Eine Taxonomie für aspektorientierte Systeme

Stefan Hanenberg, Dominik Stein, Rainer Unland

Universität Duisburg-Essen
Institut für Informatik und Wirtschaftsinformatik
Schützenbahn 70, 45117 Essen
{shanenbe, dstein, unlandr}@informatik.uni-essen.de

Abstrakt: Die aspektorientierte Softwareentwicklung entwickelte sich aus der Beobachtung heraus, dass eine Vielzahl logisch zusammenhängender Softwareelemente mit Hilfe konventioneller Techniken nicht modularisierbar ist. Eine mangelhafte Modularisierung wiederum reduziert die Verständlichkeit und Wartbarkeit der Software. Aspektorientierte Systeme bieten zusätzliche Konstrukte, um ein höheres Maß an Modularisierung zu ermöglichen. Es existiert bereits eine Reihe von Systemen, welche die aspektorientierte Softwareentwicklung in unterschiedlichsten Ausprägungen unterstützen. Es fehlt jedoch an Kriterien, anhand derer überprüft werden kann, ob ein bestimmtes System die Modularisierung eines gegebenen Anwendungsproblems ermöglicht. In diesem Papier schlagen wir eine Taxonomie aspektorientierter Systeme vor. Anhand derer können zum einen unterschiedliche aspektorientierte Systeme miteinander verglichen werden. Zum anderen ermöglicht die Taxonomie es festzustellen, ob ein System zur Lösung eines gegebenen Problems geeignet ist.

1 Einleitung

Die aspektorientierte Softwareentwicklung (engl.: *aspect-oriented software development*, AOSD, [KLM+97]) ist ein neuer Ansatz in der Softwareentwicklung, der die Modularisierung sogenannter *crosscutting concerns* in *Aspekte* fokussiert. Als *crosscutting concern* werden dabei logisch zusammenhängende Elemente eines Programms bezeichnet, die mit Hilfe der zugrundeliegenden Programmiersprache nicht modularisierbar sind. Beispiele dafür sind Implementierungen des Observer-Entwurfsmusters ([GHJV95], vgl. z.B. [GyBr03, SHU02, HaKi02]) oder Implementierungen zur Zusicherung von Objektpersistenz (vgl. z.B. [RaCh03]).

Abbildung 1 illustriert die Funktionsweise eines aspektorientierten Systems[1]. Zunächst gibt es die Basisapplikation (geschrieben in einer Basissprache), in die *crosscutting concerns* integriert werden müssen. Das aspektorientierte System dekomponiert das Basissystem in sogenannte *Join Points* [KLM+97]. Join Points beschreiben jene Elemente der Basisapplikation, an denen Aspekte integriert werden können. Um einen

[1] Wir verwenden den Begriff des aspektorientierten Systems zur allgemeinen Bezeichnung aller Ansätzen, die die aspektorientierte Entwicklung durch (a) neue Programmiersprachenkonstrukte, (b) Softwarebibliotheken oder (c) externe Werkzeuge unterstützen.

Aspekt zu integrieren, muss eine entsprechende Menge von Join Points bestimmt werden, sowie ferner eine Vorschrift, welche besagt, wie die selektierten Join Points zu adaptieren sind. Dazu bieten aspektorientierte Systeme zum einen Selektionssprachen und zum anderen unterschiedliche Adaptionsmechanismen an. Der Schritt, in dem die Integration eines Aspekts in das Basissystem vollzogen wird, wird als *Weben* bezeichnet. Als Aspekt wird ein Modul bezeichnet, welches die Spezifikation der Selektion von Join Points als auch deren Adaption beinhaltet.

Es gibt eine Menge von Systemen, die allgemein akzeptiert als *aspektorientiert* bezeichnet werden. Darunter fallen zum Beispiel *AspectJ* [KHH+01], *Hyper/J* [OsTa01], *Sally* [HaUn03] oder *AspectS* [Hirs02]. Diese Systeme unterscheiden sich zum Teil in wesentlichen Punkten: Zum einen wird das grundlegende Konzept der Aspektorientierung – der *Join Point* – unterschiedlich ausgelegt, und zum anderen weisen die Operationen auf Join Points – *Selektion* und *Adaption* – unterschiedliche Formen und Möglichkeiten auf.

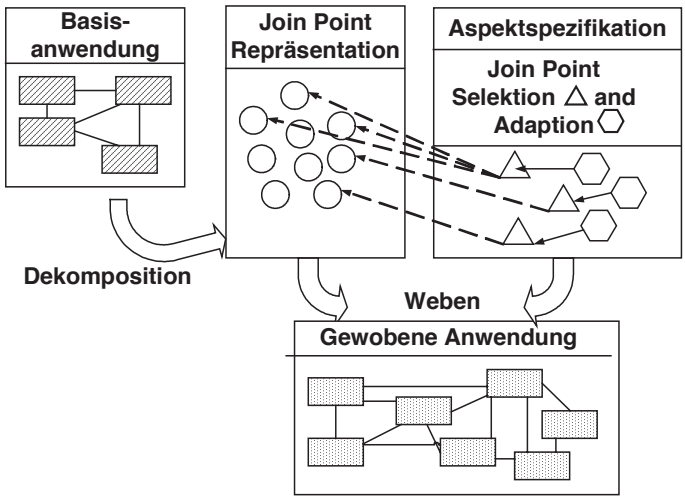

Abbildung 1: Schematische Darstellung eines aspektorientierten Systems

Das hieraus resultierende Problem ist, dass sich unterschiedliche Systeme unterschiedlich gut zur Modularisierung eines gegebenen Problems eignen. Es fehlt jedoch an abstrakten Kriterien, anhand derer die Eignung eines aspektorientierten Systems hinsichtlich eines gegebenen Problems überprüft werden kann, ohne dass detaillierte Kenntnisse über das System bekannt sein müssen.

