

Werkzeugunterstützung für die Entwicklung selbstadaptiver Anwendungen

Andreas Rasche
Fachgebiet Betriebssysteme und Middleware
Hasso-Plattner-Institut an der Universität Potsdam
Prof.-Dr.-Helmert-Str. 2-3 14482 Potsdam
andreas.rasche@hpi.uni-potsdam.de

Überblick

Moderne Kommunikationsinfrastrukturen, die Vielfalt mobiler Geräte sowie die zunehmende Komplexität verteilter Systeme erfordern die Entwicklung von Anwendungen, welche sich an wechselnde Umgebungsbedingungen ohne manuellen Eingriff anpassen können.

Bei der Entwicklung solcher selbstadaptiver Anwendungen fällt dabei auf, dass viele Teilaspekte separiert von der eigentlichen Anwendungsfunktionalität implementiert werden können. Diese Teilaspekte umfassen die Messung der Umgebungseigenschaften (Monitoring), Auswertung von Anpassungsprofilen und Mechanismen für die Umschaltung zwischen alternativen Anwendungskonfigurationen während der Laufzeit (dynamische Rekonfiguration). Die Entwicklung von alternativen Konfigurationen ist von zentraler Bedeutung für die Selbstadaptivität. Sie kann durch grafische Werkzeuge und die automatisierte Integration von Architekturmustern unterstützt werden.

Im Rahmen des *Adapt.Net* Projekts entstanden am Hasso-Plattner-Institut Werkzeuge zur Erstellung selbstadaptiver Anwendungen inklusive einer Laufzeitplattform zur Ausführung dynamisch rekonfigurierbarer Anwendungen und eine Reihe wiederverwendbarer Monitor-Komponenten zur Messung von ausgewählten Umgebungseigenschaften. In diesem Beitrag sollen die entwickelten Werkzeuge und Mechanismen vorgestellt werden.

Einleitung

Die Entwicklung von selbstadaptiven Anwendungen umfasst im Wesentlichen drei Teilbereiche: die Beobachtung von Umgebungs- und Anwendungseigenschaften (Monitoring), der Definition von Anpassungsprofilen sowie Mechanismen, auf Änderungen der gemessenen Eigenschaften zu reagieren.

Die **Dynamische Rekonfiguration** Komponenten-basierter Anwendungen ist ein leistungsfähiger Mechanismus um auf Änderungen von Umgebungseigenschaften während der Laufzeit zu reagieren. Mit Hilfe dynamischer Rekonfiguration kann eine Anwendungskonfiguration durch die Justierung von Komponentenparametern, Hinzufügen, Updaten oder Entfernen einzelner Komponenten sowie der Migration laufender Komponenten auf andere Rechner ohne Neustart der Anwendung manipuliert werden. Eine **Anwendungskonfiguration** umfasst dabei alle beteiligten Komponenten, deren Parametrierung, Verbindungen zwischen diesen Komponenten und die Zuordnung auf Rechner in einem verteilten System. Schnittstellen (Ports) bestimmen dabei die Endpunkte von Verbindungen an den Komponenten.

Die Änderung einer Konfiguration während der Laufzeit stellt eine Herausforderung dar, da nicht zu jedem Zeitpunkt die Überführung in einen konsistenten Zustand gewährleistet werden kann. Die Anwendung muss deshalb zunächst in einen rekonfigu-

rierbaren Zustand überführt werden, um Konfigurationsänderungen durchführen zu können. Hierzu müssen entsprechende Synchronisationsmechanismen implementiert werden, welche Anwendungsfunktionalität und Rekonfigurationsanforderungen serialisieren. Zusätzlich müssen gegebenenfalls Mechanismen für den Zustandstransfer von Anwendungskomponenten im Falle einer Migration bzw. eines Updates bereitgestellt werden.

Die **Monitoring**-Infrastruktur in *Adapt.Net* umfasst die Überwachung von Komponentenparametern, den Zustand von Komponenten im Allgemeinen um Komponentenausfälle erkennen zu können, sowie separate Umgebungsmonitore, die ausgewählte Umgebungsparameter (z.B. Netzwerkbandbreite, verfügbare CPU-Zyklen, Batteriezustand) überwachen können und wiederverwendbar sind.

Schlussendlich stellen **Anpassungsprofile** die Verbindung zwischen gemessenen System-/Umgebungseigenschaften und den passenden Anwendungskonfigurationen her. Anpassungsprofile können dabei statisch über die Angabe von Bereichen ($10 < \text{Bandbreite} < 1000$) auf Konfigurationen abbilden, oder unter Benutzung von *Adapt.Net*-Monitoring-Schnittstellen die Zuordnung programmatisch realisieren. Diese Möglichkeit kann benutzt werden um komplexe Regelungsmechanismen zu integrieren.

