Qualitätssichernde Koevolution von Architekturen eingebetteter Systeme

Malte Lochau und Ursula Goltz
Institut für Programmierung und Reaktive Systeme
TU Braunschweig
{lochau|goltz}@ips.cs.tu-bs.de

Abstract: Eingebettete Software-Systeme sind heute allgegenwärtig und müssen zukünftig auch in Hinblick auf eine stetig steigende Lebensdauer und Wartbarkeit entworfen werden. Entsprechend müssen diese Systeme zunehmend evolvierbar sein, d.h. sowohl Fähigkeiten zur situationsbedingten Adaption im Betrieb aufweisen, als auch flexibel an sich ändernde Anforderungen anpassbar sein. Um die Komplexität von Auswirkungen von Anpassungen an diesen Systemen beherrschbar zu halten und Erosionseffekte zu verhindern, sind abstrahierende Architektur-Modelle eine wesentliche Grundlage für die Planung und strukturierte Durchführung von Änderungen im gesamten Lebenszyklus. Gleichzeitig kann durch Evolutions-begleitende Architektur-Evaluationen die nachhaltige Erfüllung von Qualitätskriterien wie Echtzeiteigenschaften, Sicherheit und Verfügbarkeit sichergestellt werden. Wir beschreiben die Einbettung von Maßnahmen zur fortgesetzten Qualitätssicherung in einen koevolutionären Betriebs- und Wartungsprozess und die hierfür erforderliche Konsistenz der Architektur-Spezifikation mit dem laufenden System. Der Ansatz wird anhand einer Fallstudie, einer adaptiven Robotersteuerung mit harten Echtzeitanforderungen, illustriert.

1 Einführung

1.1 Architekturen evolvierender Systeme

Die Software ist der zunehmend kritische Faktor für den Erfolg eingebetteter Systeme. Dabei entsteht ein steigender Anteil der Aufwände und Kosten für derartige Systeme durch Wartungs-, Anpassungs- und Weiterentwicklungstätigkeiten, also der *Evolution* der Software [EGG+09, Mas07], die gerade bei eingebetteten Systemen von der Umgebung und dem Anwendungskontext geradezu erzwungen wird [Leh96, Par94].

Der Architektur einer Software kommt zukünftig eine entscheidende Rolle zu, damit ein Software-System evolvierbar ist und bleibt. Dies gilt sowohl für kleinere Anpassungen wie Refactorings [RH06] oder dem Austausch von Komponenten, als auch für „revolutionäre“ Eingriffe, angefangen beim kompletten Reengineering von (Teil-) Systemen bis hin zur Migration ganzer Systeme auf neue Plattformen [Mas07]. Darüber hinaus müssen moderne Systeme zunehmend adaptiv sein, d.h. sich selbständig zur Laufzeit strategisch an neue Bedingungen anpassen können [SG07, SGM09]. Um die Komplexität der Evolution zukünftiger Systeme beherrschbar zu halten, muss bereits im Rahmen des Ent-

wurfs und der Modellierung die Anpassbarkeit als Kriterium bzw. Eigenschaft explizit berücksichtigt werden. Dies kann nicht allein durch den Einsatz moderner Technologien und Paradigmen und der damit verbundenen ”Verheißungen“, wie z.B. dem Einsatz Service-orientierter Entwurfsprinzipien, erzielt werden. Vielmehr sind neben Domänenspezifischen Lösungsmustern vor allem übergeordnete Prinzipien einer modularen, konfigurierbaren Architektur-Konzeption wesentlich für eine flexible Anpassbarkeit [BCK03].

Gerade an eingebettete Systeme werden aber auch weitere Qualitätsanforderungen gestellt, wie z.B. Sicherheit, Verfügbarkeit und Echtzeitfähigkeit. Im Kontext kontinuierlicher Evolution besteht somit auch die Notwendigkeit, ein fortlaufendes Qualitätsmanagement begleitend zum gesamten Lebenszyklus zu definieren. Dennoch können nicht alle erdenklichen zukünftigen Richtungen möglicher Weiterentwicklungen und Anpassungen beim Entwurf von System-Architekturen antizipiert werden [EGG+09]. Gerade für Systeme, die sich durch eine besonders hohe Langlebigkeit auszeichnen, können zukünftige, Lebenszyklus-übergreifende Entwicklungen nur schwer abgeschätzt werden, wie z.B. sich ändernde Anforderungen, Erosionen in der Umgebung oder der technischen Infrastruktur sowie die Portierung auf andere Plattformen. Methodiken zur planvollen Evolution von Software-Systemen, die auf strukturierten, Architektur-basierten Vorgehensweisen beruhen, müssen geeignete Dokumentationsansätze für die Nachverfolgbarkeit unternommener Anpassungen beinhalten. Dabei muss nicht nur die korrekte Funktionsfähigkeit des Systems gewährleistet bleiben, sondern gleichzeitig die Sicherung weiterer Qualitätseigenschaften im Betrieb möglich bleiben.

