Systematische Betrachtung nichtfunktionaler Anforderungen beim Entwurf von I4.0 Service-Infrastrukturen

Prof. Dr. rer. nat. Dipl.-Ing. **R. Denzer**, EIG, htwsaar, Saarbrücken;
Dipl.-Inform. **G. Sutschet**, Fraunhofer IOSB, Karlsruhe

Kurzfassung

Cyber-physische Systeme (CPS) besitzen hohe nicht-funktionale Anforderungen (NFA) in Bezug auf einen sicheren, fehlertoleranten, resilienten, effizienten, selbst-organisierenden und störungsfreien Betrieb. NFA sind daher notwendigerweise in der detaillierten Ausgestaltung und Spezifikation einer Service-Infrastruktur für CPS zu berücksichtigen – so auch auf dem Weg zu einem I4.0 Referenzmodell bzw. einer I4.0-Referenzarchitektur. Wie das sinnvoll geschehen kann, war bislang wenig Gegenstand der Forschung und Entwicklung. Klassische Referenzmodelle für Service-Architekturen aus der Informatik betrachten obige NFA bislang nicht. Ein erster Ansatz wird in der Industrial Internet Reference Architecture (IIRA) lediglich in Bezug auf die Informationssicherheit (security) gemacht. NFA werden überwiegend auf so abstrahiertem Niveau beschrieben, dass sie für die konkrete Spezifikation einer Service-Infrastruktur für CPS zu abstrakt sind, um wirklich nützlich zu sein. Darüber hinaus reicht es nicht aus NFA zu postulieren, es ist vielmehr notwendig einen Brückenschlag zu antizipierten Lösungen herzustellen, welche in die Spezifikation eines Referenzmodells eingehen. Der Beitrag beschreibt einen Vorschlag aus der Forschung, wie NFA und Lösungsoptionen für NFA systematisch in die Entwicklung von Referenzmodellen einbezogen werden können, und zwar so konkret, dass sie direkt das Design der Service-Infrastruktur und das Engineering konkreter Systeme beeinflussen und testbar machen.

1. Nicht-funktionale Anforderungen an I4.0 Service-Infrastrukturen

Von Beginn an fordern die Umsetzungsempfehlungen der Plattform Industrie 4.0 [1] die Betrachtung von Referenzmodellen als Lösungsansatz für die Service-Infrastruktur von I4.0-Systemen. Dieser Artikel beleuchtet wie nicht-funktionale Anforderungen (NFA) in die Spezifikation eines Referenzmodells bzw. einer Referenzarchitektur eingehen können.

Wir beobachten, dass sowohl im Umfeld von I4.0 [2] als auch im Umfeld der Industrial Internet Reference Architecture (IIRA) [3] sowie ganz generell im Umfeld des Internet of Things (IoT) [4] NFA formuliert werden („safety“, „security“, „resilience“, usw.), dass die Begrifflichkeiten aber zu abstrakt gefasst sind, als dass sie konkret in das Design einer Service-

Infrastruktur eingehen könnten. Das gleiche gilt für die I4.0-Komponente, die bisher nicht formal spezifiziert ist. Es gibt lediglich eine Übereinkunft in Form einer „Prosa-Meinung“ über einige ihrer grundlegenden Eigenschaften. Das Schachteln ([2], Seite 22 „nestability“) einer I4.0-Komponente ist z.B. nicht klar definiert: was bedeutet das Schachteln genau für die Verwaltungsschalen einer I4.0-Komponente und den Zugriff auf geschachtelte I4.0-Komponenten?

Bei der Abgrenzung zwischen Nicht-Funktionaler Anforderung (NFA) und Funktionaler Anforderung (FA) ist offensichtlich, dass FA das „was“ definieren und NFA das „wie“, auch wenn es schwer fällt eine klare Definition für beide in der Literatur von CPS zu finden. So wird z.B. der Begriff der NFA zwar in den IoT-A Dokumenten verwendet, ist aber nicht im zugehörigen Glossar definiert. Da Anforderungen aber generell jeden Architekturprozess treiben, ist es eine unschöne Situation, wenn man nicht konkret angeben kann, an welcher Stelle des Prozesses welche Anforderungen vorwiegend berücksichtigt werden sollten. Zu Beginn müsste man daher erst einmal fragen: Was ist der Unterschied zwischen einer NFA und einer FA in Bezug auf den Architekturprozess von Industrie 4.0 Systemen?

Auf der jetzigen abstrakten Ebene kann man die NFA lediglich als Ziele oder Handlungsempfehlungen begreifen. Die abstrakte Art der Beschreibung ist nicht dazu geeignet eine I4.0-Serviceinfrastruktur zu spezifizieren, welche systematisch NFA berücksichtigt, sodass die Infrastruktur auch gegen diese geprüft (getestet) werden kann.

2. Anforderungen und Referenzmodelle

Um sich dem Unterschied zwischen NFA und FA zu nähern, ist es sinnvoll, ein Beispiel eines konkreten SOA-Referenzmodells für eine Domäne zu betrachten und zu beleuchten, wie Anforderungen in den Architekturprozess eingehen. Wir verwenden hierzu ein Referenzmodell aus dem Bereich des Risk Management, das RM-OA (Reference Model of the ORCHESTRA Architecture [5]).

