Komponierbarkeit nichtfunktionaler Eigenschaften – Versuch einer Definition*

Matthias Werner
TU Berlin

Jan Richling
Humboldt-Universität zu Berlin

Zusammenfassung

„Komponierbarkeit“ ist eine oft geforderte Eigenschaft. Jedoch gibt es kaum Ansätze zu einer konsistenten Definition. In diesem Artikel präsentieren wir unser Konzept von Komponierbarkeit, das sich insbesondere (aber nicht ausschließlich) zur Behandlung nichtfunktionaler Eigenschaften eignet. Als Beispiel für die Komponierbarkeit einer Architektur stellen wir das *Message Scheduled System* vor.

1 Einführung

Durch die ständig wachsende Komplexität fällt es zunehmend schwerer, zu verstehen, was in Rechnersystemen vor sich geht. Der Entwurf von Systemen wird dadurch behindert.

Um die Übersicht zu behalten, wird in der Regel die Technik des *divide et impera* angewandt: Ein System wird in eine Menge von Subsystemen zerlegt, die leichter beherrschbar sind. Dies wird gegebenenfalls solange rekursiv wiederholt, bis die einzelnen Subsysteme einfach zu überschauen sind.

Eine solche Zergliederung hat jedoch nur dann einen positiven Effekt, wenn bei der Konstruktion des Gesamtsystems oder beim Umgang mit ihm die Struktur der Subsysteme nicht mehr beachtet werden braucht. Man beschränkt sich dann darauf, nur wenige Eigenschaften eines Subsystems zu kennen. Bei einer Vielzahl von Eigenschaften funktioniert diese Reduktion jedoch nicht ohne weiteres. Dies ist vor allen bei *nichtfunktionalen Eigenschaften* der Fall.

Unter nichtfunktionalen Eigenschaften verstehen wir hier alle Eigenschaften, die bei einer Betrachtung des *funktionellen Kerns* in der Regel abstrahiert werden, z.B. Zeitverhalten, Ressourcenverbrauch, Verlässlichkeitseigenschaften usw.[1]

Das klassische *divide et impera* versagt bei nichtfunktionalen Eigenschaften in der Regel deshalb, weil das nichtfunktionale Verhalten meist nicht entlang der funktionalen Schnittstellen separiert werden kann.

Im Zusammenhang mit der Kreation komplexer Systeme aus einfacheren Bestandteilen wird deshalb häufig die Forderung nach „Komponierbarkeit“ (*composability*) gestellt.

Entsprechend tauchte die Frage nach Komponierbarkeit auch im Rahmen einer Dis-

*Diese Arbeit wurde teilweise unterstützt von DaimlerChrysler Research Berlin.

[1] Man beachte, daß dieses (informelle) Verständnis etwas vom klassischen Konzept der Dienstgüte (*quality of service*) abweicht, das häufig mit dem der nichtfunktionalen Eigenschaften identifiziert wird.

kussion innerhalb einer Forschungskooperation über nichtfunktionale Eigenschaften eingebetteter Systemen auf. Dabei stellte es sich heraus, daß unter den Teilnehmern lediglich ein intuitives und in der Regel inkonsistentes Verständnis über das Konzept der Komponierbarkeit vorherrschte. Die vorliegende Arbeit stellt ein Ergebnis des durch diese Diskussion angestoßenen Denkprozesses dar und versucht, das intuitive Verständnis durch eine systematischere Erörterung des Konzeptes Komponierbarkeit zu ersetzen.

Die Arbeit ist wie folgt strukturiert: Abschnitt 2 diskutiert den Stand der Wissenschaft. Im Abschnitt 3 werden grundlegende Begriffe unseres Komponierbarkeitskonzepts dargelegt, die im Abschnitt 4 vertieft und formal definiert werden. Als ein Beispiel für eine in Bezug auf das Zeitverhalten komponierbare Architektur wird im Abschnitt 5 das *Message Scheduled System* (MSS) vorgestellt. Der Artikel schließt mit einer Zusammenfassung und einem Ausblick.

2 Verwandte Arbeiten

Betrachtet man die Veröffentlichungen, die auf „Komponierbarkeit“ Bezug nehmen, sind zwei Feststellungen zu machen:

1. Komponierbarkeit ist ein aktuelles Forschungsthema. Folglich taucht der Begriff auch in einer Vielzahl von Veröffentlichungen auf.
2. Es ist unklar, was Komponierbarkeit eigentlich ist. Nur in wenigen Arbeiten zu diesem Thema wird eine Definition gegeben.