In diesem Beitrag beschreiben wir, inwieweit Architektur-Modelle nicht nur für den Entwurf und Implementierung sowie die Planung von Evolutionen eingebetteter Systeme von entscheidender Bedeutung sind, sondern auch die Grundlage für ein Lebenszyklusübergreifendes Anforderungs- und Qualitätsmanagement bilden können. Begleitend zur stetigen Anpassung des Systems kann durch modellbasierte Architektur-Evaluation eine fortlaufende Bewertung und Sicherung zu erfüllender Qualitätskriterien im Betrieb erfolgen. Die kontinuierliche *Koevolution* von Systemspezifikation und Systemimplementierung erfordert die Integration von Methodiken zur Ermittlung konkreter Architekturausprägungen des laufenden Systems in den Wartungsprozess. Somit können Verfahren sowohl für die *proaktive* als auch *reaktive* Planung von Architektur-Entscheidungen und -Evolutionen unter Abwägung aller relevanten Qualitätsziele etabliert werden. Zudem können Anpassungen geeignet dokumentiert und nachverfolgt werden, um die Auswirkungen unternommener Evolutionsschritte in nachfolgende Entscheidungen einfließen zu lassen.

1.2 Qualitätssicherung durch Architektur-Evaluation

Eine frühzeitige Bewertung der zu erwartenden Qualität eines Systems durch Architektur-Evaluation [BCK03] ist ein unverzichtbarer Bestandteil des Entwurfsprozesses für eingebettete Systeme. Wie in [FGB+07] für den Anwendungsbereich automotiver Systeme beschrieben wurde, ermöglicht ein strukturierter Evaluationsansatz die Integration von Bewertungstechniken für sämtliche Qualitätskriterien des zu entwickelnden Systems. Der

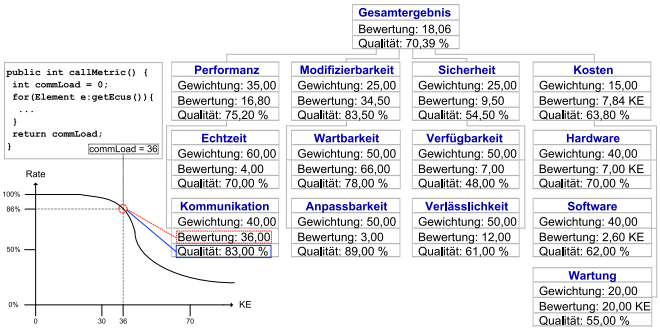

Abbildung 1: Evaluationsstruktur zur Architektur-Bewertung

prinzipielle Aufbau einer Evaluationsstruktur ist exemplarisch in Abbildung 1 dargestellt, für eine detaillierte Diskussion des Ansatzes verweisen auf [LMD+09, LMS+09]. Um zu einer Aussage über die zu erwartende Gesamtqualität des Systems auf Grundlage der Architektur zu gelangen, werden zunächst die je nach Projektzielen relevanten Einzelkriterien durch geeignete *Qualitätsattribute* charakterisiert und bewertet. Für die konkrete Bewertung dieser Kriterien hinsichtlich einer Architektur-Variante werden einheitliche *Bewertungstechniken*, wie z.B. Metriken oder Expertenbefragungen, definiert. Gerade Metriken können automatisiert auswertbar als integraler Bestandteil einer Komponente zur Anpassungsplanung implementiert werden können. Von besonderem Interesse für langlebige Systeme sind Bewertungstechniken zur Modifizierbarkeit der Architektur, bestehend aus a priori festzulegenden Anpassungsszenarien. Um aus den Bewertungsregebnissen der Einzelkriterien eine Gesamtqualität zu ermitteln, werden *Qualitätsraten* zur Normalisierung definiert. Diese ordnen Bewertungsergebnissen je nach Projektzielen und Constraints, z.B. Kostenbudgets oder QoS, eine Güte als Prozentzahl zu. K.O.-Kriterien in Form von harten Grenzwerten für Qualitätskriterien dürfen dabei nicht verletzt werden, um die Realisierbarkeit nicht zu gefährden. Die Integration zu einer Gesamtqualität erfolgt durch Hierarchisierung in Unter- und Oberkriterien mit entsprechenden Gewichtungen gemäß der Priorität der Qualitätsziele. Die Evaluationsstruktur kann dann zur Analyse, Optimierung und Dokumentation von Architektur-Varianten und -Entscheidungen dienen.

Bisher wurde dieser Ansatz zur Bewertung konzeptioneller Architektur-Varianten begleitend zum Entwurf automotiver Steuergerätenetzwerke, also vor der Implementierung und dem Betrieb des Systems, betrachtet. Für eingebettete Systeme mit evolutionsfähigen Architekturen wird zunehmend auch die Notwendigkeit bestehen, den Erhalt von Qualitätseigenschaften nicht nur initial, sondern auch begleitend zu den fortlaufenden Änderungen im Betrieb sicherzustellen.

1.3 Fallstudie: Architektur einer adaptiven Robotersteuerung

Für die nachfolgenden Überlegungen beziehen wir uns exemplarisch auf die „Parallel RObot Software Architecture - eXtended“ (PROSA-X) [SG07] als Beispiel für ein evo-

lutionsfähiges, verteiltes eingebettetes System mit hohen Qualitätsanforderungen. Abb. 2 zeigt die Prosa-X-Architektur zur Integration der Steuerungsmodule von Parallelrobotern. Die Steuerung der verwendeten Parallelkinematiken muss mit hoher Präzision erfolgen sowie harte Anforderungen bezüglich Echtzeit, Zuverlässigkeit und Sicherheit erfüllen. Darüber hinaus muss der Architektur-Aufbau flexible Anpassungen der Komponentenstruktur ermöglichen. Darüber hinaus wurde das System um eine *Self-Manager*-Komponente (Analyse-PC) zur Realisierung von Self*-Eigenschaften erweitert [SG07, SGM09]. Das System ist so in der Lage, ohne Eingriffe von außen adaptiv im Betrieb auf unterschiedliche Situationen, wie z.B. Knotenausfälle auf der Hardware-Plattform, wenn möglich unter Erhalt der Echtzeiteigenschaften zu reagieren.

