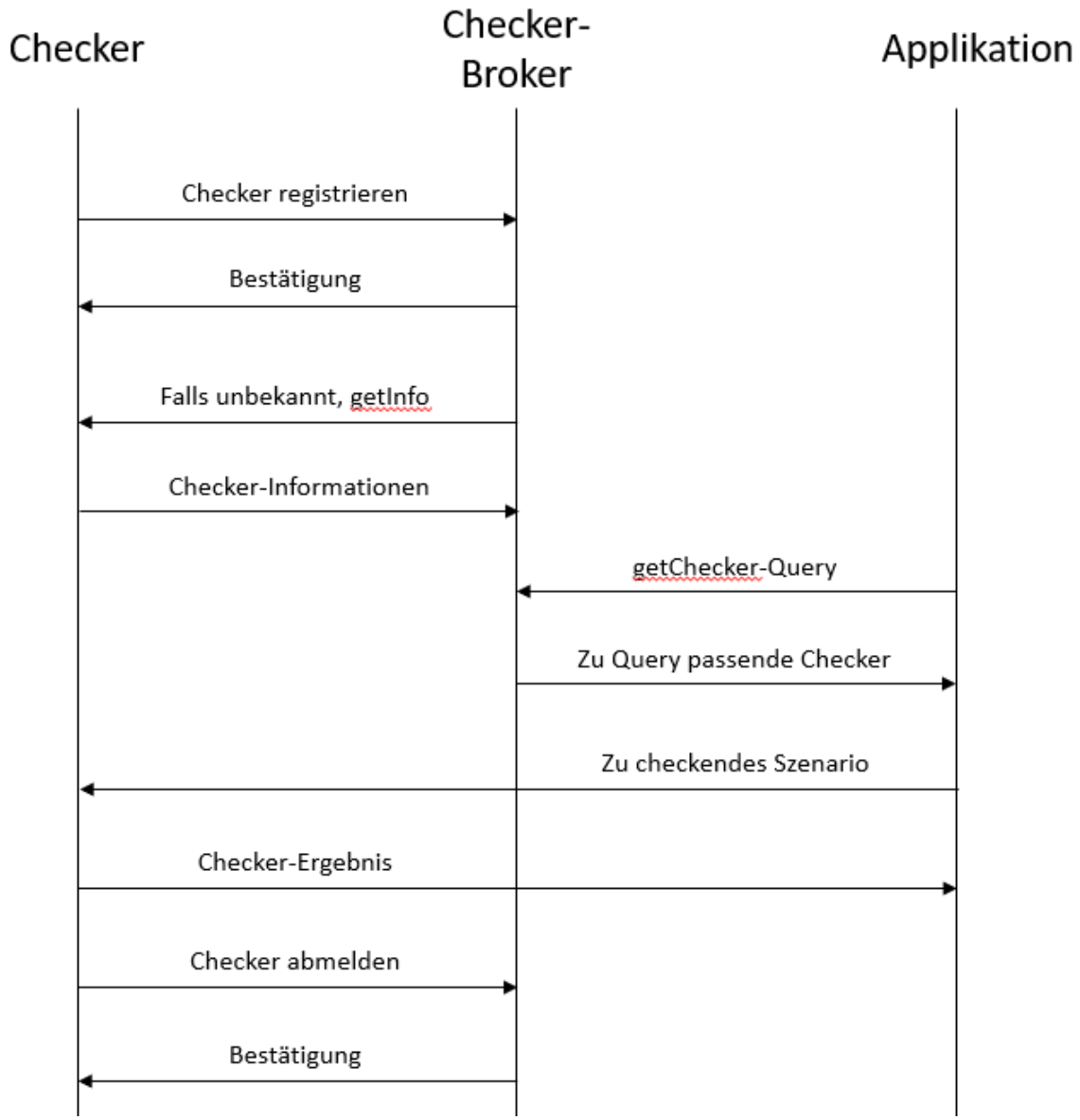

Abbildung 2: Kommunikationsvermittlung zwischen Checker und Applikation

3.4 Checker-Identifikation

Zur Ermittlung von passenden Checkern für ein gegebenes Szenario kann die Applikation eine getChecker-Query an den Checker-Broker schicken. Es ist vorgesehen, dass der Checker-Broker den oder die passenden Checker-Services anhand von drei möglichen Informationen ermittelt:

1. Durch Angabe von Checker-IDs in der getChecker-Query (A-5.4)

2. Durch Angabe von Checker-Fähigkeiten in der getChecker-Query (A-5.5)
3. Durch Angabe des Szenarios in der getChecker-Query (A-5.6)

Im ersten Fall führt der Checker-Broker lediglich einen Look-up in seiner internen Map durch und liefert die entsprechenden Checker-Informationen zurück. Für den zweiten Fall soll es möglich sein, Checker mit Fähigkeiten zu annotieren, die wiederum von einem speziellen internen Capability-Checker, der vom Checker-Broker aufgerufen wird, überprüft werden können. Der dritte Fall entspricht einem tatsächlichen Machbarkeitstest, bei dem die tatsächlichen Daten gegen die erlaubten Schemata der Checker geprüft werden. Entsprechend sollen hier interne Feasibility-Checker verwendet werden. Je nach akzeptiertem Eingabeformat überprüfen diese Checker die konkreten Daten und gehen damit über den rein symbolischen Check des Checkers in Punkt 2 hinaus.