Obwohl bereits 1996 in [7] der Mangel an konsistenten Definitionen für dieses Konzept beklagt wird, hat sich der Zustand seitdem kaum verbessert.

Am häufigsten wird der Begriff im Software Engineering benutzt, insbesondere im Rahmen der Objektorientierten Programmierung (OOP) und der Aspektorientierten Programmierung (AOP) (für Überblicke zu OOP siehe z.B. [9, 10], zu AOP siehe z.B. [2, 16, 15]). Eine typische Definition von Komponierbarkeit in diesem Kontext gibt Bergmans in [1]:

Definition 1 *Composability allows for the modular specification of modules with multiple independent concerns.*

Es bleibt in Definition 1 die Frage offen, *was* eigentlich das Attribut „Komponierbarkeit“ besitzt.

Mitunter wird ein Komponierbarkeitsbegriff im Softwareengineering für einzelne Softwaresysteme oder Kalküle festgelegt. Dieser ist dann außerhalb dieser Systeme nicht anwendbar. Wir sind mehr an einer Definition interessiert, die unabhängig vom betrachteten System gültig ist. Eine solche Definition gibt Kopetz in [4]:

Definition 2 *An Architecture is said to be composable with respect to a specified property if the system integration will not invalidate this property, once the property has been established at the subsystem level. Examples of such properties are timeliness or testability. In a composable architecture, the system properties follow from the subsystem properties.*

Genau wie Definition 2 sehen wir Komponierbarkeit als eine Eigenschaft der Architektur an. Jedoch beschränkt sich unser Konzept nicht auf den Erhalt von Eigenschaften auf Subsystemebene, sondern unterscheidet mehrere Arten der Komponierbarkeit.

Eine weitere Definition gibt Malek [8]. Er benutzt Komponierbarkeit als ein Maß für die Komplexität der Komposition:

Definition 3 *Given elements with properties. They are composable, iff the calculation of compositions' properties need polynomial time.*

Dies ist ein interessanter Ansatz, um den bei der Konstruktion komplexer Systeme entstehenden Aufwand abzuschätzen. In unser Definition betrachten wir diesen Aspekt bisher jedoch nicht, berücksichtigen ihn aber in unser Beispielarchitektur MSS (Abschnitt 5).

3 Konzepte und Begriffe

In diesem Abschnitt möchten wir einige Grundbegriffe einführen. Wir geben zunächst intuitive Beschreibungen; eine Formalisierung erfolgt im Abschnitt 4.4.

Allen unseren Überlegungen liegt der Begriff der „Architektur“ zugrunde. Man beachte, daß, obwohl wir bei der Begriffsfindung Rechnersysteme im Sinn hatten, unser Begriff nicht auf Computer oder gar reine Softwaresysteme beschränkt ist.

„Architektur“ hat in der Umgangssprache zwei verschiedene Bedeutungen. Einerseits meint „Architektur“ einen Satz von Regeln, wie etwas zu konstruieren ist (Design), und zum anderen wird der Begriff auch benutzt, um etwas zu bezeichnen, das unter Benutzung solcher Regeln konstruiert worden ist.

Hier wird der Begriff „Architektur“ in der ersten Bedeutung benutzt: Eine Architektur gibt einen Satz von Regeln vor, nach denen Dinge konstruiert werden können. Diese Dinge werden dann als „System“ bezeichnet.

Ein solches System besteht aus „Elementen“ als Bausteinen[2]. Ein System (als eine Menge zusammengefaßter Elemente) ist dabei ebenfalls ein Element. Das bedeutet, daß die Begriffe „System“ und „Element“ in Bezug auf eine Architektur austauschbar sind. Vom Standpunkt eines Benutzers aus existiert dagegen ein Unterschied: Ein System sollte eine gewisse Funktionalität und einen Zweck haben, während die Funktion eines Elements die Fähigkeit ist, mit anderen Elementen verbunden werden zu können.

Ein Element oder ein System kann verschiedene Eigenschaften haben. Genaugenommen ist alles, das ein Element von anderen unterscheidbar macht, eine Eigenschaft. Unterschieden wird dabei zwischen der Existenz einer Eigenschaft und ihrem Wert. Beispielsweise kann ein Werkzeug die Eigenschaft „Farbe“ haben, und der dazugehörige Wert mag „rot“ oder eine Wellenlänge von 650 nm sein. Eine andere Eigenschaft ist die Fähigkeit, ob das Werkzeug zum Lösen von Schrauben benutzt werden kann. In diesem Fall ist der Wert „ja“ oder „nein“, oder eine Zahl, die den Grad der Benutzbarkeit ausdrückt. Es gibt jedoch auch Eigenschaften, die auf bestimmte Systeme oder Elemente nicht anwendbar sind. Zum Beispiel kann die Laufzeit eines Computerprogramms leicht bestimmt werden, während die Laufzeit eines Hammers keinen Sinn macht. In solch einem Fall wird definiert, daß die Eigenschaft für das betrachtete Element nicht existiert.