Dieses Papier stellt eine Taxonomie aspektorientierter Systeme vor. Die der Taxonomie zugrundeliegenden Merkmale werden zum einen aus der Literatur extrahiert als auch aus dem Vergleich unterschiedlicher Systeme gewonnen. Wir zeigen, dass diese Taxonomie es ermöglicht, auf abstrakter Ebene die Eignung eines gegebenen Systems für ein bestimmtes Problem zu bestimmen.

In Abschnitt 2 stellen wir ein gegebenes Problem – die Implementierung des Observer-Entwurfmusters – vor, welches in der aspektorientierten Literatur als typischer Anwendungsfall für die Aspektorientierung identifiziert wird (vgl. z.B. [GyBr03, HaKi02, SHU02]). Wir diskutieren kurz die daraus resultierenden Anforderungen an ein aspektorientiertes System und zum anderen die Problematik, die sich aus dem Mangel an Unterscheidungsmerkmalen von aspektorientierten Systemen ergibt. In Abschnitt 3 führen wir die Taxonomie ein. In Abschnitt 4 wenden wir diese auf bestehende Systeme und auf das vorgestellte Problem an und überprüfen die Eignung der Systeme zur Lösung des Problems. In Abschnitt 5 diskutieren wir verwandte Arbeiten. Abschließend diskutieren wir die vorgeschlagene Taxonomie und fassen das Papier zusammen.

2 Problembeschreibung

Im Folgenden beschreiben wir eine Realisierung des Observer-Entwurfsmusters [GHJV95] in einem Anwendungsszenario. Beim Observer-Entwurfsmuster registrieren sich sogenannte *Observer*-Objekte bei *Subject*-Objekten und werden mittels einer entsprechenden update-Nachricht informiert, sobald sich der Zustand der Subjects ändert.

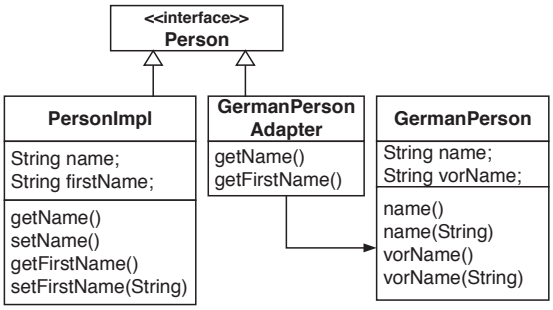

Abbildung 2: Schnittstelle Person und mögliche Implementierungen der Schnittstelle

Als Beispiel betrachten wir Objekte vom Typ Person, welche von einem GUI-Element beobachtet werden sollen. Person ist eine Schnittstellenbeschreibung, welche den Namen und Vornamen von Personen abrufbar macht, selbst aber keine Methoden zur Änderung von Namen und Vornamen beinhaltet. Abbildung 2 zeigt zwei mögliche Implementierungen des Typs Person auf: Zum einen durch eine Klasse PersonImpl, die entsprechende Felder für Namen und Vornamen einer Person bereitstellt, zum anderen durch eine Klasse GermanPersonAdapter, die die Klasse GermanPerson dem Adapter-Entwurfsmuster [GHJV95] entsprechend adaptiert. Bei dieser Implementierung erfolgen Zustandsänderungen eines Person-Objekts sowohl bei der Änderung von Feldern der Person-Objekte (d.h. von Instanzen von PersonImpl) als auch bei der Änderung der Felder von Objekten, die von einem Person-Objekt erreichbar sind (also von Instanzen von GermanPerson, die von Person-Objekten adaptiert werden).

Aus aspektorientierter Sicht stellt sich die Frage, ob ein System in der Lage ist, die Join Points „*Änderung eines Feldes eines Person-Objekts*“ und „*Änderung eines Feldes eines*

durch ein Person-Objekt erreichbaren Objekts“ zu selektieren. Ferner sollen an diesen Join Points die Beobachter der Person-Objekte informiert werden. D.h. die gewünschte Adaption – nämlich, dass zusätzliche Nachrichten an die Beobachter verschickt werden – muss durchgeführt werden können.

Um festzustellen, ob ein gegebenes aspektorientiertes System in der Lage ist, die oben beschriebenen Selektionen auszudrücken, müssen detaillierte Kenntnisse des Systems vorliegen. D.h. ein Entwickler muss sich mit Sprachkonstrukten z.B. von AspectJ vertraut machen, um abschätzen zu können, ob derartige Selektionen und Adaptionen möglich sind. Für den Fall, dass AspectJ keine entsprechenden Konstrukte zur Verfügung stellt, muss sich der Entwickler mit den Sprachkonstrukten zum Beispiel von Hyper/J vertraut machen, usw. Eine solche Herangehensweise stellt einen immensen Aufwand dar: Was an dieser Stelle benötigt wird, ist eine Menge von Kriterien, anhand derer die Eignung eines aspektorientierten Systems hinsichtlich einer Problemstellung bestimmt werden kann.