CoFRA - Die Adapt.Net Laufzeitinfrastruktur

Adapt.Net ermöglicht die dynamische Rekonfiguration komponenten-basierter Anwendungen. Der verwendete Rekonfigurationsalgorithmus basiert auf der Blockierung betroffener Verbindungen zwischen den Komponenten. Die Einführung eines Transaktionskonzepts ermöglicht die programmatische Bündelung von einzelnen Interaktionen über diese Verbindungen. Eine Verbindung zwischen 2 Komponenten wird blockiert, indem der Beginn neuer Transaktionen über diese Verbindung verhindert wird und auf die Beendigung aktiver Transaktionen über diese Verbindung gewartet wird. Sind alle betroffenen Verbindungen blockiert, – die Anwendung in einem rekonfigurierbaren Zustand – ist es möglich Änderungen an der Konfiguration vorzunehmen ohne die Konsistenz der Anwendung zu verletzen. Das *Adapt.Net*-Framework stellt entsprechende Schnittstellen für die Markierung von Transaktionen in den Anwendungskomponenten bereit. Ausführliche Details zum verwendeten Algorithmus finden sich in [Ras05].

Anwendungskonfigurationen werden in einer XML-basierten Beschreibungssprache durch Aufzählung von Komponenten, deren Parametern, Konnektoren sowie einer Rechnerzuordnung spezifiziert. *CoFRA* instanziiert entsprechend dieser Beschreibung die Anwendungskomponenten. Jede Komponente muss dabei die Schnittstelle *IConfigure* implementieren, welche Methoden zum Setzen von Komponentenparametern, Blockieren von Verbindungen, dem Verbinden von Komponenten sowie zum Starten der Komponentenverarbeitung.

Abbildung 1 zeigt die *Adapt.Net* Laufzeitinfrastruktur *CoFRA*. Anhand der Konfigurationsbeschreibung werden Anwendungskomponenten instanziiert und mit Hilfe eines generischen **Konfigurationsproxy** (CoCo) über die Schnittstelle *IConfigure* konfiguriert. Eine Komponente kann als einfaches Objekt mit Hilfe des *new*-Operators erzeugt werden oder aber es wird ein Betriebssystemprozess gestartet welcher die neue Komponente beinhaltet (Dieses Feature kann benutzt werden um Komponentenausfälle zu erkennen indem das umgebende Betriebssystem-Prozessobjekt überwacht wird). Um die Aus-

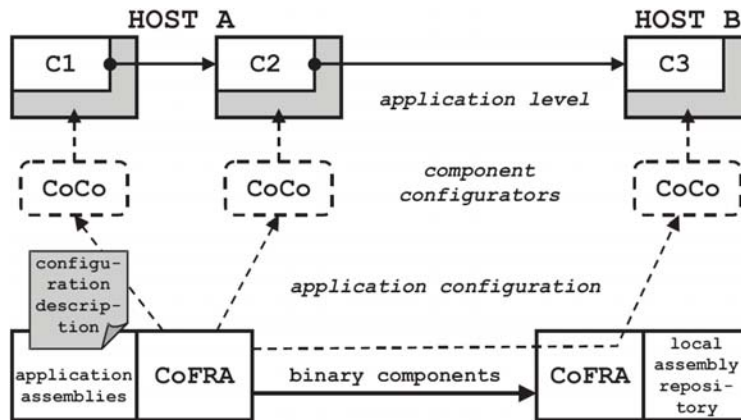


Abbildung 1: Configuration Framework Infrastruktur

führung heterogener Anwendungen zu unterstützen (CORBA, Java, .NET Komponenten) kapselt der Konfigurationsproxy plattformspezifische Details. Er stellt Funktionen zum Laden, Entladen und Abfragen des Zustands von Komponenten bereit.

CoFRA wurde auf der Microsoft .NET Plattform entwickelt. Softwarekomponenten in .NET, *Assemblies*, enthalten neben dem funktionalen Anwendungscode auch Metadaten, welche Abhängigkeiten zu anderen *Assemblies* und zusätzliche Informationen über den Code enthalten. *CoFRA* ist in der Lage anhand dieser Informationen *Assemblies* aus einem zentralen Repository auf einen Zielrechner zu kopieren und installieren (**Deployment**) indem Abhängigkeiten zwischen *Assemblies* analysiert werden und über einen *CoFRA*-Dienst auf den Zielrechner übertragen werden, wobei entsprechende Sicherheitsmechanismen der .NET-Plattform benutzt werden.