2 Qualitätssichernde Koevolution von Architekturen eingebetteter Systeme

2.1 Evolution von Software-Systemen

Die Identifikation und Durchführung notwendiger Änderungsmaßnahmen an im Betrieb befindlichen (eingebetteten) Software-Systemen kann auf zwei Arten erfolgen:

Methodische Anpassungen: Eingriffe „von Hand“ als Teil der Systemwartung. Aufwand und Auswirkungen der Anpassungen variieren dabei von einfachen Rekonfigurationen oder dem Austausch von Komponenten, bis zu komplexen Umstrukturierungen der Gesamtarchitektur und dem Deployment. Ein sorgfältig strukturierter Architektur-Entwurf mit stabilen Komponentenschnittstellen und Konfigurationsparametern kann zur Anpassbarkeit von Systemen beitragen. Allerdings erhöht jede Maßnahme zur Auslegung auf potentielle Änderungen zumeist die Komplexität der Architektur. Zudem müssen gerade eingebettete Systeme trotz durchzuführender Anpassungen durchgängig verfügbar sein, was durch eine strategische Abfolge systematischer Anpassungsschritte erreicht werden kann.

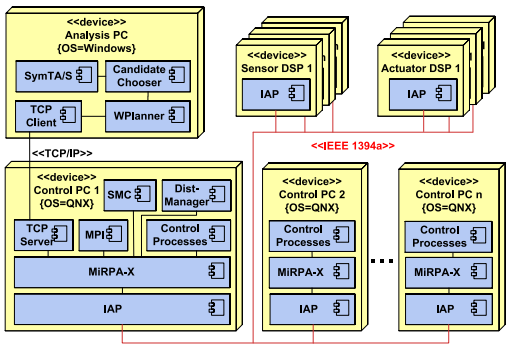

Abbildung 2: Architektur PROSA-X für Robotersteuerungen

Adaption: Die Fähigkeit des Systems, sich selbst zur Laufzeit durch geeignete Strategien an bestimmte Bedingungen oder Kontexte anzupassen, wie z.B. zur Kompensation ausfallender Systemressourcen hinsichtlich Verfügbarkeits-, Sicherheits- und Echtzeitanforderungen. Die Implementierung adaptiven Verhaltens kann z.B. durch parametrisierbare Architektur-Artefakte, also der Spezifikation des Änderungspotentials im Architektur-Modell erfolgen. Im Betrieb kann nun entweder bei Bedarf zwischen zuvor festgelegten Modi umgeschaltet, oder durch Online-Planung je nach Situation eine optimale Parametrisierung ermittelt werden.

Die in Abschnitt 1.3 beschriebene adaptive Architektur PROSA-X verfügt über eine spezielle Komponente (*Self-Manager*), die je nach Betriebszustand durch Rekonfiguration von Betriebsparametern Self*-Eigenschaften realisiert [SG07].

Für beide Ansätze sind Architektur-Modelle die Basis für die Planung und Durchführung der Änderungen. Gerade weil die Architektur das stabilste Element eines Software-Systems über den gesamten Lebenszyklus darstellen sollte, gelten Anpassungen an der Architektur aber als schwierig und kostspielig. Deshalb erfolgen Modifikationen am System häufig in nicht-optimaler Form, d.h. ohne die notwendige Evolution der Architektur, und führen gerade deshalb zur nachhaltigen „Schädigung“ (*Erosion*) der Architektur. Die so kontinuierlich anwachsende Entropie der Systeme führt dann unter anderem zu abnehmender Wartbarkeit und Performanz [Mas07]. Zudem werden durch diese „Wucherungen“ die Abgrenzungen des Systems zur Umgebung zunehmend unklar.

2.2 Architektur-Modellierung und -Rekonstruktion

Der Architektur-Entwurf ist das Bindeglied zwischen der Anforderungsanalyse und dem technischen Design, auf dessen Grundlage neben der korrekten Implementierung der geforderten Funktionalität zugleich Maßnahmen zur Qualitätssicherung in den Entwurfsprozess integrieren werden können [FGB+07, LMD+09, LMS+09].

2.2.1 Architektur-Modelle eingebetteter Systeme

Für Abstraktionskonzepte von Architektur-Spezifikationen, insbesondere zur strukturellen Dekomposition, existieren eine Vielzahl unterschiedlichster Modellierungsansätze und Formalismen [RH06]. Wir beziehen uns vor allem auf Component-Connector-Modelle [BCK03], also die explizite Darstellung grundlegender, häufig rein logischer System-Artefakte und den Abhängigkeiten zwischen diesen Elementen.