Jedes Element hat unendlich viele Eigenschaften[3], aber nicht alle davon sind von Interesse für einen gegebenen Zweck. Innerhalb einer Architektur genügt in den meisten Fällen sogar ein relativ kleiner Satz von Eigenschaften, um ein Element zu charakterisieren. Diese Eigenschaften werden „Basisqualitäten“ (oder kurz: Qualitäten) genannt, aus denen alle anderen Eigenschaften abgeleitet werden können. Beispielsweise kann die Eigenschaft, daß ein Element zum Knacken von Nüssen benutzt werden kann, aus den Basisqualitäten für die Masse und die Beschaffenheit der Oberfläche abgeleitet werden. Damit können sowohl das Siegel von England [17] oder ein Mikroskop [5] zum Knacken von Nüssen benutzt werden. Wenn ein Element Teil eines Systems wird (eines anderen Systems, als es selbst eines ist), dann können sich seine Eigenschaften ändern.

Man beachte, daß es genau genommen keine „unkomponierten“, also freien Elemente gibt. Wenn über ein einzelnes Element geredet wird, so ist es stets Teil eines Systems, das wir als „Standardumgebung“ bezeichnen. Dieser Fakt wird in den weiteren Überlegungen jedoch vernachlässigt.

[2] Der populäre Begriff „Komponente“ wird hier nicht benutzt, da viele Ansätze „Komponente“ benutzen, um einen Unterschied zu „Konnektor“ zu betonen, vgl. [14].

[3] Eine unendliche Anzahl von Eigenschaften kann bereits nach dem Schema „Element x ist (nicht) benutzbar für Zweck y“ erzeugt werden

4 Komponierbarkeit als Eigenschaft einer Architektur

4.1 Architektur, Elemente und Kompositionsoperatoren

Ein *Element* als „Grundbaustein“ wird repräsentiert durch ein Tupel von Variablen und Funktionen $c(v_0, v_1, ...v_n, f_0, f_1, ...f_m)$. Die Variablen repräsentieren die Qualitäten des Elements. Innerhalb eines Systems können Variablen statisch an Werte gebunden sein, sie können ihre Werte aber auch dynamisch mit der Zeit ändern und sind lediglich an einen Wertebereich gebunden. Diese Werte können als (Teil-)Zustände gesehen werden, und das Element selbst damit als eine State-Machine, in der die Funktionen f_i den Übergang von einem Zustand zum nächsten repräsentieren. Die State-Machine kann diskret oder kontinuierlich sein in Abhängigkeit von der Natur des Elements und den möglichen Werten der Variablen.

Einige der Variablen können als ein Interface des Elements gesehen werden, wenn es möglich ist, ihre Werte von außerhalb des Elements zu ändern, andere sind durch das Element so gekapselt, daß sie außerhalb des Elements nicht oder nicht direkt sichtbar sind.

Definition 4 (Architektur) *Sei E die Menge aller möglichen Elemente und O die Menge aller zweistelligen nichtleeren*[4] *Kompositionsoperatoren mit $\circ \in O \wedge \circ : E \times E \to E$. Eine Systemarchitektur A ist dann:*

$$A : E \times E \times O \to \{true, false\}$$

Die Funktion $A(d, e, \circ), d, e \in E, \circ \in O$ ist $true$, wenn die Architektur A die Komposition der Elemente d, e unter Benutzung des Kompositionsoperators $\circ$ erlaubt, und $false$ anderenfalls.

Mit anderen Worten: Eine Architektur beschreibt, wie und welche Elemente komponiert werden können, um andere Elemente bzw. Systeme zu erhalten, stellt also, wie im Abschnitt 3 erläutert, einen Satz von Regeln dar, wie Elemente zu neuen Elementen zusammengefügt werden können.