Um diesen Check zu ermöglichen, können Checker ihr Schema über einen entsprechenden Endpunkt bereitstellen. Für Checker, die JSON-Inputs erwarten, kann dies beispielsweise ein JSON-Schema [3] (zur Zeit noch in der Entwurfsphase) sein, für Checker, die RDF-Inputs erwarten, z.B. SHACL [4]. Das Checker-Framework soll, neben dem Checker-Broker und dem Toolkit zum Erstellen von Checker-Services und Clients, auch mindestens zwei interne Checker, die zur Verarbeitung von getChecker-Queries verwendet werden können, anbieten:

- Einen Capability-Checker für Checker-Services, die mit Fähigkeiten annotiert sind
- Einen Feasibility-Checker für Szenarien im RDF-Format

Weitere Feasibility-Checker, wie einen Feasibility-Checker für Szenarien im JSON-Format, sollen dynamisch ergänzbar sein.

4 Toolkit

Zur einfachen Erstellung von mit dem Checker-Framework kompatiblen Checker-Services (A-5.1) soll ein Toolkit zur Verfügung gestellt werden (A-5.2). Das Toolkit soll eine auf dem bereits genannten Java Framework Spring Boot aufbauende Bibliothek sein, mit deren Hilfe man mit möglichst geringem Aufwand eigene Checker und Clients zur Kommunikation mit dem Checker-Framework implementieren kann. Im Einzelnen soll das Toolkit mindestens die folgenden Funktionalitäten enthalten.

4.1 Grundfunktionalitäten

Das Toolkit soll einen

- Webserver,
- Interfaces mit den notwendigen Schnittstellen,
- alle notwendigen Kommunikationsfunktionalitäten um mit Verwaltungsschalen (A-2.1, A-2.2, A-5.9), dem Checker-Broker, Checker-Services und anderen Services zu kommunizieren,
- sowie einen Konfigurationsmechanismus

enthalten, sodass der Nutzer idealerweise lediglich die check-, schema- und info-Methode, die die eigentliche Logik enthalten, implementieren muss. Die Verwaltungsschalenkommunikation soll mit Eclipse BaSyx exemplarisch als Middleware Service umgesetzt werden (A-2.3).