Eingebettete Systeme sind zunehmend Software-intensiv und realisieren Steuerungs- und Regelungsaufgaben durch Kommunikations-orientierte Integration verschiedenster, hochspezialisierter Komponenten auf häufig massiv verteilten und heterogen strukturierten Hardware-Plattformen. Standardisierte Schichtenarchitekturen erlauben eine flexible Verteilung von Software- und Hardware-Komponenten, sodass beliebige Architektur-Varianten bezüglich (Wieder-) Verwendung, Anordnung, Verteilung und Verknüpfung dieser Komponenten möglich sind und so großen Spielraum für Optimierungen bieten. Ein Problem

bei der Entwicklung dieser Systeme besteht u.a. im konzeptionellen „Bruch“ beim Übergang von der abstrakten Architektur hin zur möglichst effizienten, d.h. Hardware-nahen Implementierung der Einzelfunktionen. Für die gezielte Planung und Durchführung von Evolutionen langlebiger eingebetteter Systeme bilden Architektur-Modelle eine wichtige Entscheidungsgrundlage, da sie nicht nur Abhängigkeiten zwischen betroffenen Systemteilen aufzeigen, sondern auch durch Modularisierung gezielt die Austauschbarkeit von Teilkomponenten unterstützen.

Architektur-Entwürfe konzentrieren sich in der Regel auf die Spezifikation statischer Strukturen, jedoch ist dies für weitergehende Zwecke, wie z.B. für Systemanalysen zur Anpassung bzw. Rekonfiguration, oft unzureichend, sodass deshalb eine Kombination mit Verhaltensmodellen notwendig ist [RB02]. Um zukünftig beim Entwurf von System-Architekturen potenzielle Evolutionen bereits während der Entwicklung zu berücksichtigen, muss die geplante Variabilität des Systems explizit mitmodelliert werden. Derartige *evolutionsfähige Architekturen* können dann als Grundlage für die Planung und Durchführung von Evolutionsschritten und für die Dokumentation der Evolutionsgeschichte dienen und ermöglichen ein Variabilitätsmanagement vergleichbar mit den Konzepten der *Produktlinien*-Ansätze [CN07]. Durch vordefinierte Variationspunkte werden mögliche Freiheitsgrade für die Anpassbarkeit des Systems bereits als Teil des Architektur-Modells festgelegt. Je nach Änderungsszenario kann die Evolution dann durch Anpassung betroffener Artefakte erfolgen, und das in sämtlichen Phasen des Lebenszyklus. Hierfür sind Architektur-Metamodelle notwendig, die neben der Zuordnung von Variabilitätspunkten auch die Integration von zusätzlichem Wissen über das System erlauben. Auf diese Weise kann sowohl auf Änderungen funktionaler, als auch nicht-funktionaler Anforderungen zielgerichtet reagiert werden.

2.2.2 Architektur-Sichten eingebetteter Systeme

Die Definition von *Architektur-Sichten* trägt beim Entwurf von Architekturen eingebetteter Systeme maßgeblich zur Übersichtlichkeit und Komplexitätsbeherrschung bei. Ausgehend von der domänenspezifischen Anforderungsspezifikation (inkl. Qualitätskriterien) in der Applikationssicht können sowohl die hochspezialisierte, modularisierte Software in der Funktionsarchitektur-Sicht, als auch die anwendungsspezifische Hardware-Plattform (technische Sicht) zunächst getrennt betrachtet werden, und erst im letzten Schritt durch möglichst günstige Verteilung der Software-Komponenten auf vorhandene Recheneinheiten unter Beachtung des resultierenden Kommunikationsaufkommens flexibel in einer System-Architektur integriert werden. Analog zu [RB02] unterscheiden wir im Folgenden zudem zwischen Architektur-Sichten zur Spezifikations- und Laufzeit:

Konzeptionelle Architektur: Die Spezifikation beschreibt abstrakte strukturelle Einheiten des Systems mitsamt ihren Abhängigkeiten. Je nach Modellierungsansatz können konzeptionelle Architekturen unterschiedlichste Artefakte beinhalten. Beispielsweise kann die Struktur des Softwaresystems auf Grundlage einer Component-Connector-Abstraktion beschrieben und durch Schnittstellen- und Prozess-Definitionen verfeinert werden. Beim Entwurf adaptiver Systeme können zudem im Vorfeld gezielt Konfigurationsparameter eingeplant werden.

Konkrete Architektur: Zur Laufzeit repräsentieren die im System agierenden, operationellen Einheiten die aktuelle Architektur-Ausprägung. Dazu gehören Artefakte wie laufende Prozesse (Tasks), deren Kommunikation untereinander sowie Werte von Konfigurationsparametern. Diese Instanzen der Elemente der konzeptionellen Architektur variieren je nach Zustand des Systems im betrachteten Zeitpunkt.

Eine Divergenz zwischen konzeptioneller und konkreter Architektur ist in der Regel unvermeidbar. Spätestens im Betrieb ergeben sich Erosions-bedingte Inkonsistenzen, z.B. durch unzureichend dokumentierte Evolutionen. Darüber hinaus ist insbesondere für Altsysteme in der Regel keine oder eine nur unvollständige Spezifikation der Architektur vorhanden. Für eine strukturierte Planung von Architektur-Evolutionen ist somit eine *Rekonstruktion* der konzeptionellen Architektur auf Grundlage der durch die aktuelle Systemausprägung gegebenen konkreten Architektur notwendig.