Innerhalb der Menge O aller Kompositionsoperatoren ist eine Menge $O(A) \subset O$ definiert, die alle gültigen Kompositionsoperatoren von A umfaßt:

Definition 5 (Gültige Komposition)
$\circ \in O(A) \Leftrightarrow \exists a, b \in E : A(a, b, \circ) = true$

Analog wird innerhalb von E die Menge $E(A) \subset E$ als Menge gültiger Elemente der Systemarchitektur A definiert:

Definition 6 (Gültiges Element)

$$\begin{aligned} e \in E(A) \Leftrightarrow (&\exists \circ \in O, f \in E, \\ &\big(A(e, f, \circ) = true\big) \vee \\ &\big(A(f, e, \circ) = true\big)\big) \vee \\ &\big(\exists \circ \in O(A), f, g \in E : e = f \circ g\big) \end{aligned}$$

Die Menge $E(A)$ umfaßt alle in A möglichen Elemente, einschließlich solcher, die ihrerseits aus anderen komponiert worden sind. Wir unterscheiden atomare Elemente einer Architektur (also solcher Elemente, die nicht aus anderen Elementen der Architektur durch Komposition entstehen können):

Definition 7 (Atomares Element)

$$\begin{aligned} &atomic(e, A) \Leftrightarrow \big(e \in E(A)\big) \wedge \\ &\big(\forall f, g \in E(A), f \neq e, g \neq e, \forall \circ \in O(A), \\ &\qquad f \circ g \neq e\big) \end{aligned}$$

Ein Element, das in einer Architektur atomar ist, muß dies in einer anderen nicht sein.

Neben atomaren Elementen gibt es auch finale Elemente, also solche, die mit keinem anderen Element der Architektur komponiert werden können:

[4] Ein zweistelliger Kompositionsoperator $\circ$ ist leer genau dann, wenn gilt: $\forall e, f \in E(\circ) : ((e \circ f = e) \vee (e \circ f = f))$, wobei $E(\circ) \subseteq E$ der Definitionsbereich von $\circ$ ist. Eine solche leere Komposition liefert als Ergebnis also eines der beiden Eingangselemente.

Definition 8 (Finales Element)

$$final(e) \Leftrightarrow (e \in E(A)) \wedge \big(\forall f \in E(A), \forall \circ \in O(A), (A(e,f,\circ) = false) \wedge (A(f,e,\circ) = false)\big)$$

4.2 Basisqualitäten und Eigenschaften

Kompositionen finden auf der Ebene von Elementen statt, die ihrerseits eine Reihe von Basisqualitäten (repräsentiert durch die Variablen) und Zustandsübergängen (repräsentiert durch die Funktionen) haben, auf die eine Komposition ebenfalls einen Einfluß hat.

Wenn zwei Elemente e_1 und e_2 zu einem System e_S komponiert werden, können die Element-Variablen von e_S aus den Element-Variablen von e_1 und e_2 auf eine der folgenden Arten abgeleitet werden:

1. **Invariante.** Eine Variable bleibt unverändert. Die Variable gehört weiterhin zu e_1 oder e_2 innerhalb von e_S. Das bedeutet, daß eine bestimmte Eigenschaft des Elements weiterhin innerhalb des komponierten Systems identifiziert[5] werden kann. Solch eine Variable heißt „invariante Element-Qualität“ in Bezug auf einen Operator $\circ$. Solche Variablen werden Teil des Zustandes des neuen Elements ohne eine Änderung ihres Wertes.
2. **Gebundene Qualität** Eine Variable wird an einen anderen Wert oder Wertebereich gebunden und gehört dabei weiterhin zu einem der Elemente. Eine Variable dieser Art heißt „gebundene Element-Qualität“ in Bezug auf einen Operator $\circ$ und wird Teil des Zustandes des neuen Elements, aber der Wert kann sich ändern.
3. **Verschwindende Qualität** Eine Variable kann innerhalb des komponierten Systems nicht mehr identifiziert werden und wird als „verschwindende Element-Qualität“ bezeichnet.
4. **Auftauchende Qualität** Eine neue Variable wird generiert. Sie gehört zum System, und ihr Wert oder Wertebereich kann von Elementvariablen von e_1 oder e_2 abhängen. Eine solche Variable wird „auftauchende Qualität“ von e_S genannt. Die neue State-Machine hat in solch einem Fall Zustandsvariablen, die in den anderen beiden State-Machines nicht existieren.
5. **Übertragene Qualität** Eine Variable kann an eine andere Variable gebunden werden, die damit zum System gehört - zwei Zustände werden damit verbunden. Eine solche Variable heißt „übertragene Qualität“ und kann als die Kombination des Verschwindens und Neuauftauchens gesehen werden.