Zunächst ist festzuhalten, dass ein Unterschied darin besteht, ob man ein *System* oder eine SOA-Infrastruktur für eine *Systemklasse* über ein Referenzmodell spezifiziert. Letzteres wurde im ORCHESTRA-Projekt für Risikomanagement-Systeme gemacht [6]. Dabei wurden zunächst typische Anforderungen unterschiedlicher Anwendungen erhoben und harmonisiert, wobei selbstverständlich die benötigten *Funktionen* (die Elemente der Lösung des Problemraums) im Vordergrund standen. Aus diesen „funktionalen“ Anforderungen [7] wurde eine Service-Infrastruktur abgeleitet, welche diese Anforderungen befriedigt (Bild 1), was in der Regel ein iterativer Prozess ist [8]. Diese Service-Infrastruktur besteht letztlich abstrakt

aus der *Menge der Dienste* und den *dazugehörigen Informationsmodellen*, und wurde in diesem Falle gemäß RM-ODP [9] modelliert.

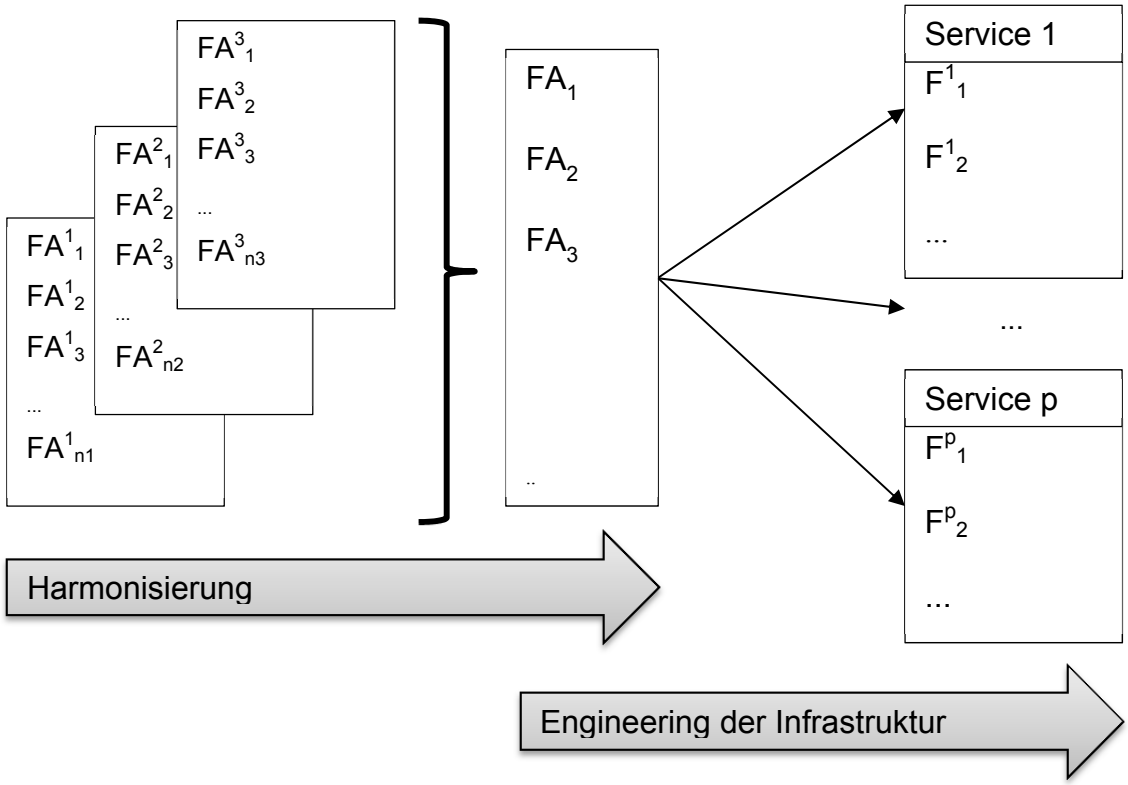

Bild 1: Funktionale Anforderungen beim Engineering einer SOA-Infrastruktur

Eine Eigenschaft funktionaler Anforderungen ist, dass man für jede von ihnen überprüfen kann ob sie erfüllt ist. Eine SOA ist dann vollständig, wenn die Menge aller Funktionen der Dienste

$$\{ F^1_1, F^1_2, ... F^1_{m1}, F^2_1, F^2_2, ... F^2_{m2}, ... F^p_1, F^p_2, ... F^p_{mp} \}$$

die Menge aller harmonisierter Anforderungen

$$\{ FA_1 , FA_2, ..., FA_n \}$$

abdecken, oder anders ausgedrückt: Man kann an den Dienst-Schnittstellen direkt ablesen, ob sie zusammen genommen in der Lage sind die benötigten Funktionen bereitzustellen. Ob eine funktionale Anforderung also erfüllt ist, kann man klar entscheiden, und es gibt ein Mapping

$$\{ FA_1 , FA_2, ..., FA_n \} \rightarrow \{ F^1_1, F^1_2, ... F^1_{m1}, F^2_1, F^2_2, ... F^2_{m2}, ... F^p_1, F^p_2, ... F^p_{mp} \}.$$