3 Eine Taxonomie für aspektorientierte Systeme

Im Folgenden stellen wir eine Taxonomie vor, welche auf den unterschiedlichen Ausprägungen von Join Points, Selektionskriterien und Adaptionen basiert.

3.1 Join Points

Ein Join Point stellt ein selektierbares und adaptierbares Element der Basisanwendung dar. Bei Betrachtung konkreter AO-Systeme stellt sich heraus, dass der Join Point Begriff unterschiedlich interpretiert wird. Diese Interpretationen lassen sich zum einen hinsichtlich ihrer *Dynamik*, zum anderen hinsichtlich ihrer *Abstraktion* unterscheiden, wobei beide Unterscheidungskriterien zueinander orthogonal sind (siehe Abbildung 3 a).

Das Merkmal **Dynamik** beschreibt den Zeitpunkt, an dem ein Join Point im System auftritt. Wir unterscheiden zwischen statischen und dynamischen Join Points:

> **Statischer Join Point:** Ein statischer Join Point beschreibt ein selektierbares und adaptierbares Element, das direkt aus dem Quellcode der Basisapplikation extrahiert werden kann[2].
>
> **Dynamischer Join Point**: Ein dynamischer Join Point beschreibt ein selektierbares und adaptierbares Laufzeitelement der Basisapplikation[3].

Ein statischer Join Point ist zum Beispiel eine Methoden- oder Klassendefinition in einer klassenbasierten objektorientierten Programmiersprache. Ein dynamischer Join Point ist

---

[2] Diese Interpretation spiegelt sich in der Literatur wieder, in der ein Join Point als ein „*systematischer Ort*“ [Fiel01] oder als „*Punkt einer Komponente, wo Aspekte integriert werden*“ [AßLu99] bezeichnet wird.
3 Dieser Interpretation folgt insbesondere [KHH+01], wo ein Join Point als „*wohl-definierter Punkt in der Ausführung eines Programms*“ bezeichnet wird.

beispielsweise ein Laufzeitobjekt oder eine Nachricht, die zur Laufzeit zwischen Objekten verschickt wird.

Jenseits des Unterscheidungsmerkmals der Dynamik unterscheiden sich die gängigen Interpretationen des Join Point Begriffs hinsichtlich der **Abstraktionsebene** des selektierbaren Elements. Das zugrundeliegende Unterscheidungsmerkmal hierbei ist, ob Join Points strukturelle Eigenschaften oder aber Verhalten widerspiegeln.

> **Struktureller Join Point:** Ein struktureller Join Point ist ein selektierbares und adaptierbares Element, welches eine strukturelle Eigenschaft der Basisanwendung widerspiegelt[4].
>
> **Verhaltensbasierter Join Point**: Ein verhaltensbasierter Join Point ist ein selektierbares und adaptierbares Element, welches das Verhalten der Basisanwendung widerspiegelt[5].

Je nachdem, welche Arten von Join Points ein aspektorientiertes System implementiert, können zum Beispiel Nachrichten oder Zuweisungen selektiert und adaptiert werden (verhaltensbasierte Join Points) oder aber Klassen und Objekte (strukturelle Join Points).

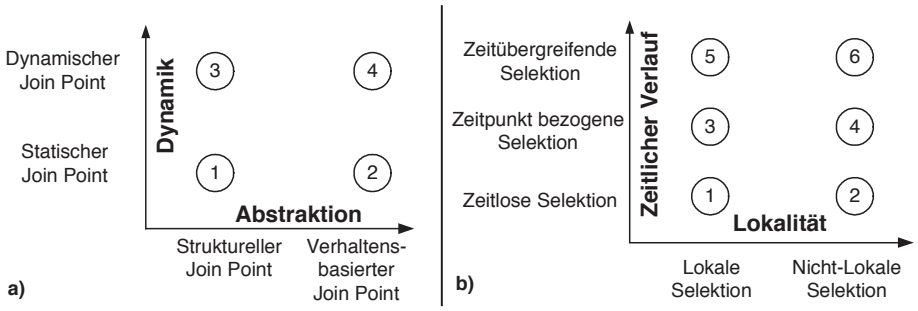

Abbildung 3: Ausprägungen von a) Join Points und b) Join Point Selektionen

Die Merkmale Dynamik und Abstraktionsebene sind orthogonal. D.h. Join Points eines gegebenen Systems lassen sich gleichzeitig hinsichtlich ihrer Dynamik als auch hinsichtlich ihrer Abstraktionsebene unterscheiden. Entsprechend ergeben sich – wie in Abbildung 3 a beschrieben – vier unterschiedliche Arten von Join Points. Ein aspektorientiertes System basierend auf einer klassenbasierten Basissprache kann zum Beispiel Objekte als strukturelle und dynamische Join Points unterstützen sowie Methodenaufrufe (d.h. die Elemente des Quelltexts, welche Methodenaufrufe repräsentieren) als verhaltensbasierte und statische Join Points.