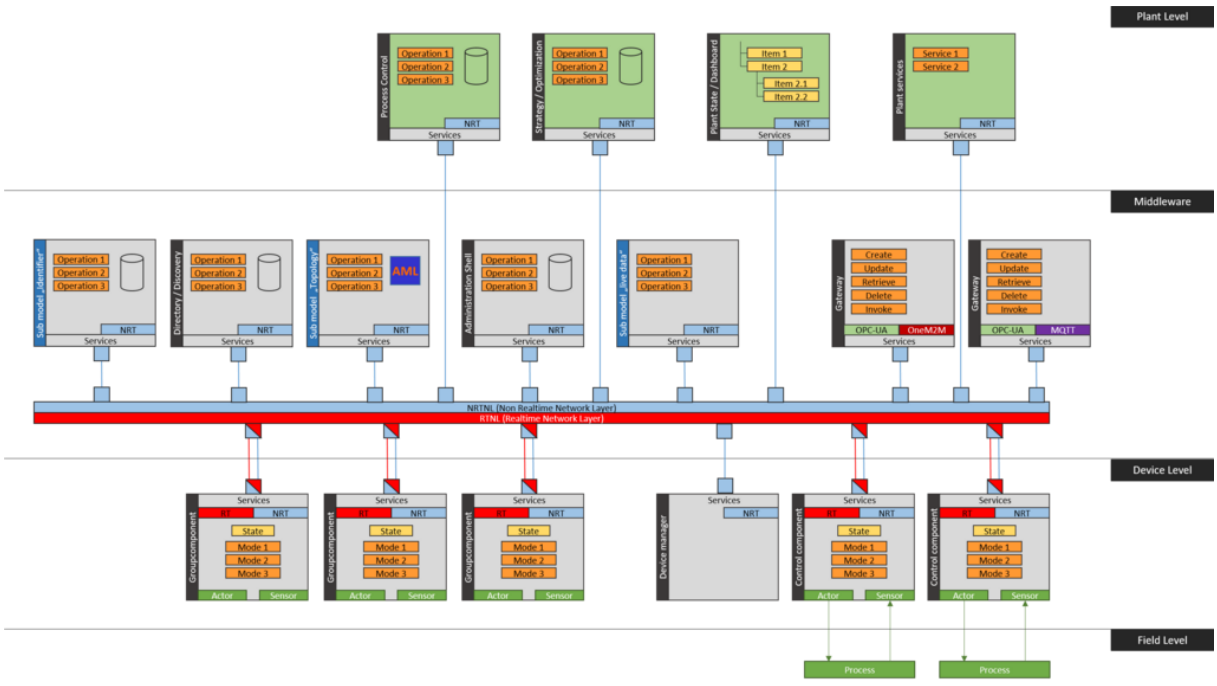

Abbildung 3: Eclipse BaSyx Middleware (Bild von https://wiki.eclipse.org/BaSyx_/_Documentation#BaSyx_SDK_Architecture)

4.2 Teilmodell-Wrapper

Verwaltungsschalen-Teilmodelle können in der Zukunft Informationen enthalten, die direkt für bestimmte Capability- oder Feasibility-Checks verwendet werden können. Zum Beispiel kann die Traglast eines Roboters mit dem Gewicht eines zu transportierenden Produkts verglichen werden, falls die Traglast-Information in der Verwaltungsschale angegeben ist. Um den Aufwand zur Erstellung von Checkern möglichst gering zu halten, soll das Toolkit für bereits in Verwaltungsschalen-Teilmodellen angegebene Eigenschaften, die sich für naheliegende Checks eignen, diese out-of-the-box in Checker überführen können (A-2.4).

4.3 Nebenläufige Anfragen

Wenn ein Checker Teile seiner Anfrage an andere Checker oder Drittanbieter-Software delegiert, soll ihn das Toolkit hierbei unterstützen, indem es neben den notwendigen Kommunikationsmechanismen zum Aufrufen von Service-Endpoints und Verwaltungsschalen auch einen einfachen Mechanismus bereitstellt diese zu parallelisieren, um sie schneller abarbeiten zu können.

4.4 Checker mit Zuständen

Typischerweise werden zustandslose REST-Services in Micro-Service-Architekturen verwendet. Daher wird auch im Capability- und Feasibility-Checker-Framework die Verwendung von zustandslosen Checkern empfohlen. In bestimmten Situationen kann es jedoch nützlich sein, dass ein Checker einen Zustand vorhält (A-8.1).

Das Toolkit soll den Entwickler eines solchen Checkers hierbei unterstützen, indem es einen Mechanismus zur Verwaltung von Sessions und einen einfachen Weg, die Information, dass der Checker zustandsbehaftet ist, über die info-Schnittstelle zu kommunizieren (A-3.2), bereitstellt. Die Sessions können evtl. auch von einer externen Komponente verwaltet werden. Die Applikation, die den Checker-Services aufruft erhält von diesem beim ersten Aufruf einen Identifier für die geöffnet Session, die sie bei zukünftigen Aufrufen mitschicken kann. Der Checker-Service nutzt die mitgeschickte Session-ID um den vorgehaltenen Zustand zuzuordnen.

4.5 Kommunikation mit Checker-Broker

Das Toolkit soll die Methoden zur An- und Abmeldung eines Checker-Services beim Checker-Broker bereitstellen. Darüber hinaus soll das Toolkit sicherstellen, dass jeder Checker stets eine Versionsnummer und einen eindeutigen Identifier (A-5.4) erhält, mit denen er sich beim Broker meldet. Um Kommunikation zu sparen, erfragt der Broker die Informationen vom Service nur, wenn er diese nicht schon kennt. Dieses ist der Fall, wenn entweder der Identifier unbekannt ist oder der Broker nur die Informationen einer veralteten Version kennt. Entsprechend sollte der Entwickler des Checkers darauf hingewiesen werden, die Versionsnummer des Checkers bei jedem Update entsprechend zu erhöhen.

5 Schnittstellen Checker-Broker

Da die Checker semantische Tools sind, die auf Grundlage von logischer Inferenz und Simulation Aussagen über die Machbarkeit von Prozessen treffen, ist es naheliegend die Rückgabe der Ergebnisse ebenfalls in einem semantischen Standard-Format bereitzustellen. Entsprechend wird das Checker-Framework seine Rückgabe in verschiedenen RDF-Varianten wie JSON-LD und Turtle bereitstellen. JSON-LD wird hier als bevorzugte Sprache gewählt, da JSON auch außerhalb der Semantik-Community weithin akzeptiert und verwendet wird. Über einen optionalen Parameter bei der Eingabe kann eines der alternativen Formate angefordert werden.