Auch für die Architektur-Evaluation können passende Architektur-Sichten definiert werden, die für die jeweiligen Bewertungstechniken relevanten Artefakte enthalten. Für die Modellierung evolvierender Architekturen können zudem Sichten definiert werden, die eine explizite Spezifikation von Variabilitätsparametern unterstützen. Für die Bewertung von Echtzeiteigenschaften in PROSA-X ist z.B. die Verteilung und Kommunikation von Prozessen im System von entscheidender Bedeutung, wobei eine Reihe von Randbedingungen zu beachten sind. Eine Sicht zur Planung und Bewertung von Prozessmigrationen repräsentiert diese Verteilungs- und Kommunikationsstruktur durch eine entsprechende Graphendarstellung [SGM09].

2.2.3 Architektur-Rekonstruktion

Die Kenntnis der Architektur eines Systems ist Grundvoraussetzung für ein tiefer gehendes Verständnis der internen Systemstrukturen zur Analyse und Evolutionsplanung. Für langlebige Systeme mit hohen Anteilen an Legacy-Komponenten liegen allerdings häufig nur unvollständige bzw. inkonsistente Architektur-Modelle vor, in denen Änderungen am System häufig nicht adäquat dokumentiert wurden. Zudem beinhalten Spezifikationen oftmals nur für die Qualitätsbewertung unzureichende statische Aspekte der Systemstruktur. Es gibt eine Vielzahl von Ansätzen, um für ein bestehendes System Architektur-Eigenschaften zu rekonstruieren, über die nachfolgend ein kurzer Überblick gegeben werden soll.

Rekonstruktion: Durch „heuristisches“ Reverse Engineering wird versucht, höhere Abstraktionsebenenen, wie z.B. Component-Connector-Strukturen und andere Pattern, aus dem Quellcode eines bestehenden Software-Systems ohne adäquate Spezifikation zu ermitteln. Eine wesentliche Problematik besteht hier im „mismatch“ zwischen abstrakten Architektur-Konzepten und den Artefakten von Programmiersprachen, also der Definition eines geeigneten *correspondence model* zwischen Code und Architektur-Modell. Bei der Hardware-nahen Programmierung eingebetteter Systeme gestaltet sich die Identifikation von High-Level-Komponenten als besonders schwierig. Ansätze zum Architektur-Recovery sind in der Regel nur (semi-) automatisierbar, bedürfen also geführter Entscheidungen von Expertenseite.

Extraktion: Eine Erweiterung der Rekonstruktionsansätze basiert auf der Einbettung von Architektur-Modellen als *Metawissen* in den Programmcode, wodurch eine präzisere Identifikation und Zuordnung von Architektur-Artefakten möglich wird. Speziell objektorientierte Programmierparadigmen weisen eine enge konzeptionelle Verwandtschaft zu Komponenten-orientierten Architektur-Modellen auf. Die Einbettung von Verhaltensmodellen werden z.B.. [BSG09] und [SVB+03] beschrieben. Weiterhin können hier Programmierumgebungen mit Reflection-Funktionen angeführt werden, von denen zur Laufzeit Informationen über die eigene Programmstruktur abgefragt werden können.

Monitoring: Für eine beliebig detaillierte Ermittlung von dynamischen Informationen können schließlich eigene Observer-Komponenten innerhalb des Systems zum *Tracking* von Laufzeitinformationen, z.B. dem Erstellen von Last- und Kommunikationsprofilen, verwendet werden, was insbesondere ein wesentlicher Bestandteil adaptiver Systeme ist. Auf dieser Grundlage können auch entsprechend aussagekräftige Metriken zur Validierung vorhergehender Qualitätsbewertungen definiert werden.

Rekonstruktion und Extraktion eignen sich vor allem zur Ermittlung statischer Aspekte und Strukturen. In Kombination mit Monitoring-Ansätzen können diese um dynamische Eigenschaften ergänzt werden. In erster Näherung ergeben die genannten Verfahren zunächst konkrete Architektur-Sichten, der Übergang zu einem konzeptionellen Modell bedarf anschließend weiterer Abstraktionsschritte.

2.3 Architektur-Koevolution unter Erhalt von Qualitätseigenschaften

Die Integration der beschriebenen Verfahren erfordert Vorgehensmodelle, die den gesamten Lebenszyklus eingebetteter Systeme vom initialen Entwurf, der Implementierung und Inbetriebnahme, bis zur Wartung und Anpassung im Betrieb, umfassen. Die Konsistenzsicherung zwischen Anforderungsspezifikation, Architektur-Modell und Systemimplementierung verlangt einen umfassenden Ansatz zur Koevolution, d.h. der fortgesetzten Synchronisation des laufenden Systems und der Spezifikation. Neben der Planung und Dokumentation möglicher Evolutionen kann so gleichzeitig die Sicherung der geforderten Qualitätseigenschaften auf Grundlage der aktuellen Architektur-Ausprägung erfolgen. Der prinzipielle Aufbau eines entsprechend zyklischen Vorgehensmodells ist in Abbildung 3 skizziert. Das dargestellte Roundtrip Engineering ist notwendig, um die beiderseitige Synchronisation zwischen Spezifikation und laufendem System sicherzustellen, da (1) Erosionen im laufenden System Anpassungen an der Architektur-Spezifikation erzwingen, und (2) Änderungen der Anforderungen Anpassungen am laufenden System erzwingen. In beiden Fällen bildet die Architektur-Spezifikation die „Brücke“ zwischen Anforderungen und Implementierung: (1) als Entwurf der initialen (konzeptionellen) Systemstruktur, (2) zur modellbasierten Repräsentation der wesentlichen Aspekte des (konkreten) Systemzustandes, und damit (3) als Grundlage für die Planung, Durchführung und Dokumentation von Anpassungen und Adaptionen. Sowohl beim ersten Entwurf, als auch bei der fortgesetzten Evolution der System-Architektur kann die Sicherstellung geforderter Qualitätseigenschaften durch wiederholte Architektur-Evaluation erfolgen.