Im ORCHESTRA-Projekt wurde für den Bereich des Risikomanagements auch untersucht, was im Umfeld von CPS bzw. IoT heute gemeinhin unter NFA verstanden wird. Im Risikomanagement spielen Echtzeit- und Robustheitseigenschaften für den Großteil der System-

klasse keine Rolle. Hingegen sind Langlebigkeit, Autonomie und dynamische Änderungen der Teilsysteme wichtig. Betrachtet man die zugehörigen Dokumente (Development Dimensions [10] und System Requirements [11]) und legt das Referenzmodell [5] und die Service-Spezifikationen [12] daneben, so erkennt man, dass manche der NFA (dort *key system requirements* und *architectural principles* genannt) sich auf das Referenzmodell als solches auswirken, andere auf Spezifikationsverfahren, andere auf konkrete Dienste. Eine NFA wirkt in der Regel auf viele Stellen der Gesamtarchitektur, auch wenn dies nicht überall im Sinne von Architekturentscheidungen gut dokumentiert ist. Daraus könnte man ableiten, dass man vielleicht NFA von FA dadurch unterscheiden könnte, dass FA auf einzelne Funktionen wirken, während NFA sich im Gesamtsystem-Verhalten ausdrücken. Bei genauer Betrachtung verschwimmt diese Grenze aber auch sehr schnell. Letztlich zerfallen Anforderungen in zwei Kategorien: Solche, die man durch „Abzählen" und Einzelfunktionstests testen kann und solche, die auf alle Elemente der Architektur wirken. Bei letzteren hat man oft ein Bündel an Handlungsoptionen und ein quantitativer Test ist schwierig.

Traditionelle Verfahren zur Definition einer SOA wie RM-ODP oder ISO 42010 wie in der IIRA sind unserer Meinung nach unvollständig. Es wird einer Erweiterung der Spezifikationsmethodik bedürfen, in der definiert ist, an welcher Stelle welche Anforderungen eingehen und in welcher *konkreten* Form. Das heißt auch, dass man zu Beginn dieses Prozesses klären muss, welches die *Grund-Elemente der Architektur* (GEA_i) sind, und wie und wann der Vektor der FA und NFA im Architekturprozess auf diese abgebildet wird:

$$\{ FA_1 , FA_2, ..., FA_n \}, \{ NFA_1 , NFA_2, ..., NFA_m \} \rightarrow \{ GEA_1, GEA_2, ... , GEA_q \}$$

4. Notwendige Konkretisierung

Um auf dem Weg zu einem Referenzmodell voran zu kommen, ist es notwendig, die bislang auf abstraktem Niveau geführte Diskussion *auf allen Ebenen* (nicht nur für NFA) zu konkretisieren und zwar so weit, dass diese Abbildung auf *konkrete Maßnahmen* und *Architekturentscheidungen* erfolgen kann. Diese führen zu Spezifikationen, die tatsächlich interoperabel implementiert werden können, nachdem sie auf eine oder mehrere konkrete Middleware-Plattformen gemappt worden sind. In diesem Zusammenhang fällt auf, dass auch die I4.0-Komponente noch recht abstrakt gefasst ist (IIRA kennt keinen ähnlichen Begriff und definiert stattdessen noch abstraktere „Domänen"). Die bisherige Beschreibung der I4.0-Komponente erlaubt Interpretationsmöglichkeiten und ihre Eigenschaften sind auf abstrakt hohem Niveau definiert. Das IoT-A [13] kennt zwar ein analoges Konzept, nämlich das der *Device*, sagt aber wenig über dessen innere Funktionsweise aus und gibt im UML-Modell

Kardinalitäten von *Device*, *PhysicalEntity* und *VirtualEntity* an (alles Selbstreferenzen), welche zu viele Interpretationsmöglichkeiten lassen.
UML-Modelle alleine sind nach unserer festen Überzeugung nicht ausreichend um die I4.0-Komponente zu definieren. Wir schlagen vor, ein mentales Modell für die I4.0-Komponente zu definieren (ähnlich dem im Projekt SANY [14,15] für Sensoren und Sensor-Plattformen), welches im Kern der Architektur angesiedelt ist und welches die Kommunikation zwischen Informatikern und Anwendern erleichtert. Im Gegensatz zu einem reinen UML-Informations-Modell berücksichtigt dies auch funktionale Aspekte im Inneren der I4.0-Komponente. Diese Darstellungsweise ist Ingenieuren sehr viel näher.