---

[4] Beispielsweise bezeichnet Ossher in [EAK+01] Klassen und Methoden als Join Points.
[5] So argumentiert Lieberherr in [EAK+01], dass Knoten in einem „dynamischen Aufrufgraphen“ Join Points repräsentieren.

3.2 Selektionskriterien

Wir unterscheiden die von aspektorientierten Systemen angebotenen Selektionskriterien zum einen anhand ihres *Zeitbezugs*, andererseits anhand ihrer *Lokalität* bezüglich des zu selektierenden Join Points (siehe Abbildung 3 b), wobei auch in diesem Falle beide Klassifikationsmerkmale zueinander orthogonal sind.

Hinsichtlich des **Zeitbezugs** unterscheiden wir drei Arten von Selektionskriterien:

> **Zeitloses Selektionskriterium:** Ein Selektionskriterium ist zeitlos, wenn der Join Point aufgrund seiner Beziehung zum Quelltext der Basisapplikation selektiert wird. Dies ist z.B. der Fall, wenn Nachrichten aufgrund ihres Namens selektiert werden.
>
> **Zeitpunktbezogenes Selektionskriterium:** Ein Selektionskriterium ist zeitpunktbezogen, wenn der Join Point aufgrund des aktuellen Zustands der Applikation selektiert wird. Ein Beispiel hierfür ist die Selektion einer Nachricht aufgrund des dynamischen Typs eines Laufzeitparameters.
>
> **Zeitübergreifendes Selektionskriterium:** Ein Selektionskriterium ist zeitübergreifend, wenn das selektierbare Konstrukt dynamisch ist und aufgrund des Zustands der Applikation zu einem anderen Zeitpunkt selektiert wird. Ein Beispiel einer zeitübergreifenden Selektion ist die kontrollflussspezifische Selektion eines Join Points.

Die zeitpunktbezogene und zeitübergreifende Selektion ist nur für dynamische Join Points relevant. Die Selektion statischer Join Points reduziert sich auf zeitlose Selektionskriterien, da statische Join Points grundsätzlich durch Quelltextelemente repräsentiert werden und damit keinen Zeitbezug haben.

Abgesehen vom Zeitbezug unterscheiden wir aspektorientierte Systeme ferner hinsichtlich der **Lokalität** der innerhalb des Selektionskriteriums verwendeten Informationen (in Bezug auf den zu selektierenden Join Point): Zum Beispiel werden in Hyper/J Methoden oder Klassen (d.h. statische, strukturelle Join Points) durch ihre Methoden- oder Klassennamen selektiert. Es ist nicht möglich eine *musterbasierte Selektion* (vgl. [GyBr03]) anzugeben, wie etwa die Selektion einer Klasse aufgrund ihrer Verwendung als Typ in einer Methode.

> **Lokales Selektionskriterium**: Ein Selektionskriterium ist lokal, wenn es sich auf lokal abfragbare Informationen des zu selektierenden Join Points bezieht.
>
> **Nicht-lokales Selektionskriterium**: Ein Selektionskriterium ist nicht-lokal, wenn das Kriterium sich auf Systeminformationen bezieht, welche nicht zur lokalen Umgebung des Join Points gezählt werden.

Wir führen nicht-lokale Selektionskriterien auf die Erreichbarkeit von Informationen an einem zu selektierenden Join Point zurück. Dementsprechend bezeichnen wir zum Beispiel die Selektion einer Nachricht aufgrund des Zustands eines übergebenen Parameters als lokales Selektionskriterium: Der Parameter ist aus Sicht der Nachricht direkt erreich-

bar. Wenn eine Nachricht hingegen aufgrund dessen selektiert wird, dass ein Parameter von einem anderen Objekt referenziert wird, dann bezeichnen wir ein solches Kriterium als nicht-lokal, da Informationen über andere Objekte, die Referenzen auf einen Parameter haben, nicht aus dem aktuellen Parameter gewonnen werden können.

Entsprechend Abbildung 3 b sind die Merkmale Zeitbezug und Lokalität orthogonal, und es ergeben sich 6 unterschiedliche Arten von Selektionskriterien.

3.3 Join Point Adaptionen

Wir unterscheiden Adaptionen von Join Points hinsichtlich ihrer *Abstraktion*, der *Variabilität* und des *Effekts* der Adaption. Hinsichtlich der **Abstraktion** unterscheiden wir zwischen den Ausprägungen Struktur- und Verhaltensadaption:

> **Strukturadaption**: Eine Strukturadaption ist die Adaption eines Join Point, wenn dieser um Strukturelemente erweitert oder verringert wird. Ein Beispiel für Strukturadaptionen sind z.B. *introductions*, welche Klassen um zusätzliche Methoden oder Felder erweitern (siehe [HaUn03] für weitere Diskussionen).
>
> **Verhaltensadaption:** Eine Verhaltensadaption eines Join Points modifiziert den Join Point hinsichtlich seines Verhalten, ohne dabei seine strukturellen Eigenschaften zu ändern. Adaptionen dieser Art sind zum Beispiel *advice* in AspectJ, die das Verhalten der Basisanwendung an einem bestimmten Join Point ergänzen.