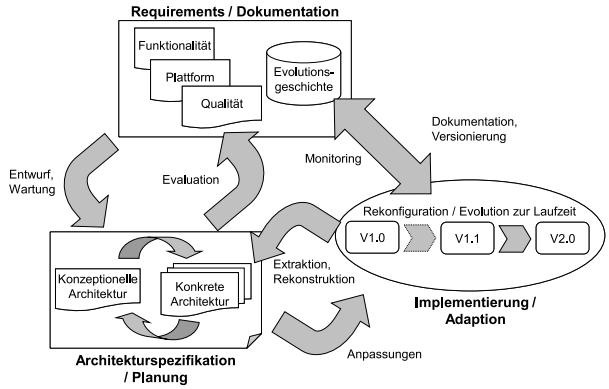

Abbildung 3: Koevolution im gesamten Lebenszyklus

Grundlage bzw. Auslöser von Anpassungen sind sich stetig ändernde Anforderungen an das System bezüglich: (1) der Funktionalität, (2) der (technischen) Plattform, wozu prinzipiell auch die Systemumgebung gezählt werden kann sowie (3) Qualitätsanforderungen. Die Planung dieser Anpassungen auf Grundlage evolvierender Architekturen kann sowohl anhand der konkreten, als auch der konzeptionellen Architektur erfolgen. Erstere kann insbesondere für „leichtgewichtige“ Adaptionen, wie z.B. der Prozessmigration von Bedeutung sein, wohingegegen das konzeptionelle Architektur-Modell für die Planung von Umstrukturierungen größeren Umfangs, wie z.B. der Integration neuer Komponenten, dienen kann. In diesem Zusammenhang kann bereits im Vorfeld sowie Evolutions-begleitend durch das Qualitätskriterium „Modifzierbarkeit“ [BCK03] die Architektur auf potenzielle Anpassungsszenarien hin ausgelegt und bewertet werden. Neben der *globalen* Koevolution zwischen Anforderungen, Architektur-Modellen und der Implementierung, besteht auch die Notwendigkeit, die Konsistenz von konkreter und konzeptioneller Architektur sicherzustellen. Die konkrete Architektur ergibt sich als *Snapshot* aus Systemzustand und Laufzeitartefakten im betrachteten Betriebszeitpunkt und kann, wie in Abschnitt 2.2.3 beschriebenen, ermittelt werden. Ergibt sich für die so ermittelte konkrete Architektur zu einem bestimmten Zeitpunkt, dass sie z.B. durch eine Folge von Adaptionen strukturell zu weit von der konzeptionellen Architektur entfernt ist und nicht mehr als dessen Ausprägung gelten kann, ist auch eine Evolution, d.h. Rekonstruktion der konzeptionellen Architektur notwendig. Umgekehrt führt die Planung von Evolutionen auf Grundlage der konzeptionellen Architektur zu einer Diskrepanz mit der konkreten Architektur, die gerade die für diese Anpassungen notwendigen Änderungen im laufenden System impliziert. In beiden Fällen können die aktualisierten Architektur-Modelle zur erneuten Architektur-Evaluation und somit fortlaufenden Qualitätssicherung verwendet werden, bzw. als Entscheidungsgrundlage in den Planungsprozess von Anpassungen und Adaptionen einbezogen und durch Monitoring-Ansätze ergänzt werden. Die laufenden Anpassungen der Implementierung können je nach *Grad* der hieraus entstehenden Evolution zu unterschiedlich stark ausgeprägten Erosionen der konkreten, und schließlich auch der konzeptionellen Architektur führen. Diese „Versionssprünge“ stellen die wesentlichen Bezugspunkte bei der Dokumentation der Evolutionsgeschichte für das Änderungs-, Anforderungs- und

Qualitätsmanagement langlebiger Systeme dar. Somit sind Architektur-Entscheidungen als Bestandteil der Versionsverwaltung und des *Requirements Tracing* mitsamt der Evaluationsstruktur für die Qualitätskriterien nachverfolgbar und können auch als Anhaltspunkte für spätere Anpassungen erneut herangezogen werden. Vor der Durchführung von Änderungen am System sind neben der Überprüfung von Qualitätskriterien auch entsprechende Techniken zum Erhalt der funktionalen Korrektheit einzusetzen, wie beispielsweise die Kombination kontinuierlicher Refactorings und Tests aus dem Bereich der agilen Methoden [RH06]. Für eine nähere Diskussion der in diesem Bereich bestehenden Probleme und Herausforderungen verweisen wir u.a. auf [RB02].