5. Grundidee und Vorgehensweise

In diesem Kapitel stellen wir die Grundidee der vorliegenden Arbeit vor. Die Methodik besteht aus der Kombination der folgenden fünf Elemente:

- einem **Mentalen Modell** einer I4.0-Komponente (in den Diagrammen CPC, Cyber Physical Component, genannt),
- einer **Taxonomie Nicht-Funktionaler Anforderungen**, welche detailliert genug ist eine Abbildung der NFA auf ein Portfolio möglicher Lösungen zu machen,
- einem **Mapping** der NFA und ihrer möglichen Lösungen auf alle Grundelemente der Architektur einer I4.0-Serviceinfrastruktur (was wiederum eine klare Definition erfordert, welches die „Grundelemente der Architektur“ sind),
- **Service Interaction Patterns** als Blaupausen für die wichtigsten generischen Abläufe, welche es möglich machen die Lösungen gegen die NFA zu testen, sowohl theoretisch als auch in Testbeds, sowie
- **Service Domain Patterns** als Blaupausen, die typische physische Konfigurationen von I4.0-Systemen in der Produktion reflektieren und die erneut testbar sind.

In der Folge zeigen wir beispielhaft wie diese Methodik verwendet werden kann. Was wir hier erläutern ist allerdings keine fertige Lösung. Wir behaupten nicht, dass das verwendete mentale Modell „das Richtige“ ist; die Taxonomie ist noch rudimentär und ihre Definition erweist sich als schwierig; das Mapping ist beispielhaft illustriert; die Service Patterns sind einzelne Beispiele aus der Praxis und die Domain Patterns aus Platzgründen nur angerissen. Darüber hinaus sind alle Elemente in eine Vorgehensweise einzubetten. Auf dem Weg zu einem I4.0-Referenzmodell wird noch beträchtlich in Forschung, Entwicklung und Konsensbildung zu investieren sein.
In unserem vereinfachten Beispiel gehen wir dazu von lediglich drei Grund-Elementen der Architektur aus (in der Realität kommen noch einige hinzu):

GEA_1 = CPC	Cyber Physical Component
GEA_2 = CPIS	Cyber Physical Infrastructure Service
GEA_3 = PE	PhysicalEntity

5.1 Mentales Modell

Bild 2 zeigt oben ein abgespecktes, zunächst rein funktionales mentales Modell einer CPC (I4.0-Komponente). Dies ist ein ausgesprochen einfaches Modell, welches lediglich der Tatsache Rechnung trägt, dass CPC Sensoren und Aktoren beinhalten können, welche mit dem physischen System interagieren (mögliche Regelkreise innerhalb der Komponente werden hier nicht betrachtet).