Die **Variabilität** unterscheidet zwischen fixen und variablen Adaptionen:

> **Fixe (nicht-variable) Adaption**: Eine Adaption ist fix, wenn die Art der Adaption für alle selektierten Join Points identisch ist.
>
> **Variable Adaption**: Eine Adaption ist variabel, wenn sich die Art der Adaption in unterschiedlichen Join Points unterscheiden kann.

Die Unterscheidung der Variabilität von Adaptionen beruht auf der Beobachtung, dass eine Adaption in manchen aspektorientierten Systemen in Abhängigkeit von dem Join Point, auf den sie angewandt wird, variieren kann (wobei alle zu adaptierenden Join Points durch die gleiche Selektion ausgewählt worden sind). So ermöglichen zum Beispiel die in [BMDV02, HaUn03] vorgestellten Systeme die Generierung von Methoden in Abhängigkeit von der Selektion der zugehörigen Klasse. Systeme wie etwa AspectJ unterscheiden sich von diesen Systemen dahingehend, dass ihre Adaptionen nicht variabel sind, d.h. dass eine gegebene Adaption für jeden selektierten Join Point identisch ist.

Zuletzt unterscheiden wir hinsichtlich des **Effekts**, der auf die Join Points ausgeübt wird:

> **Konstruktive Adaption**: Eine Adaption ist konstruktiv, wenn das Verhalten oder die Struktur eines zu adaptierenden Join Points durch die Adaption erhalten bleibt.
>
> **Destruktive Adaption**: Eine Adaption ist destruktiv, wenn das Verhalten oder die Struktur eines zu adaptierenden Join Points durch die Adaption ersetzt wird.

Die zugrundeliegende Beobachtung ist hierbei, dass Systeme wie AspectJ Join Points dahingehend adaptieren können, dass das ursprüngliche Verhalten ersetzt wird. Auf der anderen Seite erlaubt AspectJ bei der Adaption von Klassen lediglich das Hinzufügen von neuen Methoden, Feldern oder Schnittstellen, nicht aber ihre Entfernung. In ähnlicher Weise erlaubt Hyper/J bei der Adaption von Methodenaufrufen nur, dass zusätzliche Aufrufe hinzugefügt werden dürfen, aber nicht, dass die in der Basisanwendung enthaltenen Aufrufe vollständig ersetzt werden.

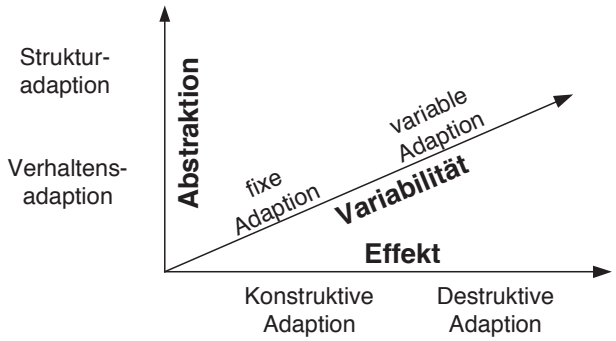

Abbildung 4: Ausprägungen von Join Point Adaptionen

Auch die hier beschriebenen Merkmale sind orthogonal, d.h. die von einem aspektorientierten System bereitgestellten Möglichkeiten der Adaption können gleichzeitig hinsichtlich ihrer Abstraktion, der Variabilität und des Effekts klassifiziert werden. So stellen *introductions* in AspectJ konstruktive und fixe Strukturadaptionen dar, *parametric introductions* [HaUn03] hingegen konstruktive und variable Strukturadaptionen.

4 Anwendung der Taxonomie

Das in Abschnitt 2 beschriebene Problem lässt sich in die in den vorherigen Abschnitten beschriebene Taxonomie einordnen. Zunächst erfordert die Observer-Implementierung die Selektion von Zustandsänderungen – und zwar (1) von Person-Objekten und (2) von Objekten, die von Person-Objekten aus erreichbar sind. Da Zustandsänderungen durch Feldzuweisungen induziert werden, bezieht sich die Selektion ferner auf **verhaltensbasierte** Join Points. Darüber hinaus besagt Kriterium (1), dass der Join Point aufgrund des *Typs* eines *Objekts* selektiert werden soll. Hieraus folgt, dass die Selektion auf einem **dynamischen** Join Point basiert. Da sich die Selektion auf den aktuellen Typ eines Objekts bezieht, ist die zugrundeliegende Sektion **zeitpunktbezogen**. Das Selektionskriterium ist hinsichtlich des Join Points ferner **lokal**, da das Objekt, dessen Feld einen neuen Wert bekommt, bei einer Feldzuweisung bekannt sein muss, also zum lokalen Kontext des Join Points gehört.

Die getroffenen Aussagen bezüglich der Dynamik und der Abstraktion des Join Points treffen ebenfalls auf Kriterium (2) zu (**dynamischer** und **verhaltensbasierter** Join Point). Das Selektionskriterium beinhaltet darüber hinaus jedoch die Aussage, dass das Objekt, auf welchem die Zuweisung ausgeführt wird, von einem Person-Objekt aus

erreichbar sein muss. Damit bezieht sich das Selektionskriterium auf ein Objekt, das nicht zum lokalen Kontext des Join Points gehört – es ist daher **nicht-lokal**. Die Selektion ist schließlich **zeitpunktbezogen**, da die Beziehungen zwischen den *Objekten* (d.h. die Erreichbarkeit des Person-Objekts) zu dem Zeitpunkt überprüft wird, in dem der Join Point auftritt.