Die Komposition zweier Elemente hat nicht nur auf die Variablen Einfluß, sondern auch auf die Funktionen beider Elemente. Diese werden in gleicher Weise wie die Variablen behandelt.

Dieser hier vorgestellte Ansatz unterscheidet sich von den meisten klassischen Kompositionsmodellen, in denen die Komposition lediglich zwei Komponenten über ein Interface verbindet. In dem hier gezeigten Modell bedeutet das, daß die Variablen unverändert bleiben und nur die Interfacevariablen über auftauchende Funktionen verbunden werden. Das Modell geht jedoch weiter, und erlaubt auch Änderungen an der Struktur und den Werten der Elemente.

Eine Eigenschaft eines Elements ist eine Klausel P, die testet, ob eine gewisse Qualität existiert oder der Wert dieser Qualität Bedingungen erfüllt. Beispielsweise führt die Eigenschaft „X ist rot“ zu $Red(X) : (w(X) \neq \perp) \wedge w \in (620nm, 680nm)$, wobei w die Wellenlänge des von X reflektierten Lichts ist.

[5] Identifizierbar bedeutet an dieser Stelle: Meßbar oder beobachtbar ohne Dekomposition des Systems

4.3 Erreichbarkeits- und Sicherheitseigenschaften von Architekturen

Elemente innerhalb einer Architektur haben Eigenschaften, die durch die Variablen und Funktionen repräsentiert werden. Die Architektur selbst hat jedoch ebenfalls Eigenschaften. Hier sollen zwei Arten von Eigenschaften betrachtet werden, die sich wiederum auf Eigenschaften von in der Architektur möglichen Elementen beziehen. Bei der ersten Art handelt es sich darum, daß die Architektur die Konstruktion von Elementen mit bestimmten Eigenschaften ermöglicht; bei der zweiten darum, daß die Architektur sicherstellt, daß jedes konstruierte Element bestimmte Eigenschaften hat.

Ersteres wird „Erreichbarkeitseigenschaft“ genannt und ist folgendermaßen definiert:

Definition 9 (Erreichbarkeitseigenschaft)

$$R(A,P) \Leftrightarrow \exists e \in E(A), P(e)$$

Die andere Art von Eigenschaften heißt „Sicherheitseigenschaften“:

Definition 10 (Sicherheitseigenschaft)

$$S(A,P) \Leftrightarrow \forall e \in E(A), P(e)$$

Erreichbarkeitseigenschaften einer Architektur stellen sicher, daß man innerhalb der Architektur ein bestimmtes Ziel erreichen kann, d.h., ein bestimmtes Element bzw. System konstruieren kann.

Sicherheitseigenschaften sind Invarianten der Architektur. Alle Elemente einer Architektur besitzen dann die Eigenschaft, auf die sich die Sicherheitseigenschaft der Architektur bezieht.

4.4 Komponierbarkeit

Eine Relation $a \prec b$ soll ausdrücken, daß das Element a benutzt wurde, um b zu komponieren:

Definition 11 (Ist-enthalten-Relation)

$$\begin{aligned} a \prec b \Leftrightarrow a,b \in E(A), (a = b) \vee \\ (\exists x,y \in E(A), \circ \in O(A), (x \circ y = b) \wedge \\ ((a \prec x) \vee (a \prec y))) \end{aligned}$$

Weiterhin definieren wir eine Relation $\diamond$, die ausdrückt, daß zwei Elemente gemeinsam Teil von wenigstens einem anderen innerhalb von A gültigen Element sind:

Definition 12 (Verträglichkeitsrelation)

$$\begin{aligned} a \diamond b \Leftrightarrow a,b \in E(A), \\ \exists e \in E(A), (a \prec e) \wedge (b \prec e) \end{aligned}$$

Mit Hilfe der gegebenen Definitionen können wir nun *Komponierbarkeit* definieren. Eine Architektur A wird *komponierbar* genannt, wenn jedes gültige atomare Element mit jedem anderen verbunden werden kann (unter Umständen indirekt):

Definition 13 (Komponierbarkeit)

$$\begin{aligned} composable(A) \Leftrightarrow \forall a,b \in E(A), \\ (atomar(a) \wedge atomar(b)) \rightarrow a \diamond b \end{aligned}$$

Üblicherweise wird komponiert, um bestimmte Eigenschaften zu erhalten. Das führt zu der Definition von „Komponierbarkeit in Bezug auf eine Eigenschaft“:

Eine System-Architektur A wird komponierbar in Bezug auf eine Erreichbarkeitseigenschaft P (erreichbar komponierbar) genannt, genau dann wenn A komponierbar ist und $R(A,P)$ gilt:

Definition 14 (erreichbar komponierbar)

$$\begin{aligned} reachable\_composable(A,P) \Leftrightarrow \\ composable(A) \wedge R(A,P) \end{aligned}$$

Eine System-Architektur A wird komponierbar in Bezug auf eine Sicherheitsseigenschaft P (sicher komponierbar) genannt, genau dann wenn A komponierbar ist und $S(A,P)$ gilt:

Definition 15 (sicher komponierbar)

$$safe\_composable(A, P) \Leftrightarrow composable(A) \wedge S(A, P)$$

Das bedeutet, daß A in Bezug auf eine Eigenschaft erreichbar komponierbar ist, wenn wenigstens ein Element von A diese Eigenschaft hat, und A sicher komponierbar in Bezug auf eine Eigenschaft ist, wenn jedes Element diese Eigenschaft hat.

5 Das Message Scheduled System

In diesem Abschnitt stellen wir als Beispiel einer komponierbaren Architektur das *Message Scheduled System* (MSS) vor. Wir präsentieren hier nur einige Details zu MSS, die genaue Beschreibung findet sich in [13, 11, 12].

5.1 Die Grundidee von MSS

MSS ist eine Architektur für Echtzeitsysteme, die es erlaubt, Komponierbarkeitsentscheidungen *zur Laufzeit* durchzuführen. Die Grundidee von MSS ist es, alle Entscheidungen über das Hinzufügen oder Entfernen von Komponenten (Komponierbarkeitsentscheidungen) auf Scheduling-Entscheidungen auf drei verschiedenen Ebenen abzubilden. Diese Scheduling-Entscheidungen können auf Grundlage bekannter und etablierter Verfahren wie RMA oder EDF [6] getroffen werden.

Auf dieser Basis kann in MSS das zeitliche Verhalten einer Komponente beziehungsweise eines Systems beschrieben werden, ohne daß die Beschreibung alle Details der Komponente oder des Systems enthält — es genügen die Informationen, die das Scheduling der drei Ebenen betreffen. Dabei werden die Parameter bereits eingefügter Komponenten in zusammenfassenden Parametern wie einer Auslastung verborgen. Es ist also nicht erforderlich, daß auf jedem Knoten globales Detailwissen über das gesamte System vorhanden ist.

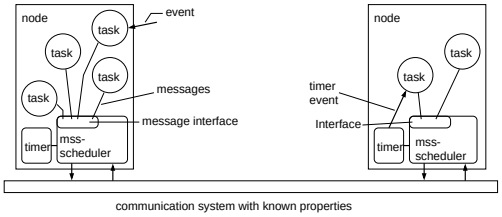

Abbildung 1: Message Scheduled System

MSS kennt drei verschiedene Arten von atomaren Elementen (Abbildung 1): Das kleinste Element ist eine *Task* (E_t). Dies ist im MSS-Kontext eine Abarbeitungseinheit, die ausgehend von einem Satz von Eingangsnachrichten einen Satz von Ausgangsnachrichten erzeugt. Die Betrachtung einer solchen *Task* als Element ist unter dem Aspekt der Komponierbarkeit sinnvoll, da es in vielen Fällen genügt, eine *Task* zu einem System hinzuzufügen, um neue Funktionalität zu erhalten.

Die nächste Form von Elementen in MSS sind *Knoten* (E_n). Ein *Knoten* in MSS ist eine Maschine, die in der Lage ist, *Tasks* auszuführen (und zu schedulen) und die von diesen *Tasks* benötigten und erzeugten Nachrichten zu empfangen beziehungsweise weiterzuleiten.

Das dritte atomare Element ist das Kommunikationssystem (E_b), das in der Lage ist, Nachrichten zu transportieren. In jedem System mit MSS-Architektur gibt es genau ein solches Element.

Durch Komposition der atomaren Elemente entstehen *Systeme*. Sie bestehen aus einem oder mehreren Knoten, die über das Kommunikationssystem (ein Netzwerk, das durch MSS spezifiziert wird) miteinander verbunden sind, wobei auf den Knoten wiederum Tasks laufen.