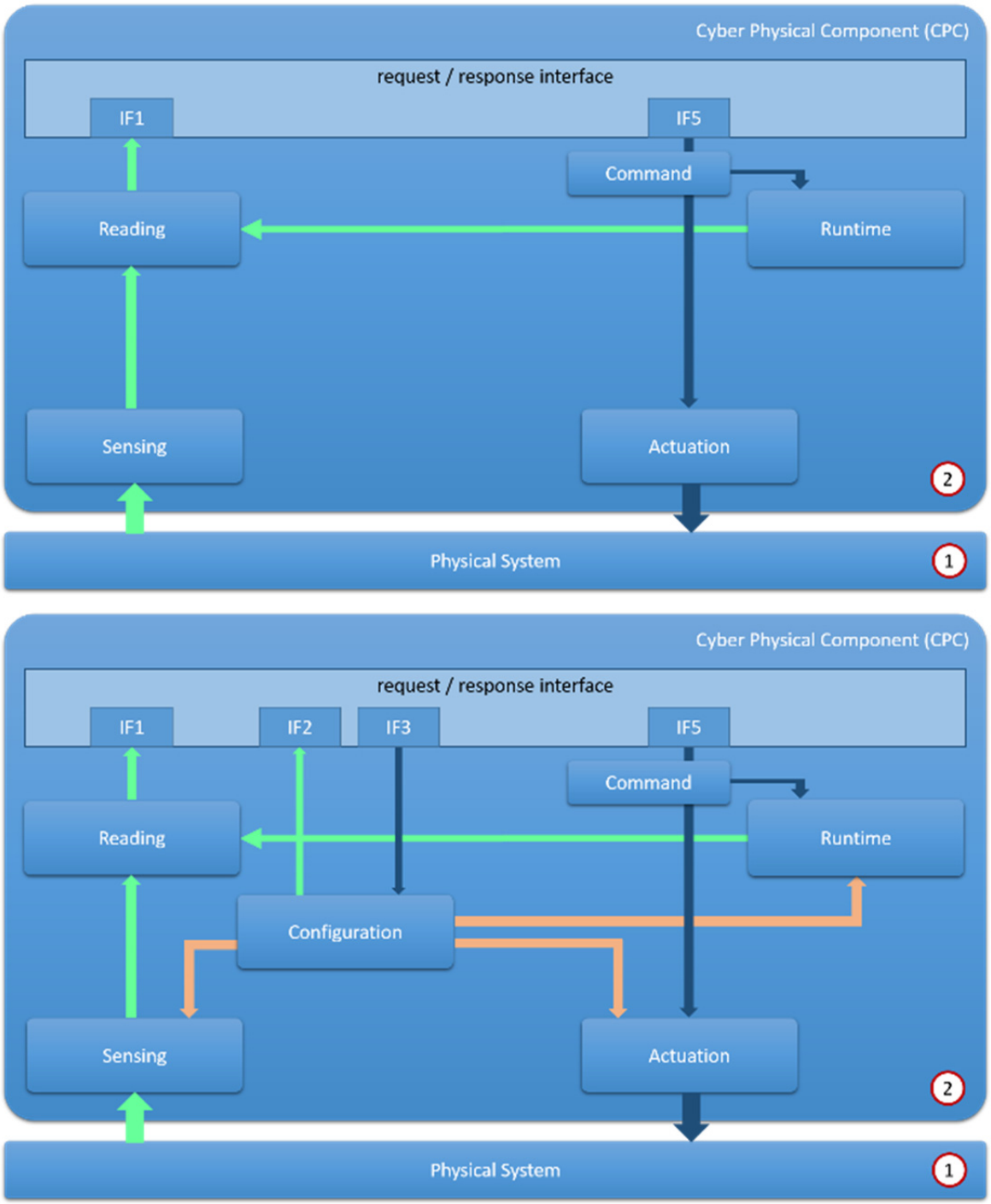

Bild 2: Erste Elemente des mentalen Modells einer CPC (I4.0-Komponente)

Da es sich um eine CPC handelt, inkludiert dieses Modell allerdings auch das eingebettete elektronische System, die *Runtime*, denn diese ist für die Funktionsfähigkeit der CPC genauso wichtig, wie das eigentliche physische System. Sie kann darüber hinaus Beobachtungen produzieren und kann durch Kommandos beeinflußt werden.

Bild 2 unten zeigt wie die Berücksichtigung einer NFA eine Auswirkung auf dieses eine Architekturelement GEA_1 hat. Betrachten wir zunächst die NFA *Adaptability* / *Adaptivity* (eine brauchbare Definition in der vorhandenen CPS-Literatur zu finden ist nicht ganz einfach). Eine Definition findet sich in [16], basierend auf früheren Arbeiten, welche diese als *Flexibility* und *Changeability* einführen, und ohne im Detail auf die Begrifflichkeiten einzugehen wird klar, dass eine CPC dafür einen Konfigurationsmechanismus besitzen muss, was noch immer eine Trivialität ist. Die Konfiguration kann sich auf *Sensing*, *Actuation* und *Runtime* auswirken. Im Umfeld von I4.0 ist auch viel von *Auto Configuration* („plug & produce“, „plug & work“) die Rede und es ist klar, dass der Konfigurationsmechanismus diese NFA berücksichtigen muss, sobald Autokonfiguration gefragt sein sollte.

Betrachten wir, erneut ohne ins Detail zu gehen, als weitere NFA den Bereich der *Safety*. Diese hat unter anderem auch immer mit dem Management von *Faults* zu tun, und ein Teil der Lösung ist die Definition eines Fehlerzustands (oder allgemein eines Zustands *State*, so wie auch in RAMI4.0 gefordert). Wir gelangen damit zu dem mentalen Modell welches wir dem Rest der Diskussion zugrunde legen (Bild 3).