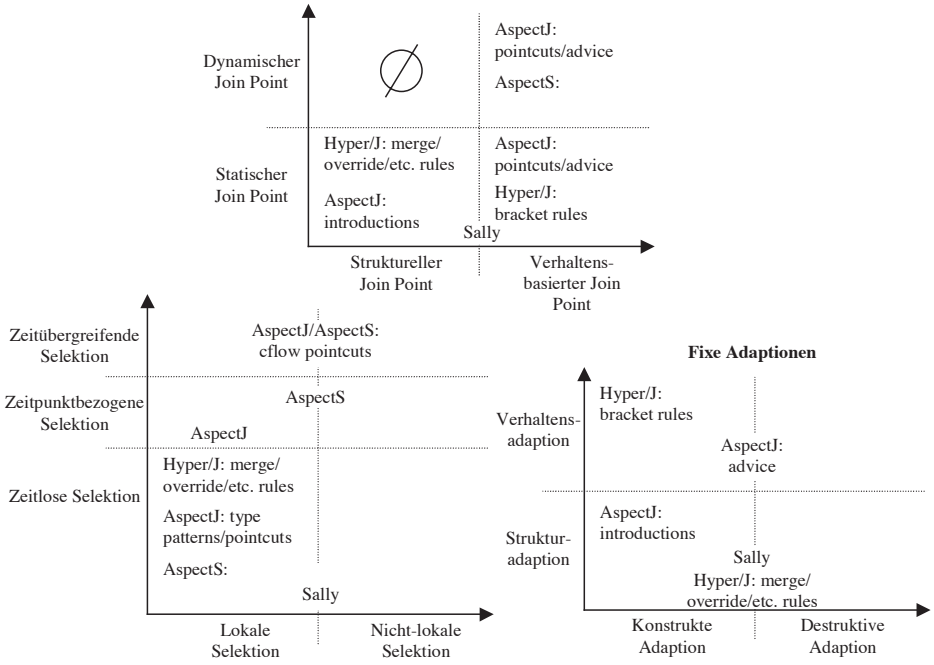

Abbildung 5: Charakterisierung der Systeme AspectJ, AspectS, Hyper/J und Sally[6]

Hinsichtlich der Adaption des Join Points erwartet das Observer-Beispiel, dass – zuzüglich zum ursprünglichen Verhalten (der Zustandsänderung) – eine update-Nachricht an alle Observer-Objekte versendet wird. D.h. es werden **konstruktive Verhaltensadaptionen** benötigt. Da das auszuführende Verhalten statisch beschreibbar ist (Nachricht update), sind **fixe** Adaptionen ausreichend.

Nachdem wir das Anwendungsproblem hinsichtlich der in diesem Papier beschriebenen Merkmale charakterisiert haben, können wir diese Beobachtungen den Merkmalen konkreter aspektorientierter Systeme gegenüberstellen.

Abbildung 5 illustriert die Charakterisierung der Systeme AspectJ, Hyper/J, Sally und AspectS gemäß der in Abschnitt 3 beschriebenen Taxonomie (auf eingehendere Erläuterungen muss an dieser Stelle aus Platzgründen verzichtet werden). Anhand der Abbil-

[6] Hinsichtlich der Adaption verzichten wir auf die Darstellung von variablen Adaptionen, da nur Sally von den genannten Systemen variable Adaptionen unterstützt.

dung wird ersichtlich, dass ein System sich nicht auf eine Art von Join Point, Selektionskriterium oder Adaption beschränken muss. Z. B. ermöglichen Systeme, die dynamische Join Points unterstützen, in der Regel sowohl zeitlose als auch zeitbehaftete Selektionen.

Die zur Lösung des Anwendungsproblems geeigneten aspektorientierten Systeme sind jene, welche die aus dem Anwendungsproblem identifizierten Join Point Arten, Selektionskriterien und Adaptionsmöglichkeiten zur Verfügung stellen. Da weder Hyper/J noch Sally dynamische Join Points (und somit keine zeitbehafteten Selektionen) anbieten, sind beide für eine Observer-Implementierung eher ungeeignet. Zur Implementation des ersten Selektionskriteriums bieten sich AspectJ und AspectS an, da beide Systeme **dynamische** und **verhaltensbasierte** Join Points, **lokale** und **zeitbehaftete** Selektionskriterien, sowie **konstruktive** und **fixe Verhaltensadaptionen** unterstützen.

Hinsichtlich des zweiten Selektionskriteriums hingegen erweist sich AspectJ als nicht geeignet: AspectJ ermöglicht nur kontrollflussspezifische, nicht-lokale Selektionen. Infolgedessen kann die Erreichbarkeit zwischen Objekten nicht als Selektion ausgedrückt werden. AspectS dagegen erlaubt die Spezifikation von **zeitpunktbezogenen**, **nicht-lokalen** Selektionskriterien und scheint demnach als einziges System für die Lösung des beschriebenen Problems geeignet.