Evolutionsfähige Architekturen: Für die Planung, Durchführung und Dokumentation von Lebenszyklus-belgeitenden Evolutionen auf Grundlage von Architektur-Modellen bedarf es entsprechender Ansätze, die an dieser Stelle kurz skizziert werden sollen. Wie bereits in [EGG+09] diskutiert wurde, müssen potenzielle Evolutionen integraler Bestandteil der Systemspezifikation werden. Wissen über das System in Form von Konfigurationsparametern für Anpassungen, Annotationen zur Metrikauswertung sowie Instrumentierungen für das Monitoring des Laufzeitverhaltens müssen hierfür als fester Besatandteil des Metamodells vorgesehen werden. Je nach Art und Umfeld des Systems können so entsprechende Profile für Anpassungsklassen definiert werden. Das in [Par94] propagierte *Design for Change* basiert beispielsweise auf der Definition von Änderungsklassen, für die je nach Eintrittswahrscheinlichkeit einer Änderung eine Auswirkungsanalyse für betroffene Artefakte und deren Isolation ermöglicht wird. Die eigentliche „technische“ Durchführung von Anpassungen ist von vielerlei Aspekten abhängig [RH06] und kann sowohl durch eine Abfolge kleiner Einzelschritte, als auch in einen „Big Bang“ erfolgen. So basieren beispielsweise Architektur-Refactorings auf konsistenzerhaltenden Transformationsvorschriften zwischen Artefakten des Architektur-Metamodells und entsprechender Pattern in der Implementierung. Bei der Migration ganzer (Teil-) Systeme sind hingegen vor allem Faktoren wie verwendete Schichten-Architekturen und Virtualisierungs-Komponenten von Bedeutung.

Qualitätssicherung von Evolutionen: Obwohl für die Qualitätsbewertung bestehender evolvierender Systeme im Gegensatz zur Entwurfs-begleitenden Architektur-Evaluation prinzipiell sämtliche Implementierungs- und Laufzeitdetails verfügbar sind, bietet sich auch hier die Verwendung von schwerpunktmäßig auf Architektur-Abstraktionen basierender Bewertungstechniken an, um die Komplexität nicht zu groß werden zu lassen. Bei der Weiterverwendung einer während der Entwurfsphase des Systems für die Qualitätsanforderungen definierten Evaluationsstruktur aus Abbildung 1 zur fortlaufenden Überprüfung der Qualität evolvierender Systeme ergeben sich folgende Fragestellungen:

1. Welche Qualitätskriterien sind im Betrieb im Vergleich zur Entwurfsphase relevant? Kriterien, die harte Randbedingungen für den korrekten Betrieb des Systems darstellen, wie z.B. Sicherheit und Echtzeit, sind im gesamten Lebenszyklus einzuhalten. Kriterien, die schwerpunktmäßig für Entscheidungen während des Entwurfs von Bedeutung sind, wie z.B. Kosten, sind bei der Bewertung von Architektur-Evolutionen hingegen zweitrangig. Ein Spezialfall bilden Kriterien wie Modifzierbarkeit, die nicht nur fortlaufend evaluiert werden sollten, sondern insbesondere durch neue Modifikationsszenarien, die zum Zeitpunkt des Entwurfs noch gar nicht bekannt waren, erweitert werden können. Somit werden

auch die Bestandteile der Evaluationsstruktur selbst eine stetige Evolution erfahren.

2. Welche Anpassungen werden an den Bewertungstechniken vorgenommen? Neben der konzeptionellen Architektur können nun auch Artefakte der konkreten Architektur sowie weitere Anreicherungen um Laufzeitinformationen durch Monitoring für die Präzisierung der vormals nur auf Grundlage des Architektur-Entwurfs definierten Bewertungstechniken herangezogen werden. Zudem können Erfahrungswerte aus der Evolutionsgeschichte in die Bewertung einfließen.

Die methodische Integration der Architektur-Evaluation gemäß Abbildung 3 kann auf zwei Arten erfolgen: (1) reaktiv, d.h. schon beim Entwurf des Systems werden für bestimmte Betriebsmodi feste Anpassungspläne definiert und vor der Anwendung durch Architektur-Evaluation „abgesichert“, um so K.O.-Kriterien auszuschließen, oder (2) proaktiv, d.h. die Planung von Anpassungen erfolgt „online“ und die Qualitätsattribute gehen als Planungsparameter in die Entscheidung ein, um einen möglichst guten Trade-off zu ermitteln. Diese Planungen können als Reaktion auf Änderungen von (funktionalen) Anforderungen oder Laufzeitbedingungen erfolgen sowie zur Optimierung der Qualitätseigenschaften selbst. Der enge Zusammenhang zwischen Evolution und Evaluation wird vor allem an Kriterien wie Modifzierbarkeit deutlich: Diese sind einerseits Gegenstand der Bewertungsszenarien und zugleich Ziel der Anpassungen.