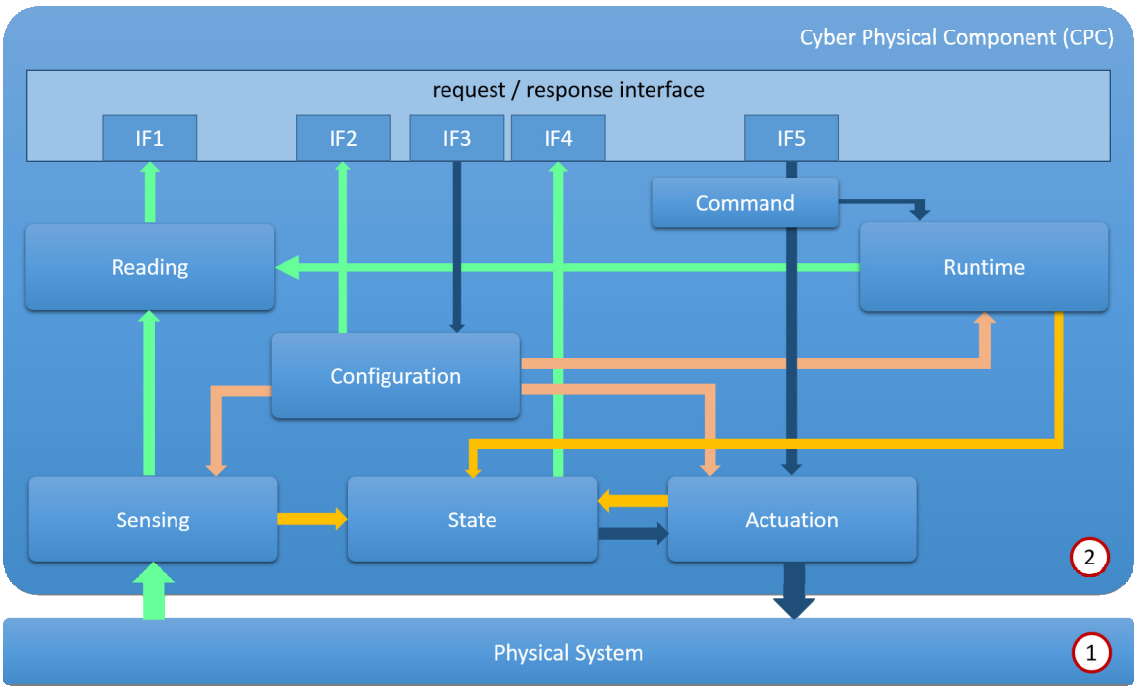

Bild 3: Beispiel für das mentale Modell einer CPC (I4.0-Komponente); der durch (1) bezeichnete Block findet sich im IoT-Modell als *PhysicalEntity*, (2) als *Device*

Diesem Modell kann man alle internen und externen Interfaces entnehmen, und man kann den einzelnen Informationsflüssen Bedeutungen zuweisen. Man kann auch einzelne Elemente sinnvoll standardisieren, so zum Beispiel den *State*, den man durch ein formales Modell einer Zustandsmaschine informationstechnisch spezifizieren kann. Wir sehen an diesen Beispielen wie sich unterschiedliche NFA auf die CPC, also auf GEA_1 auswirken.

5.2 Taxonomie und Mapping

In Tabelle 1 zeigen wir beispielhaft das Grundprinzip: Die NFA-Kategorien werden soweit detailliert heruntergebrochen, dass man ihnen konkrete Lösungsmechanismen zuordnen kann, die durch eines oder mehrere GEA umgesetzt werden. Die ideale Taxonomie würde dabei eine *klare, verständliche, redundanz- und widerspruchsfreie Definition und Kategorisierung* aller im CPS-Umfeld relevanten NFA beinhalten. Jedes Blatt dieser Hierarchie wäre so konkret, dass es mit einer oder mehreren Lösungen verbunden werden kann.

Wir möchten einige Beispiele der Tabelle detailliert erläutern. Sobald man z.B. die NFA *Adaptability* ausreichend konkretisiert, sieht man sehr schnell den Unterschied zwischen *Integration* und *Automatic Integration* aus plug&work, nämlich dass man z.B. einen Mechanismus für *Automatic Detection* benötigt, was eines dafür geeigneten **Dienstes** bedarf, also eines Dienstes, der bei ausschließlicher Betrachtung der FA nie benötigt würde. Während man im ersten Fall (der Registrierung einer Komponente) auf eine *Self Description* der Komponente theoretisch verzichten könnte, ist diese *Self Description* absolut notwendig für den Fall *Automatic Integration*. Man kann auch erkennen, dass bei dieser Betrachtung notwendige Informationselemente abfallen (in diesem Fall die *Beschreibung* der Komponente, also das Manifest im Sinne von RAMI4.0), welche sich im Informationsmodell des Referenzmodells niederschlagen müssen. Aus Gründen der Übersichtlichkeit haben wir darauf verzichtet diese eigentlich gedanklich vorhandene Spalte in der Tabelle darzustellen und stellen das verkürzt dar („Involves also Information model“).

Man erkennt auch, dass manche Definition einer NFA hier vollkommen sinnlos ist. *Flexibility* als “*adaptability already implemented in a production system*” hat als NFA keine Bedeutung, weil der über allem stehende Gedanke von I4.0-Systemen ist, dass das Design flexibel ist. Das ist also ein Axiom. Alle verwendeten Definitionen sind daher sehr kritisch zu hinterfragen.

Tabelle 1: Beispiel für das Mapping von NFA auf Architekturelemente

Taxonomy of non-functional requirements	Solutions	Performed by / involves
Adaptability / Adaptivity		
Flexibility: “adaptability already implemented in a production system” [16], ref #9	[1)]	
Changeability: “… after additional engineering and installation effort” [16], ref #9		
Integration of a CPC (“of a new controllable entity”) [16] into the production system	Registration	CPC, CPIS
	Selection (“comparison of (…) tasks with (…) capability”)	CPS
	Initial configuration	CPC, CPIS
	Registration in safety framework (fault management)	CPC, CPIS, involves also Safety
	Start of component	CPC
	Fault management	CPC, CPIS involves also Safety
Modification of an existing CPC	…	…
…		
Extraction of an existing CPC, (better term may be removal)	…	…
…		
Replacement (of one CPC by another one)	…	…
…		
Plug-and-work (“automatic changeability”)		
Automatic integration of a CPC	Automatic detection of a new I4C	CPIS
	Automatic registration	CPC, CPIS, PE involves also Safety and Information model
	Self description	CPC, PE, CPIS Involves also Information model
…		
… more non-functional requirement categories		
Safety		
…		
Fault management capability	Fault detection	CPC, CPIS, Involves also Information model
	Component reconfiguration	CPIS
	Component shutdown	CPC
	Component restart	CPC
	Alerting of subscribers	CPIS
	Alerting of domain	CPIS
…		

[1] Definitionen zu *Adaptability* sind [16] entnommen, der sich auf frühere Artikel bezieht.

