Krasemann, Hartmut; Brauer, Johannes; Crasemann, Christopher

Working Paper

Ein Referenzmodell für Schnittstellen

Arbeitspapiere der Nordakademie, No. 2013-01

Provided in Cooperation with:
Nordakademie - Hochschule der Wirtschaft, Elmshorn

ARBEITSPAPIERE DER NORDAKADEMIE

ISSN 1860-0360

Nr. 2013-01

Ein Referenzmodell für Schnittstellen

Hartmut Krasemann, Johannes Brauer, Christoph Crasemann

Januar 2013

Eine elektronische Version dieses Arbeitspapiers ist verfügbar unter
http://www.nordakademie.de/arbeitspapier.html

NORDAKADEMIE
HOCHSCHULE DER WIRTSCHAFT
THEORIE
PRAXIS

Köllner Chaussee 11
25337 Elmshorn
http://www.nordakademie.de

Ein Referenzmodell für Schnittstellen

Hartmut Krasemann, Johannes Brauer, Christoph Crasemann

28. Januar 2013

Inhaltsverzeichnis

Abstract

Schnittstellen zwischen Anwendungen werden heute in der Praxis meist auf Basis sehr fundamentaler Techniken auf die unterschiedlichsten Arten programmiert. Das führt zu einer großen Menge technischen Codes im Anwendungsprogramm, der heute in Projekten in der Regel jedesmal neu erfunden wird. Deshalb ist eine DSL für Schnittstellen erstrebenswert. Mit solch einer DSL kann die Anwendung von dem technischen Schnittstellencode befreit werden.

Bei der Entwicklung dieser DSL wurde schnell deutlich, dass die heute eingesetzen Schnittstellen-Techniken und Semantiken zu heterogen sind, um sie alle in einer DSL zusammenzuführen. Als Basis für eine Schnittstellen-DSL muss die Vielfalt der möglichen Schnittstellen-Semantiken durch ein solides Referenzmodell eingegrenzt werden. Solch ein Referenzmodell gibt es bisher nicht.

Wir schlagen daher ein Referenzmodell für Schnittstellen vor, das auf dem Command-Query-Muster auf Basis asynchroner Nachrichten aufbaut. Das Referenzmodell beschreibt erstens die Nachrichten und zweitens die Muster des Austauschs der Nachrichten zwischen den beteiligten Anwendungen, die beide mit einem DSL-Text spezifiziert werden können. Drittens beschreibt es das API der Schnittstelle für die Anwendung, das aus den DSL-Texten erzeugt wird. Die Inhalte der Nachrichten sind orthogonal zu den Nachrichten selbst und ihren Austauschmustern. Deshalb ist eine wesentliche Eigenschaft unseres Referenzmodells die Trennung der Nachrichteninhalte von den Nachrichten selbst. Für die Beschreibung der Inhalte mag man XML-Schema oder eine andere DSL einsetzen.

1 Einleitung

DSLs[1] sind ein vielversprechender Weg, um in der Programmierung die Abstraktionslücke zu schließen und dem Programmierer problemnahe Sprachen an die Hand zu geben[2,3].

In einer gut verstandenen und abgegrenzten Domäne ist das Konstruieren einer DSL nicht schwer, wie die *AuDSL* der Autoren für Harel Statecharts gezeigt hat[4]. Bei dem Versuch, eine DSL für Schnittstellen, die *SStDSL*, zu entwerfen[5], wurde schnell klar, dass in diesem Fall ein Referenzmodell fehlt.

Die Herausforderung war, erst einmal ein Referenzmodell für Schnittstellen festzulegen. Vorher lässt sich keine DSL konstruieren. Herausforderungen dieser Art dürften für die Programmierung mit DSLs typisch sein, ein fertig definiertes Referenzmodell wie für die Harel-Statecharts ist eher die Ausnahme.

Erst das Referenzmodell für Schnittstellen setzt uns in die Lage, ein konzises, schlankes API für die Anwendung und eine ebenso konzise und schlanke DSL für Schnittstellen, die *SStDSL*, zu definieren. Die Autoren haben gezeigt[6] , dass es mit solch einem Referenzmodell gelingt, die sehr verschiedenen fundamentalen Techniken der Schnittstellenprogrammierung vor dem Anwendungsprogrammierer vollständig zu verbergen.