Fallstudie: Rekonfiguration unter Erhalt von Echtzeiteigenschaften

Durch das Einsatzgebiet von PROSA-X werden hohe Anforderungen an die Sicherheit, Verfügbarkeit, Performanz, etc. gestellt. Um insbesondere die Erfüllung harter Echtzeitanforderungen sicherzustellen, sind unter Umständen Anpassungen im laufenden Betrieb notwendig, z.B. bei Änderungen an der Ausführungsplattform (Knotenausfall) oder der Betriebslast (Prozessanzahl). In diesen Fällen kann der *Self-Manager* zur Laufzeit eine neue Konfiguration durch Umverteilung von Prozessen auf Netzwerkknoten ermitteln, um den Erhalt der Echtzeiteigenschaften weiterhin zu gewährleisten. Ein exemplarisches Adaptionsszenario ist in Abbildung 4 dargestellt. Die Planung der Umverteilung erfolgt auf Grundlage einer extrahierten, konkreten Architektur-Sicht, bestehend aus einem durch Monitoring ermittelten Kommunikationsgraphen der laufenden Prozesse. Durch eine Kombination verschiedener Heuristiken zur Graphpartitionierung wird eine möglichst ausgewogene Neuverteilung der Prozesse auf vorhandene Knoten geplant (im Beispiel drei Kontroll-PCs), d.h. (1) eine gleichmäßige Auslastung der einzelnen Prozessorknoten, und (2) eine möglichst geringe Buslast angestrebt. Die Adaption der konkreten Architektur erfolgt durch Prozessmigration im laufenden System. Gleichzeitig kann das angepasste Architektur-Modell zur Prüfung der übrigen Qualitätskriterien herangezogen werden, z.B. die Verletzung von Sicherheitsanforderungen, falls durch die Migration die notwendige Isolation zweier Prozes-

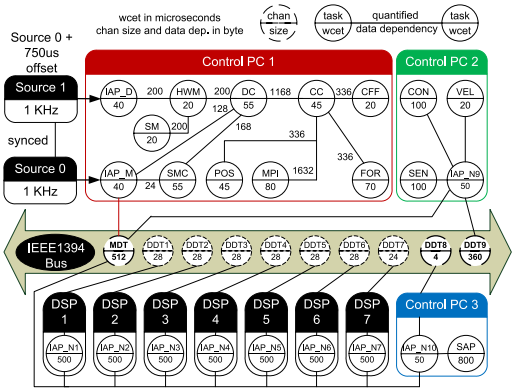

Abbildung 4: Prozess-Migration in PROSA-X

se mit unterschiedlichem Sicherheitslevel aufgehoben würde. Schließlich kann die resultierende Evaluationsstruktur zur Dokumentation der vollzogenen Evolution als Bestandteil des Qualitäts- und Versionsmanagements dienen.

Literatur

[BCK03] L. Bass, P. Clements und R. Kazman. *Software Architecture in Practice*. Addison-Wesley Longman, Amsterdam, 2. Auflage, 2003.

[BSG09] M. Balz, M. Striewe und M. Goedicke. Embedding Behavioral Models into Object-Oriented Source Code. In *Software Engineering*, Seiten 51–62, 2009.

[CN07] P. Clements und L. M. Northrop. *Software Product Lines: Practices and Patterns*. Addison Wesley, 6. Auflage, 2007.

[EGG+09] G. Engels, M. Goedicke, U. Goltz, A. Rausch und R. Reussner. Design for Future - Legacy-Probleme von morgen vermeidbar? In *Informatik Spektrum*. Springer, 2009.

[FGB+07] B. Florentz, U. Goltz, J. Braam, R. Ernst und T. Saul. Architekturevaluation: Qualitätssicherung in frühen Entwicklungsphasen. In *8. Symposium AAET 2007*, Seiten 238 – 248, 2007.

[Leh96] M. Lehman. Laws of Software Evolution revisited. In *Software Process Technology*, Seiten 108–124. 1996.

[LMD+09] M. Lochau, T. Müller, S. Detering, U. Goltz und T. Form. Architektur-Evaluation von AUTOSAR-Systemen: Adaption und Integration. In *Tagungsband des 1. Elektronik automotive congress, München*, 2009.

[LMS+09] M. Lochau, T. Müller, J. Steiner, U. Goltz und T. Form. To appear: Optimierung von AUTOSAR-Systemen durch automatisierte Architektur-Evaluation. In *VDI-Berichte, 14. Internationale Konferenz Elektronik im Kraftfahrzeug*, 2009.

[Mas07] D. Masak. *Legacysoftware: Das lange Leben der Altsysteme*. Springer, 2007.

[Par94] D. L. Parnas. Software Aging. In *ICSE '94: Proceedings of the 16th international conference on Software engineering*, Seiten 279–287, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[RB02] A. Rausch und M. Broy. Evolutionary Development of Software Architectures. In *Technology for Evolutionary Software Development, a Symposium organised by NATO's Research & Technology Organization (RTO)*, 2002.

[RH06] R. Reussner und W. Hasselbring. *Handbuch der Software-Architektur*. Dpunkt, 2006.

[SG07] J. Steiner und U. Goltz. Engineering Self-Management into Legacy Systems. In *Proceeding for 3rd International Conference on Self-Organization and Autonomous Systems in Computing and Communications (SOAS 2007)*, Jgg. 2 of *System and Information Science Notes*, Seiten 114–117, 2007.

[SGM09] J. Steiner, U. Goltz und J. Maaß. Dynamische Verteilung von Steuerungskomponenten unter Erhalt von Echtzeiteigenschaften. In *6. Paderborner Workshop Entwurf mechatronischer Systeme*, 2009.

[SVB+03] R. Sekar, V. Venkatakrishnan, S. Basu, S. Bhatkar und D. DuVarney. Model-carrying code: A practical approach for safe execution of untrusted applications, 2003.