5.3 Service Interaction Patterns

Aus der oben geschilderten Vorgehensweise werden neben den Eigenschaften der CPC **generische Dienste** abgeleitet, wobei generisch bedeutet: „für den Bereich I4.0 allgemeingültige, anwendungsunabhängige Dienste“. Diese nennen wir CPIS (Cyber Physical Infrastructure Services). Es ist klar, dass es sich hierbei um Authorisierung, Security, Registrierung u.s.w. handelt, also um Dienste die selbst keine Produktionsfunktion sondern eine rein dienende Funktion haben. Diese Dienste könnte man für die gesamte CPS Community implementieren, analog zu Internetdiensten.

Betrachten wir erneut den Fall „Integration of a CPC“ aus obiger Tabelle so wird klar, dass Managementprozesse im CPS in der Regel nicht aus einzelnen Dienstaufrufen bestehen, sondern aus ganzen Abläufen (also je nach Terminologie *„business processes“* oder *„services chains“*). Für diese Standard-Managementabläufe kann das Referenzmodell generische Muster bereitstellen, die wir *Service Interaction Patterns* nennen. Das Wesen dieser Muster ist nun, dass sie sich in vielen Fällen nicht alleine auf eine NFA beziehen, sondern *gleichzeitig mehrere NFA adressieren können*. Auch hierzu geben wir ein Beispiel. Nehmen wir an aus einer der NFA aus dem Bereich *Safety* wurde ein CPIS namens *FaultMan* definiert, dessen Aufgabe es ist, Fehler in einer Domäne (z.B. einer Produktionszelle) zu behandeln. Dieser habe bei einer CPC über ein Subscriber-Muster Fehler subskribiert, welche die CPC meldet (Bild 4 links). Man sieht augenblicklich, dass das Muster eine Auswirkung auf die CPC hat: Sie muss die Subskription bereitstellen. Nun werde (wie auch immer) dieser CPC eine Subkomponente hinzugefügt (NFA *Composability*, Bild 4 rechts). Dann kann man ein (oder einige wenige) Standardmuster angeben, wie die Subkomponente in der Domäne registriert und diese gleichzeitig automatisch im Fehlermanagement eingefügt wird. Erneut sieht man direkt die Auswirkungen auf die Funktionalität der CPC und was die beteiligten CPIS anbieten müssen.

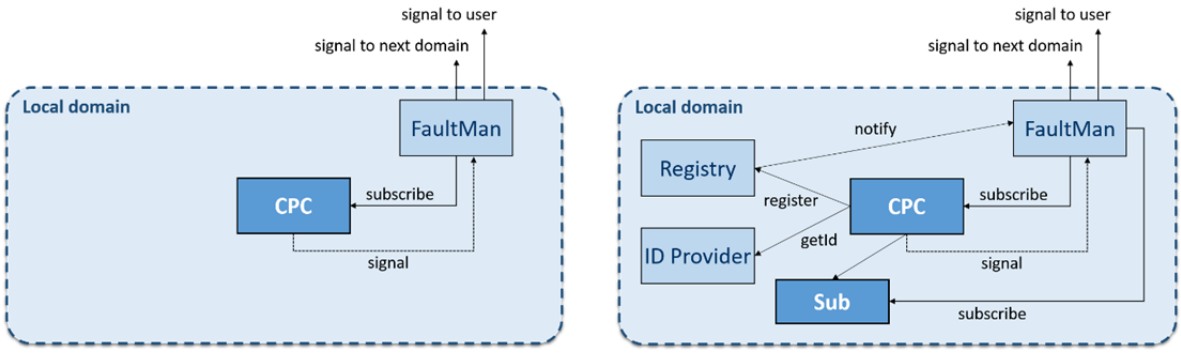

Bild 4: Pattern welches die NFA *Automatic Integration* (bzw. allgemein *Autoconfiguration*) und *Alerting of Subscribers* adressiert, sowie eine weitere: *Discoverability*

Der Fortschritt gegenüber bisherigen SOA-Spezifikationsverfahren ist, dass man diese Standardmuster **testen** kann (durch Simulation oder in Testbeds) indem man Ausfälle erzeugt, Cyber-Attacken oder menschliches Versagen simuliert.