Während des Starts einer Anwendung initialisiert *CoFRA* die Monitoring-Komponenten anhand eines XML-basierten Profils. Stellen diese während der Laufzeit der Anwendung eine signifikante Änderung der Umgebungseigenschaften fest, wird eine neue Konfiguration bestimmt und die Rekonfiguration ausgelöst. *CoFRA* lädt ggf. die neue Konfiguration, ermittelt hinzugekommene Komponenten und startet diese zunächst. Im nächsten Schritt werden Verbindungen ermittelt die anschließend über die Konfigurationsschnittstellen der Komponenten blockiert werden. Im dritten Schritt werden die Komponenten entsprechend der neuen Konfiguration verbunden und Komponentenparameter justiert. Abschließend wird die Verarbeitung der Komponenten wieder gestartet, womit die neue Anwendungskonfiguration aktiviert ist.

Konfigurationen können dabei vor der Laufzeit der einer Anwendung definiert und installiert werden, wenn entsprechende Umgebungssituationen vorab bekannt sind, können aber auch nachträglich während der Laufzeit der Anwendung entwickelt und installiert werden.

Adapt.Net unterstützt das Updaten einzelner Komponenten während der Laufzeit. Das heißt, dass Fehler in Komponenten behoben bzw. Komponentenimplementationen an neue – während der Installationszeit unbekannt - Umgebungseigenschaften angepasst werden können. Diese Funktionalität sowie auch die Migration von Komponenten machen einen Zustandstransfer zwischen neuen und alten Komponenten unumgänglich. In *Adapt.Net* wurde deshalb ein Mechanismus integriert, welcher durch Traversieren des Objektgraphen einer Komponente sowie einer Element-weisen Kopie der primitiven Elemente (string,int,...) der beteiligten Objekte, die Zustandsübertragung realisiert. Der implementierte Algorithmus verfolgt rekursiv alle Referenzelemente ausgehend vom Wurzelobjekt der Komponente. Dabei muss beachtet werden, dass bereits untersuchte Knoten markiert werden und zusätzlich auch Referenzen in *Delegates* (Funktionspointer

in .NET) betrachtet werden müssen, da diese potentiell weitere Referenzen enthalten. Beim Komponenteupdate müssen evtl. auch Transformationen auf dem Zustand durchgeführt werden, welche mit Hilfe so genannter Transformations-Konstrukturen unterstützt werden. In diesen können Komponentenprogrammierer den Zustandstransfer manuell implementieren. Details hierzu finden sich in [Ras05/2]

Werkzeugunterstützung

Im Rahmen des *Adapt.Net*-Projekts wurden Werkzeuge entwickelt um Anwendungen für die beschriebene Laufzeitinfrastruktur in wenigen Schritten entwickeln zu können. Abbildung 2 zeigt den „*Adaptation Profile Creator*“. Im Navigationsmenü auf der linken Seite können Anwendungskonfigurationen, Monitorkomponenten und Anpassungsprofile verwaltet werden. Die rechte Seite zeigt eine Anwendungskonfiguration mit 6 Komponenten.