Das Verbindungsglied zwischen den Elementen sind *Nachrichten*, die in einer Publisher/Subscriber-Semantik durch die Tasks unter Benutzung des MSS-Schedulers des jeweiligen Knotens über das Kommunikationssystem gesendet oder empfangen

werden. Jede Task hat eine spezielle Nachricht (*wake up message, WUP*), deren Ankunft die Ausführung der Task auslöst. *Minimal interarrival times, MIT* der Nachrichten eines Typs sind bekannt und Teil der Komponierbarkeitsentscheidungen (Scheduling auf der Ebene des Kommunikationssystems — Begrenzung der durch einen Nachrichtentyp genutzten Bandbreite)

5.2 MSS und Komponierbarkeit

5.2.1 Komponierbarkeitsentscheidungen

MSS unterscheidet sich von den meisten anderen komponierbaren Architekturen (z.B. TTA [3]) dadurch, daß Komponierbarkeitsentscheidungen (d.h. die Entscheidung, ob eine Komposition möglich ist, ohne die Architektur bzw. eine Sicherheitseigenschaft zu verletzen) zur Laufzeit getroffen werden kann.

Die MSS-Architektur benutzt auf drei verschiedenen Ebenen Scheduling-Verfahren, um die Einhaltung von Deadlines — sowohl von Tasks als auch der Übertragung von Nachrichten — zu garantieren. Die Einhaltung aller Deadlines ist eine Sicherheitseigenschaft von MSS. Eine Komposition erfolgt dadurch, daß zu einem existierenden System ein atomares Element oder ein bereits komponiertes System hinzugefügt werden soll. Erst zu diesem Zeitpunkt wird entschieden, ob eine Komposition durchführbar ist, d.h., ob nicht durch sie diese Sicherheitseigenschaft verletzt werden würde.

Die benötigten Eingangsdaten für die Komponierbarkeitsentscheidung sind aus den Elementenbeschreibungen für Tasks (Ausführungszeiten und MITs), für Knoten (Rechenkapazität) und für das Kommunikationssystem (Bandbreite) ableitbar.

Der Aufwand dieser Berechnung ist dabei linear zu der Anzahl der Schnittstellen der Elemente — also auch linear zu der Anzahl der Elemente. Ist eine Komponierbarkeitsentscheidung positiv, so wird ein neues System komponiert — die Komposition ist erfolgreich. Andernfalls bleibt das ursprüngliche System bestehen.

5.2.2 Komposition

Mit Hilfe des im Abschnitt 4 dargestellten Frameworks, kann MSS mit einer Reihe von Aussagen beschrieben werden:

$$\forall e \in E(MSS), (atomic(e)) \vee (e_b \prec e, e_b \in E_b)$$
$$\forall e_1 \in E(MSS), e_2 \in E_b, e_1 \circ e_2 \Leftrightarrow \neg\exists e_3 \in E_b, e_3 \prec e_1$$

D.h., in jedem nichtatomaren MSS-System gibt es genau einen Bus

$$\forall e_1 \in E(MSS), \neg atomic(e_1), \forall e_2 \in E_n, e_1 \circ e_2$$

D.h., zu einem nichtatomaren MSS-System können stets neue Knoten hinzugefügt werden

$$\forall e_1, e_2 \in E(MSS), e_{t_1}, e_{t_2} \in E_t, e_{t_1} \prec e_1, e_{t_2} \prec e_2, e_1 \circ e_2 \Leftrightarrow comp\_decision(e_1, e_2)$$

D.h., zwei Elemente, die Tasks enthalten, lassen sich nur bei positiver Komponierbarkeitsentscheidung[6] komponieren.

Um die Komponierbarkeit in Bezug auf Eigenschaften (vgl. Definitionen 14 und 15) für MSS zu betrachten, müssen gewünschte Eigenschaften festgelegt werden. MSS garantiert, wie oben erwähnt, daß alle Taskdeadlines im komponiertem System eingehalten werden. Diese Eigenschaft sei als P_T bezeichnet. Desweiteren garantiert MSS die Einhaltung aller End-to-end-Nachrichtenauslieferungszeiten. Diese Eigenschaft sei als P_M bezeichnet.

Man beachte, daß P_T eine invariante Eigenschaft (bzw. eine Menge von invarianten Eigenschaften) ist, während P_M eine auftauchende Eigenschaft ist — auf Knotenebene ist es z.B. sinnlos, End-to-end-Nachrichtenzeiten zu betrachten.

Bezogen auf P_T und P_M gilt für MSS:

$$safe\_composable(MSS, P_T) \wedge safe\_composable(MSS, P_M) \quad (1)$$

[6] Zur Erörterung der Berechnung der Komponierbarkeitsentscheidung siehe [13].

(1) ist bisher noch eine Behauptung. Wir sind dabei, diese formal mit Hilfe von zeitbehafteten Petri-Netzen zu beweisen. Unserer Lösungsansatz ist in [11] beschrieben.