[1] Domänenspezifische Sprachen – domain specific languages

[2] Brauer, Johannes ; Crasemann, Christoph ; Krasemann, Hartmut: Auf dem Weg zu idealen Programmierwerkzeugen – Bestandsaufnahme und Ausblick. In: *Informatik-Spektrum* 31 (2008), Dezember, Nr. 6, S. 580 – 590

[3] Fowler, Martin: *Domain-Spezific Languages*. Addison-Wesley, 2010

[4] Krasemann, Hartmut ; Brauer, Johannes ; Crasemann, Christoph: DSL mit Parser-Kobinatoren: Mit wenig Code zu einer Harel-Statechart-DSL. In: *OBJEKTspektrum* (2011), Juli/August, Nr. 04

[5] Krasemann, Hartmut ; Brauer, Johannes ; Crasemann, Christoph: SStDSL, eine DSL für Schnittstellen mit PetitParser / NORDAKADEMIE Hochschule der Wirtschaft. 2012 (04). – Forschungsbericht

[6] Krasemann, Hartmut ; Brauer, Johannes ; Crasemann, Christoph: SStDSL, eine DSL für Schnittstellen mit PetitParser / NORDAKADEMIE Hochschule der Wirtschaft. 2012 (04). – Forschungsbericht

2 Motivation

Im Bereich der Schnittstellenprogrammierung gibt es eine große Vielfalt von Semantiken und Techniken. Der Referenztext von Hohpe und Woolf[7] beschreibt die Semantik des Nachrichtenaustauschs im Wesentlichen auf der technischen oder Transport-Ebene, wie z. B. Nachrichtensysteme und Kanäle als asynchrone Alternative zu Aufrufschnittstellen oder die verschiedenen Möglichkeiten für Nachrichtenkanäle wie Punkt-Punkt-Verbindungen oder Veröffentlichen-Abonnieren. Weiter differenziert er Nachrichtenkanäle nach ihren Eigenschaften wie z. B. Persistierung und Nachrichten nach ihrer Art wie z. B. Command-, Document- und Event-Nachrichten.

Die Web Service Description Language (WSDL)[8] beschreibt eine Schnittstelle (interface) als eine Menge von Nachrichten mit ihren Austausch-Mustern (operations). Sie wird ergänzt durch die Web Service Notification (WSN)[9], die Schnittstellen nach dem Beobachtermuster beschreibt.

Auf dieser Basis ist die Zahl der möglichen Semantiken zu groß.

Command-Query-Separation[10] zielt darauf ab, Seiteneffekte in Programmen besser beherrschen zu können[11]. Das Referenzmodell greift diese Idee auf und verwendet Nachrichten, die dem Prinzip der Command-Query-Separation genügen. Der kleinstmögliche Satz an Nachrichten besteht aus

[7] Hohpe, Gregor ; Woolf, Bobby: *Enterprise Integration Patterns*. Pearson Education, 2004 (A Martin Fowler Signature Book)

[8] OMG: *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, Juni 2007. `http://www.w3.org/TR/wsdl20/`

[9] OASIS: *Web Service Notification*. `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn`. Version: 2006

[10] siehe Abschnitt 7.7.2 in Meyer, Bertrand: *Object-oriented Software Construction*. Prentice Hall, 1988 (C.A.R. Hoare Series)

[11] Mey88: „The ... distinction between commands and queries ... are essential for the proper development of large systems“

- einer Anfrage (Request),
- der zugehörigen Antwort (Reply), die im antwortenden System keinen Zustand ändert, und
- aus einem Kommando (Command), auf das hin das empfangende System etwas tun soll und seinen Zustand ändert.

Asynchrone Nachrichten wie die Commands können mit oder ohne *Zustellgarantie* verwendet werden. Zustellgarantie bedeutet, dass die Schnittstelle mit dem Quittieren des Sendens der Nachricht die Zustellung beim Empfänger zusichert.

Zustellgarantie heißt auch *send&forget*. Die Zustellung wird solange versucht, bis der Empfänger erreicht wird und die Abnahme quittiert. Ohne Zustellgarantie kann eine Nachricht z. B. verloren gehen, wenn der Empfänger zur Sendezeit gerade nicht erreichbar ist.