5 Verwandte Arbeiten

In [RaSu03] werden zwei orthogonale Klassifizierungsmerkmale für aspektorientierte Systeme vorgeschlagen. Das erste Merkmal beschreibt die *Reichhaltigkeit* der Selektionssprache, das zweite Merkmal beschreibt die *Ebene des Aspektwebens*. Hinsichtlich der Reichhaltigkeit unterscheiden [RaSu03] zwischen *einfacher* und *reichhaltiger Selektionssprache*. Bezüglich der Ebene des Aspektwebens unterscheiden [RaSu03] zwischen *typbasiertem* und *objektbasiertem Weben*. Die Autoren beschreiben jedoch nicht, anhand welcher Kriterien festgestellt werden kann, ob eine Selektionssprache als einfach oder reichhaltig bezeichnet werden kann.

In [MaKi03] wird ein Rahmenwerk zur Implementierung aspektorientierter Systeme vorgestellt. Dieses Rahmenwerk beinhaltet einen Aspektweber, welcher zwei Programme miteinander kombiniert. Der Weber ist als 11-Tupel modelliert, wobei jedes Element des Tupels eine unterschiedliche Sicht auf das System widerspiegelt, wie zum Beispiel Join Points, Unterscheidungsmerkmale von Join Points und Adaptionen von Join Points. Das Rahmenwerk bietet die Möglichkeit, Gemeinsamkeiten von aspektorientierten Systemen zu beschreiben: Aspektorientierte Systeme können demnach mit Hilfe des 11-Tupels beschrieben werden. Unterscheidungsmerkmale von aspektorientierten Systemen jedoch werden durch das Rahmenwerk nicht explizit erfasst.

In [LLM99, GyBr03] werden Terme wie *lexikalisches crosscutting* und *aufzählungsbasiertes crosscutting* eingeführt. Lexikalisches crosscutting liegt demnach vor, wenn die Aspektdefinition lexikalische Elemente der Basisapplikation enthält. Diese Betrachtung lässt sich zum Teil auf das in Abschnitt 3.2 beschriebene zeitlose Selektionskriterium anwenden, da auch hier Elemente der Spezifikation des Basisprogramms zur Selektion

eingesetzt werden. Aufzählungsbasiertes crosscutting bezeichnet den Mangel der Selektionssprache, Funktionen über Join Points anzubieten. Die von uns beschriebenen Merkmale dagegen gehen nicht auf diese Eigenschaften der Selektionssprache ein, sondern beschreiben nur, welche Arten von Selektionskriterien ausdrückbar sind.

6 Zusammenfassung und Diskussion

Das vorliegende Papier schlägt eine Taxonomie für aspektorientierte Systeme vor. Diese unterscheidet aspektorientierte Systeme hinsichtlich der Art der Join Points, die durch das System angeboten werden, und der Art der Selektionen und Adaptionen, die auf Join Points angewendet werden können. Dabei unterscheiden wir Join Points hinsichtlich ihrer *Dynamik* und *Abstraktion*, Selektionen aufgrund ihres *zeitlichen Bezugs* und ihrer *Lokalität* und Adaptionen hinsichtlich ihrer *Abstraktion*, *Variabilität* und ihres *Effekts*. In Abschnitt 4 haben wir die Taxonomie auf bestehende Systeme angewandt. Ferner haben wir anhand des Observer-Beispiels gezeigt, dass die Taxonomie geeignet ist, gegebene aspektorientierte Systeme auf ihre Eignung zur Lösung eines Problem hin zu überprüfen.

Die vorgeschlagenen Merkmale sind sehr abstrakter Natur. So abstrahierten sie zum Beispiel davon, welche konkreten Join Points ein System anbietet. So wird nicht überprüft, ob ein System zum Beispiel Zugriffe auf Arrays oder etwa Ausnahmefehler als Join Points anbietet. Ebenso wird davon abstrahiert, welche Ausdrucksmächtigkeit eine Selektionssprache besitzt: Es wird nicht ersichtlich, dass Sally z.B. eine Turingvollständige (auf Prolog basierende) Sprache zur Selektion einsetzt oder dass AspectJ unterschiedliche Möglichkeiten zur Spezifikation von Parametertyplisten ermöglicht. Ferner besagt die Ausprägung der zeitübergreifenden Selektionen lediglich, ob Join Points aufgrund von Elementen zu unterschiedlichen Zeitpunkten selektiert werden können. Welche Formen diese Selektionen haben, wird nicht festgestellt: Vielmehr wird darüber abstrahiert, ob zum Beispiel – wie in AspectJ und AspectS – nur Call-Stack-Informationen abrufbar sind oder – wie in [MaKa03] vorgeschlagen – Join Points auch aufgrund des Datenflusses selektiert werden können.

Bei der Abbildung der Problembeschreibung in Abschnitt 4 haben wir jene Systeme verworfen, die keine zeitübergreifenden Selektionen unterstützen. In manchen Situationen kann es jedoch möglich sein, zeitübergreifende Selektionen durch zeitpunktbezogene Selektionen mit entsprechenden Adaptionen nachzubilden. Die Konsequenzen dieser Möglichkeit auf die Eignung eines Systems zur Lösung eines gegebenen Problems gilt es in zukünftigen Arbeiten näher zu untersuchen.