6 Zusammenfassung und Ausblick

Komponierbarkeit ist eine vielgesuchte Eigenschaft. Wir betrachten Komponierbarkeit als ein Konzept, das über Domaingrenzen hinaus betrachtet werden sollte. Unsere Komponierbarkeitsdefinitionen erweitern den Ansatz von Definition 2 aus [4], der Komponierbarkeit als eine Eigenschaft einer Architektur betrachtet. Anders als Kopetz, beschränken wir uns nicht auf den Erhalt von Eigenschaften auf Subsystemebene, sondern betrachten auch mögliche Eigenschaften des Gesamtsystems.

Mit der Unterscheidung von sicherer und erreichbarer Komponierbarkeit ermöglichen wir die Berücksichtigung verschiedener nichtfunktionaler Eigenschaften. Jedoch läßt sich unser Konzept auch nach Belieben auf funktionale Eigenschaften anwenden.

Die *Message Schedule Architecture* demonstriert, wie eine Architektur sicher komponierbar (in diesen Fall in Bezug auf zeitliche Eigenschaften) sein kann. Andere Architekturen, die z.B. Sicherheit (*security*) jedes in ihr komponierten Systems garantieren, können mit unserem Ansatz bewertet werden.

Die nächsten Schritte unserer Forschung werden darin bestehen, zu untersuchen, wie sich bekannte Implementationsansätze mit unserem Konzept verbinden lassen. Dabei werden wir besonders die aspekt-orientierte Programmierung betrachten, aber auch Ansätze des Hardware-Software-Codesigns. Ziel wird es dabei sein, die Designmittel so einzuschränken, daß eine sichere Komposition für die gewünschte Eigenschaft garantiert (und verifiziert) werden kann, bzw. die Existenz einer erreichbaren Komposition innerhalb einer Architektur nachgewiesen werden kann.

Literatur

[1] Lodewijk M.J. Bergmans. Composability: Why, what, and how? In *Workshop on Composability Issues in Object-Orientation — Tenth European Conference on Object-Oriented Programming, July 8-12, 1996, Linz - Austria.*

[2] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland.* Springer-Verlag LNCS 1241, Jun 1997.

[3] H. Kopetz, M. Braun, C. Ebner, A. Krueger, D. Millinger, R. Nossal, and A. Schedl. The design of large real-time systems: The time-triggered approach. In *IEEE Real-Time Systems Symposium*, pages 182–189, Vienna, Austria, 1995.

[4] Hermann Kopetz. *Real-Time Systems — Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, Norwell, Massachusetts 02061, 1997.

[5] Stanislaw Lem. The inquest. In *More Tales of Pirx the Pilot.* Harcourt Brace, 1982.

[6] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[7] Carine Lucas and Patrick Steyaert. Research topics in composability. In *Workshop on Composability Issues in Object-Orientation — Tenth European Conference on Object-Oriented Programming, July 8-12, 1996, Linz - Austria.*

[8] M. Malek. Definition of composability. private correspondence, 2000.

[9] Bertrand Meyer. *Object-Oriented Software Construction.* Prentice Hall PTR, 1997.

[10] Wolfgang Pree. *Design Patterns for Object-Oriented Software Development.* Addison-Wesley, 1995.

[11] J. Richling, L. Popova-Zeugmann, and M. Werner. Verification of non-functional properties of a composable architecture with petrinets. In *Proceedings of the CS&P'2001 Workshop*, 2001.

[12] J. Richling, M. Werner, and L. Popova-Zeugmann. Automatic composition of timed petrinet specifications for a real-time architecture. In *Proceedings of 2002 IEEE International Conference on Robotics and Automation*, Washington DC, 2002.

[13] Jan Richling. Message Scheduled System - A Composable Architecture for Embedded Real-Time-Systems. In *Prooceedings of 2000 Int. Conference on Parallel and Distributed Processing techniques and Applications (PDPTA 2000)*, volume 4, pages 2143–2150, Jun 2000.

[14] Mary Shaw. Constructing systems from parts: What students should learn about software architecture. In B. Randell, editor, *Software Architecture and Design*, pages VIII.1 – VIII.25. International Computers Limited and University of Newcastle upon Tyne, 1998.

[15] AOSD steering committee. Aspect-oriented software development. website. `http://www.aosd.net`.

[16] Focus theme AOP. Communications of the ACM. Vol. 40, October 2001.

[17] Mark Twain. *The Prince and the Pauper.* James R. Osgood and Co, Boston, 1882.