Die Zustellgarantie wird verwendet, wenn der Sender sich auf die Zustandsänderung verlassen will, die die Nachricht im Empfänger auslöst, ohne auf eine Quittung zu warten. Damit wird die Anwendungsebene von der Quittungsüberwachung befreit. Der Preis dafür ist die erforderliche Zwischenspeicherung der Nachricht in der Schnittstellenschicht. Periodische Nachrichten wie z. B. der aktuelle Kurs eines Schiffes können deshalb oft auch ohne Zustellgarantie verschickt werden – mit entsprechend höherer Performanz.[12]

[12] Wenn eine Anwendung höhere Performanz der Schnittstelle braucht, kann sie auch auf die Zustellgarantie verzichten, selbst Quittungsnachrichten definieren und das Eintreffen der Quittung überwachen.

Mit den drei oben genannten Nachrichten könnte man bereits jede Schnittstelle implementieren. Es fehlt allerdings noch eine Definition der Austauschmuster, die auf der Definition der Nachrichten aufsetzt, vgl. Abbildung 1.

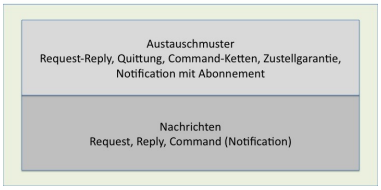

Abbildung 1: Austauschmuster auf der Basis von Nachrichten

Bei Request und Reply ist dies einfach: Antworten auf Anfragen sind synchron. Der Programmierer wird im API der Schnittstelle eine Request-Methode finden, die ihm unmittelbar die Antwort liefert.

Bei Commands gibt es zwei Möglichkeiten.

1. Der Sender erwartet vom Empfänger eine Antwort in Form eines anderen Commands. Dies kann eine Quittung sein oder eine andere anwendungsbezogene Nachricht. In diesem Fall ist die Zeitüberwachung der Schnittstelle in der Verantwortung der Anwendung.
2. Das Command wird mit Zustellgarantie gesendet. In dem Fall verlässt sich der Sender darauf, dass die Nachricht vom Empfänger verarbeitet wird.

Bisher haben wir ausschließlich 1:1-Kommunikation betrachtet. In ereignisgesteuerten Systemen bietet das Beobachtermuster[13] eine wirksame Entkopplungsstrategie an. Eine Änderungsnachricht oder *Notification* wird an alle potenziellen Empfänger gesendet. Ein Empfänger abonniert eine Notification, d. h. er entscheidet dynamisch über ihren Empfang. Hier muss der Empfänger einer Notification spezifizieren, ob die Notification mit oder ohne Zustellgarantie ausgeliefert werden soll.[14] Die Notification überträgt immer nur den geänderten Aspekt des Absenders als spezielles Schlüsselwort und ändert den Zustand des Empfängers nicht per se. Der Empfänger mag, so er will, Details der Änderungen des angegebenen Aspekts per Query erfragen.

[13] observer pattern, vgl. Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995

[14] Dies ändert in diesem Fall nicht die Semantik der Verarbeitung, aber ohne Zustellgarantie kann eine Notification verloren gehen. Damit verpflichtet der Empfänger sich auch, das Abonnement abzumelden, sobald er die Abnahme der Notifications nicht mehr garantieren kann.

Prinzipiell könnte ein Command als Nachrichtentyp für diese Notifications verwendet werden. Dies hätte allerdings zwei Nachteile:

- Wegen der unterschiedlichen Behandlung der Zustellgarantie müsste die Verwendung des Command als Notification beim Senden dennoch spezifiziert werden.
- Um das API für Notifications erzeugen zu können, muss die Verwendung des Command als Notification auch beim Empfänger spezifiziert werden.

Die Einführung einer speziellen vierten Nachricht für die Notification senkt deshalb die Kompliziertheit und verbessert die Lesbarkeit eines möglichen DSL-Textes.

FÜR DIE ÜBERTRAGUNG DER NACHRICHTEN kann ein Router (oder Broker) eingesetzt werden. Asynchrone Nachrichten werden heute oft im Kontext eines zentralen Routers verwendet. Dies geschieht i. A. wegen der besseren Administrierbarkeit eines zentralen Routers im Vergleich zu dezentraler Routing-Funktionalität. Alle bisher genannten Nachrichtentypen – auch die Notification mit ihrem Abonnement – können mit oder ohne Router benutzt werden.