Der Nutzen der hier vorgeschlagenen Taxonomie liegt darin, dass auf sehr abstraktem Niveau grundlegende Aussagen über ein gegebenes aspektorientiertes Systems getroffen werden können, ohne dass detaillierte Kenntnisse über das System bekannt sein müssen. Darüber hinaus fördert die Taxonomie ein abstraktes Verständnis der aspektorientierten Softwareentwicklung, ohne dass systemspezifische Terme wie *pointcuts*, *advice* (für AspectJ) oder *composition rules* (für Hyper/J) zur Erklärung herangezogen werden müssen. Dieses abstrakte Verständnis fördert ebenfalls die Kommunikation zwischen Entwicklern, die unterschiedliche aspektorientierte Systeme nutzen.

Literaturverzeichnis

[Aks03] Aksit, M. (Hrsg.): Proc. of AOSD 2003, Boston, MA, March 17 - 21, ACM, 2003.

[AßLu99] Aßmann, U.; Ludwig, A.: Aspect Weaving with Graph Rewriting. In: Czarnecki, K.; Eisenecker, U. W. (Eds.): Proc. of GCSE'99, Erfurt, Germany, September 28-30, 1999, LNCS 1799, Springer, 2000, S. 24-36.

[BMDV02] Brichau, J.; Mens, K.; De Volder, K.: Building Composable Aspect-Specific Languages with Logic Metaprogramming. In Batory, D. (Hrsg.): Generative Programming and Component Engineering (GPCE), LNCS 2487, Springer 2002, S. 110-127.

[EAK+01] Elrad, T.; Aksit, M.; Kiczales, G.; Lieberherr, K.; Ossher, H.: Discussing Aspects of AOP, Communication of the ACM, Vol. 44, No. 10, October, 2001, pp. 33-38.

[Fiel01] Filman, R. E.: What is AOP: Revisited, Workshop on Multi-Dimensional Separation of Concerns at ECOOP, Budapest, Hungary, June 18-22, 2001.

[GHJV95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[GyBr03] Gybels, K.; Brichau, J.: Arranging Language Features for More Robust Pattern-based Crosscuts, In: [Aks03], S. 60-69.

[HaKi02] Hannemann, J.; Kiczales, G.: Design pattern implementation in Java and AspectJ. Proc. of OOPSLA 2002, November 4-8, 2002, Seattle, Washington, USA. SIGPLAN Notices 37(11), ACM, S. 161-173.

[HaUn03] Hanenberg, S.; Unland, R.: Parametric Introductions, , In: [Aks03], S. 80-89.

[Hirs02] Hirschfeld, R.: AspectS - Aspect-Oriented Programming with Squeak. In M. Aksit, M. Mezini, R. Unland (Hrsg.): Objects, Components, Architectures, Services, and Applications for a Networked World, LNCS 2591, Springer, 2003, S. 216-232.

[Kic02] Kiczales, G. (Hrsg.): Proc. of AOSD 2002, Enschede, The Netherlands, April 22-26, ACM, 2002.

[KHH+01] Kiczales, G.; Hilsdale, E.; Hugunin, J.; Kersten, M.; Palm, J.; Griswold, W. G.: An Overview of AspectJ, In: Proc. of ECOOP 2001, LNCS 2072, Springer, 2001, S. 327-353.

[KLM+97] Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C.; Loingtier, J.-M.; Irwing, J.: Aspect-Oriented Programming. In Aksit, B; Matsuoka, V., (Hrsg.): Proc. of ECOOP 1997, LNCS 1241, Springer, 1997, S. 220-242.

[LLM99] Lieberherr, K.; Lorenz, D.; Mezini, M.: Programming with Aspectual Components, Technical Report, College of Computer Science, Northeastern University, March, NU-CCS-99-01, Boston, MA, 1999.

[MaKa03] Masuhara, H.; Kawauchi, K.: Dataflow Pointcut in Aspect-Oriented Programming, In: Ohori, A. (Hrsg.): Proc. of APLAS 2003, LNCS 2895, Springer, 2003, S. 105-121.

[MaKi03] Masuhara, H.; Kiczales, G.: Modeling Crosscutting in Aspect-Oriented Mechanisms, In: Cardelli, L. (Ed.): Proc. of ECOOP 2003, Darmstadt, Germany, July 21-25, 2003, LNCS 2743, Springer 2003, S. 2-28.

[OsTa01] Ossher, H.; Tarr, P.: Using multidimensional separation of concerns to (re)shape evolving software. Communication of the ACM, 44 (10), 2001, S. 43-50.

[RaCh03] Rashid, A.; Chitchyan, R.: Persistence as an aspect, In: [Aks03], S. 120-129.

[RaSu03] Rajan, H.; Sullivan, K.: Need for Instance Level Aspect Language with Rich Pointcut Language, Proc. of the SPLAT Workshop at AOSD 2003, Boston, MA, USA, March 18, 2003, S. 21-29.

[SHU02] Stein, D.; Hanenberg, S.; Unland, R.: A UML-based aspect-oriented design notation for AspectJ, In: [Kic02], S. 106 - 112.