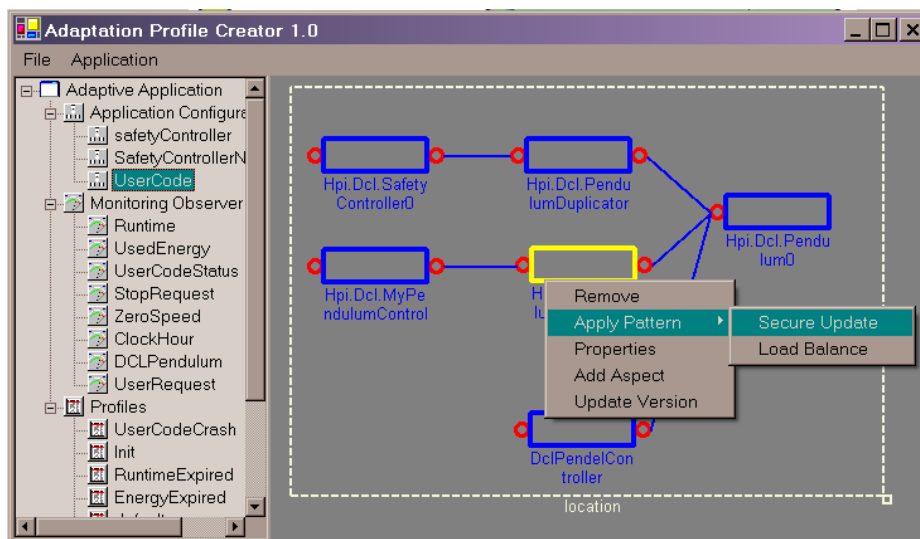


Abbildung 2 Grafische Erstellung alternativer Anwendungskonfigurationen

Der Anwendungsentwickler kann neue Komponenten per *Drag&Drop* hinzufügen. Wie schon zuvor erwähnt muss jede Komponente die Konfigurationsschnittstelle *IConfigure* unterstützen. Im Rahmen des *Adapt.Net* Projekts wurden Werkzeuge entwickelt, um diese **Schnittstellen** automatisch zu **generieren**. Bei der Entwicklung selbstadaptiver Anwendungen kann dabei leicht auf existierende Komponenten zurückgegriffen werden, da die Generierungswerkzeuge auf Binär-Ebene arbeiten. Die .NET Plattform unterstützt die *Assembly*-übergreifende Vererbung von Klassen. Beim Hinzufügen funktionaler Komponenten in den Konfigurationseditor, wird eine Ableitung der Hauptklasse generiert, welche zusätzlich das Interface *IConfigure* implementiert. Mittels Reflektion kommuniziert der hinzugefügte konfigurationsspezifische Code mit der Originalklasse. Durch die Manipulation von Elementen des Hauptobjekts werden Komponentenparameter und Verbindungen konfiguriert. Die Code-Generierung wurde mit Werkzeugen der Aspekt-orientierten Programmierung (*Loom.Net* [Sch02]) realisiert. Der Konfigurationsaspekt wird dabei mit dem funktionalen Anwendungscode mit Hilfe eines Aspektwebers verknüpft. Abbildung 3 zeigt die Verknüpfung von Anwendungscode und Konfigurationslogik.

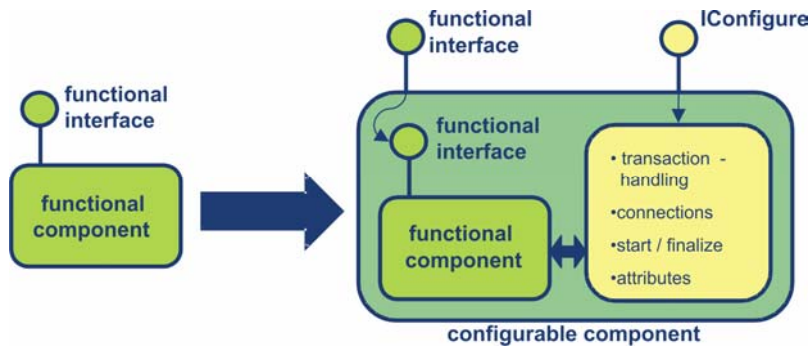


Abbildung 3: Generierung konfigurierbarer Komponenten

Komponentenentwickler können mit Hilfe von .NET Attributen, die eine Möglichkeit zur Annotierung von Code in .NET darstellen die Zusammenarbeit mit dem Konfigurationswerkzeug erleichtern. Mit dem *Property*-Attribut können Komponentenparameter markiert werden und mit Initialwerten versehen werden. Mit dem *Connection*-Attribut werden Verbindungsports von Komponenten markiert die Ausgangspunkt für Konnektoren darstellen. Dies ist im folgenden Code-Ausschnitt dargestellt.

```
public class Filter
{
    [Property default="75"] // Configuration Property Hook
    private double compression;
    [Connection] // Configuration Connector Hook
    protected PictureBox viewer;

    ...

    void SendPicture(byte[] picture)
    {
        // process data
        Viewer.SendPicture(picture);
    }
}
```

Ein weiterer wichtiger Aspekt bei der Erstellung von Anwendungskonfigurationen ist die Unterstützung vordefinierter **Architekturmuster**. Bei der Entwicklung alternativer Anwendungskonfigurationen treten häufig wiederkehrende Architekturmuster auf. So werden typischerweise bei verringerter Bandbreite zwischen Server und mobilem Gerät Datenkompressionskomponenten in den Kommunikationsfluss eingebracht. Ein weiteres Beispiel ist, dass bei der Berechnung komplexer Funktionen oft die Verteilung auf mehrere Rechner von Vorteil ist - dabei werden spezielle Lastverteilungs-Komponenten benötigt. *Adapt.Net* unterstützt eine Reihe solcher Architekturmuster um die Entwicklung alternativer Konfigurationen erleichtern. Dazu gehören:

- *SmartConnector*: Erkennung abgebrochener Verbindungen
- *LoadBalancer*: Lastverteilung auf multiple Rechner
- *Secure Update*: Einspielen neuer Version + Rollback im Fehlerfall
- *Voter Pattern*: Erhöhung der Zuverlässigkeit durch redundante Komponenten
- *Filter-/Compression-/Encryption Pattern*: Datenmanipulation
- *Simplex Pattern*: Analytische Redundanz – Einsatz im Distributed Control Lab

Der Anwendungsentwickler kann mit Hilfe von Werkzeugen die Codegenerierung dieser speziellen Komponenten veranlassen und so sehr schnell alternative Konfigurationen der Anwendung erstellen.

Monitoring

In vielen Fällen bietet sich bei der Entwicklung selbstadaptiver Anwendungen die Erstellung von Alternativkonfigurationen bezüglich bestimmter Einsatzkategorien an. Für diese Fälle ist es z.B. ausreichend zu unterscheiden, ob eine Kommunikationsverbindung über ein 100 MBit/s-Netzwerk oder eine GSM-Verbindung mit 56 KBit/s besteht. Für die Kategorie-basierte Adaptivität ist die Definition von Profilen möglich, welche die entsprechenden Einsatzsituation anhand von Parameterbereichen definieren. Mit Hilfe von Umgebungsmonitoren kann eine Datenvorverarbeitung erfolgen, welche die Klassifizierung von Umgebungseigenschaften im Voraus durchführen. In den Profilen müssen nur noch die Kategorien Alternativkonfiguration zugeordnet werden.

Für eine Reihe von Anwendungen ist es aber nötig programmatisch Verbindungen zwischen gemessenen Umgebungseigenschaften und Anwendungskonfigurationen zu definieren. *Adapt.Net* bietet die Möglichkeit Anpassungsprofile in Form von Anwendungslogik zu spezifizieren und dabei auf bereitgestellte Bibliotheksfunktionen zurückzugreifen. Dies erlaubt die Benutzung der *Adapt.Net* Monitoring-Infrastruktur und den Zugriff auf eine Vielzahl wieder verwendbarer Monitorkomponenten. Es ist möglich Ereignishandler für das Verlassen bestimmter Parameterbereiche zu konfigurieren, oder auch periodisch Messwerte zu verarbeiten.

Zusammenfassung

Mit dem *Adapt.Net* Framework, welches am Hasso-Plattner-Institut entwickelt wurde, entstand eine Reihe von Werkzeugen, die die Entwicklung selbstadaptiver Anwendungen durch Generierung konfigurationsspezifischer Schnittstellen stark vereinfachen. Eine wiederverwendbare Infrastruktur für die Beobachtungen von Umgebungsparametern sowie die unterstützte Auswertung von Adaptionenprofilen stellen weitere Vereinfachungen dar. Von zentraler Bedeutung bei der Entwicklung selbstadaptiver Anwendung ist die Entwicklung von alternativen Anwendungskonfigurationen und die Auswahl und Änderung von Konfiguration während der Laufzeit, welche durch einem implementierten Rekonfigurationsalgorithmus mit sehr kurzen Umschaltzeiten realisiert wurde. Bei der Entwicklung von alternativen Konfigurationen wird der Entwickler durch automatische Codegenerierung für die Integration ausgewählter Architekturmuster unterstützt.

Die Zuordnung von Anwendungskonfigurationen zu gegebenen Umgebungseigenschaften kann Kategorie-basiert anhand von Profilen erfolgen - muss in einigen Situationen aber programmatisch erfolgen, um komplexere Anpassungen vornehmen zu können.

[Ras05] Andreas Rasche, Marco Puhmann and Andreas Polze "Heterogeneous Adaptive Component-Based Applications with Adapt.Net" in Proceedings of International Symposium on Object-oriented Real-time Distributed Computing (ISORC), Seattle, Washington, USA, May 2005

[Ras05/2] Andreas Rasche, Wolfgang Schult, and Andreas Polze "Self-Adaptive Multithreaded Applications - A Case for Dynamic Aspect Weaving" in Proceedings of the 4th Workshop on Adaptive and Reflective Middleware (ARM 2005), Grenoble, France - November 28, 2005

[Sch05] Wolfgang Schult and Andreas Polze "Aspect-Oriented Programming with C# and .NET" in Proceedings of International Symposium on Object-oriented Real-time distributed Computing (ISORC) 2002, pp. 241-248, Crystal City, VA, USA, April 29 - May 1 2002