5.4 Service Domain Patterns

Aus weiteren Überlegungen geht hervor, dass die Muster je nach Use Case oder physischer Konfiguration unterschiedlich ausgeprägt sein können. Die IIRA kennt dieses Konzept im *Implementation Viewpoint* unter dem Begriff *Architecture Patterns*. Wir halten sowohl den Viewpoint unglücklich gewählt („concerned with the technical representation of an Industrial Internet System“) als auch den Begriff, da er zu allgemein ist. Da es sich im I4.0-Umfeld um Teilbereiche der Produktion handelt (welche durch ein Service-Netzwerk abgegrenzt sind), schlagen wir daher den Begriff *Service Domain Pattern* vor. Diese typischen Domänen könnte man an ISO/IEC 62264-1 anlehnen und vielleicht sogar versuchen Muster so zu kategorisieren, dass sie den *Funktional Hierarchy Levels* (Abschnitt 5.2) zugeordnet werden können. Die Domain Patterns würden z.B. Konfigurationen bereitstellen für das Kaskadieren von *Faults* oder die systemweite Abfrage von *States*. Sie wären erneut **testbar**. Aus Platzgründen können wir hier kein Beispiel eines solchen Patterns darstellen.

7. Diskussion

Wir sind der festen Überzeugung, dass ein Referenzmodell für CPS alle fünf vorgestellten Elemente (sowie die dazu gehörigen Informationsmodelle) benötigt, wenn es in der Lage sein soll die NFA von CPS in der Produktion abzudecken. Aus unseren Überlegungen geht hervor

> *dass es eines neuen Typs SOA-Referenzmodell bedarf, um die Anforderungen aus dem Bereich der Cyber Physical Production Systems zu befriedigen, und*
>
> *dass alle fünf Elemente des Referenzmodells gleichzeitig, abgestimmt und widerspruchsfrei bei der Definition der Infrastruktur entworfen werden müssen, weil sie sich wie gezeigt alle gegenseitig beeinflussen.*

Es reicht nicht aus, Infrastruktur-Dienste zu definieren und die Befriedigung der NFA auf das Engineering konkreter Systeme zu verschieben.

Betrachtet man die bislang vorhandenen Ansätze, so wird offensichtlich, dass viele Elemente existieren, es aber an **Rigorosität** fehlt. In IoT-A wird z.B. in einem Deliverable (D2.3, Orchestration of distributed IoT service interactions, [13]) eine Taxononomie von NFA aus der

Literatur erwähnt und es wird versucht einige NFA quantitativ zu definieren, allerdings nicht in anderen Dokumenten der gesamten Spezifikation und auch nicht im eigentlichen Referenzmodell (D1.4, Converged architectural reference model for the IoT v2.0, [13]). Es bleibt auch unklar, wie sich die NFA auf das IoT Domain Model auswirken.

Unsere Diskussionen anlässlich dieses Papers zeigen auch, dass die wirklichen Schwierigkeiten erst beginnen, wenn man versucht die Vorgehensweise praktisch und konkret umzusetzen.

Auf dem Weg zu einem Referenzmodell wird es notwendig sein klarer zu definieren, welche Anforderungen wann und in welche Spezifikations-Artefakte wie eingehen. Wir sehen an dieser Stelle erheblichen Forschungsbedarf.

[1] Acatech, Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0, 2013

[2] VDI/VDE, Status Report Referenzarchitektur Industrie 4.0 (RAMI4.0)

[3] Industrial Internet Consortium, Industrial Internet Reference Architecture V1.0, http://www.iiconsortium.org

[4] IoT-A project, Deliverable D6.2 – Updated Requirements List, siehe [13]

[5] T. Usländer, R. Denzer and ORCHESTRA Consortium, Reference Model for the ORCHESTRA Architecture (RM-OA) V2 (Rev 2.1), OGC Best Practive Paper OGC 07-097, August 2007, portal.opengeospatial.org/files/?artifact_id=20300

[6] T. Usländer, R. Denzer, Requirements and Open Architecture of Environmental Risk Management Information Systems, in: B. van der Walle (ed.), Information Systems for Emergency Management, ISBN 978-0-7656-2134-4, pp. 344-368, 2010

[7] ORCHESTRA Consortium, ORCHESTRA User Requirements of Environmental Risk Management, DOI: 10.13140/RG.2.1.2233.0728

[8] T. Usländer, Service-Oriented Design of Environmental Information Systems, KIT Scientific Publishing, ISBN 978-3-86644-499-7

[9] ISO/IEC 10746-1:1998, Reference Model of Open Distributed Processing, www.iso.org

[10] ORCHESTRA Consortium, RM OA V2 Annex A1 Development Dimensions, DOI: 10.13140/RG.2.1.2936.7442

[11] ORCHESTRA Consortium, RM-OA-V2 Annex A2 Requirements for the Orchestra Architecture and the Orchestra Service Networks, DOI: 10.13140/RG.2.1.1232.8083

[12] ORCHESTRA Consortium, ORCHESTRA Abstract Service Specifications, DOI: 10.13140/RG.2.1.1757.0962

[13] IoT-A website, www.iot-a.eu

[14] SANY Sensor Service architecture, portal.opengeospatial.org/files/?artifact_id=35888

[15] J. Douglas, T. Usländer, G. Schimak, J. Esteban, R. Denzer, An Open Distributed Architecture for Sensor Networks for Risk Management, Journal Sensors 2008, 8, pp. 1755-1773

[16] M. Schleipen, Requirements and concept for Plug-and-Work, at Automatisierungstechnik; Vol. 63, Issue 10, pp. 801-820, 2015