Der Checker-Broker soll für die Checker-Services Schnittstellen zum An- und Abmelden und für die aufrufende Applikation eine Schnittstelle zur Abfrage verfügbarer Checker bereitstellen. Für die einzelnen Schnittstellen sind im Folgenden Beispielaufrufe und -antworten aufgeführt.

5.1 Anmelde-Methode

Eingabe

Beispiel:

{

„@context“ : „http://basys42.de/rdf/“,

„@id“ : „ http://basys42.de/rdf/Checker/7890057a-35f9-4b86-aeb4-89a9781a540d“,

“@type”: „ProbabilisticChecker“,

“version”: „1.0.0“

}

Ausgabe

Beispiel:

{

„success“ : true

}

5.2 Abmelde-Methode

Eingabe

Beispiel:

{

„@context“ : „http://basys42.de/rdf/“,

„@id“ : „ http://basys42.de/rdf/Checker/7890057a-35f9-4b86-aeb4-89a9781a540d“,

“@type”: „ProbabilisticChecker“
}

Ausgabe

Beispiel:

{
„success“ : true
}

5.3 GetChecker-Methode

Eingabe

Beispiel ID:

{
„@context“ : „http://basys42.de/rdf/“,
„@id“ : „http://basys42.de/rdf/getCheckerById/7890057a-35f9-4b86-aeb4-89a9781a540d“,
“@type”: „GetCheckerByIdQuery“,
“ids”: [„http://basys42.de/rdf/Checker/7890057a-35f9-4b86-aeb4-89a9781a540d“]
}

Beispiel Fähigkeiten:

{
„@context“ : „http://basys42.de/rdf/“,
„@id“ : „http://basys42.de/rdf/getCheckerByCapability/7890057a-35f9-4b86“,
“@type”: „GetCheckerByCapabilityQuery“,
“capabilities”: [„AcceptsJson“, „Real-time“, „ChecksPayload“]
}

Beispiel Szenario:

{
„@context“ : „http://basys42.de/rdf/“,
„@id“ : „http://basys42.de/rdf/Process/0567057a-35f9-4b86-aeb4-89a9781a540d“,
„@type“ : „http://basys42.de/rdf/Process“,
„products“ : [{
„@id“ : „http://basys42.de/rdf/Product/4b67057a-35f9-4b86-aeb4“,
„@type“ : „http://basys42.de/rdf/Product“,

„weight“ : 56
}],
„resources“ : [{
“@id”: “http://basys42.de/rdf/Resource/3247057a-35f9-4b86-aeb4“,
„@type“ : „http://basys42.de/rdf/Resource“,
“payload”: 100
}],
“steps”: [{
“@id” : “http://basys42.de/rdf/Skill/LiftProduct”,
„@type“ : „http://basys42.de/rdf/Step“,
“name” : “Lift product”,
“resource”: {
“@id”: “http://basys42.de/rdf/Resource/3247057a-35f9-4b86-aeb4“,
„@type“ : „http://basys42.de/rdf/Resource“
}
“product”: {
„@id“ : „http://basys42.de/rdf/Product/4b67057a-35f9-4b86-aeb4“,
„@type“ : „http://basys42.de/rdf/Product“
}
}]
}

Ausgabe

Beispiel:

{
„@context“ : „http://basys42.de/rdf/“,
„@id“ : „http://basys42.de/rdf/CheckerList/4567057a-35f9-4b86-aeb4-89a9781a540d“,
“@type”: „CheckerList“,
“availableChecker” : [
{
„@id“ : „http://basys42.de/rdf/Checker/7890057a-35f9-4b86-aeb4-
89a9781a540d“,
“@type”: „ProbabilisticChecker“,
“address” : “http://localhost:9090”
},

{
„@id“ : „http://basys42.de/rdf/Checker/7890157a-35f9-4b86-aeb4-
89a9781a540e“,
“@type”: „ProbabilisticChecker“,
“address” : “http://localhost:9091”
}
]
}

6 Schnittstellen Checker

Die Schnittstellen des Checkers sind in Deliverable 2.11 beschrieben. Sie umfassen eine info-Methode (Anforderung A-3), die eine Selbstauskunft des Services bereitstellt, eine check-Methode, von der der eigentliche Check durchgeführt wird, sowie eine optionale schema-Methode bereitstellen, die angibt, in welchem Format Szenarienbeschreibungen von ihm verarbeitet werden können.

Literatur

[1] R. Drath, A. Luder, J. Peschke und L. Hundt, „AutomationML - the glue for seamless automation engineering,“ in IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2008.

[2] Shannon, C. E.. Communication in the presence of noise. Proceedings of the IRE, 37(1), 10-21, 1949.

[3] JSON Schema Webseite, https://json-schema.org/, Stand: 14.09.2020.

[4] Shapes Constraint Language (SHACL), W3C Recommendation 20 July 2017, https://www.w3.org/TR/shacl/, Stand: 14.07.2020.