Von der Nutzung eines Routers kann also abstrahiert werden. Damit wird die Frage, ob ein zentraler Router genutzt werden soll, zu einem Implementierungsdetail, das im Referenzmodell keine Rolle mehr spielt.

Genauso wird von der Synchronizität oder Asynchronizität der Übertragung selbst abstrahiert. Die technische Übertagung selbst kann mit dem Ziel einer möglichst guten Entkopplung vollständig asynchron erfolgen, während es auf der Anwendungsebene beide Semantiken gibt: synchron und asynchron.

3 Ziele

Das Referenzmodell für Schnittstellen soll ein möglichst breites Anwendungsgebiet erschließen. Dies bedeutet, dass sich mit ihm sowohl

- zustandslose Kommunikation organisieren läßt, als auch
- zustandsbehaftete Kommunikation, wie sie in verteiten Anwendungssystemen üblich ist.

In einer zustandslosen Kommunikation ist der Zustand des Kommunikationspartners vor und nach der Kommunikation gleich. Prototypisch für eine zustandslose Kommunikation können wir uns eine Internet-Sitzung vorstellen, in der der Client seinen Sitzungs-Zustand hält und mit jeder Anfrage (Request) alle notwendige Zustandsinformation zum Server schickt, die der zum Erzeugen der angefragten Seite braucht. Mit dem Reply vergisst der Server alle übertragene Zustandsinformation wieder.

Prototypisch für zustandsbehaftete Kommunikation können wir uns das verteilte Steuerungssystem eines Schiffes vorstellen, in dem die Kollisionswarnfunktion des Navigationssystems immer weiß,

mit welcher Geschwindigkeit das Schiff gerade läuft. In einem solchen System werden regelmäßig Notifications wie „Geschwindigkeit v“und gelegentlich Commands wie „Maschine stopp“gesendet.

Weiterhin soll das Referenzmodell eine möglichst kleine Anzahl verschiedener Nachrichtentypen beinhalten. Diese sind nach dem oben gesagten *Request*, *Reply*, *Command* und *Notification*.

Eine DSL auf Basis des Referenzmodells soll alle technischen Details vor dem Programmierer verbergen. Dies sind insbesondere die Kanäle, Kanaleigenschaften und ihre Implementierung oder die Eigenschaften eines Routers.

Zu den sichtbaren Kanaleigenschaften gehört die Zustellgarantie, i. e. die Zusicherung, dass eine erfolgreich gesendete Nachricht dem Empfänger so lange zugestellt wird, bis er sie abnimmt. Die Zustellgarantie wird mit der DSL spezifiziert, der Anwendungsprogrammierer braucht sich darum nicht zu kümmern.

Ein in der Praxis immer schwieriger Aspekt ist die Evolution von Schnittstellen. Oft müssen beteiligte Systeme gleichzeitig angepasst werden[15]. Oder eine Schnittstelle fällt partiell aus, bis die beteiligten Systeme die Änderungen nachvollzogen haben.

Hier hilft die Versionierung der Nachrichten. Beteiligte Systeme können die Version einer Nachricht prüfen und auf die Versionskennung reagieren.[16]

[15] Insbesondere muss die Anpassung auch gleichzeitig in Betrieb genommen werden.

[16] Dazu gehört auch, dass die Verarbeitung einer falschen Nachrichtenversion bereits abgelehnt wird, bevor sie das Anwendungsprogramm erreicht.

4 *Das Referenzmodell*

Wir beschreiben zunächst die vorgeschlagenen Nachrichtentypen, um im übernächsten Abschnitt auf die Austauschmuster des Referenzmodells einzugehen. DSL-Beispiele für die Spezifikation dieser Nachrichten zeigt der Anhang A.1.

4.1 Nachrichtentypen des Referenzmodells

Jede Nachricht trägt zwei Schlüsselworte. Das erste ist der Name der Nachricht, das zweite bezeichnet den Dateninhalt:

- Ein *Request* erfragt Information, d. h. einen bestimmten *Reply*, und zwar per vereinbarter Bedeutung des Namens und optional weiterer Daten.
- Ein *Reply* ist einem Request zugeordnet und trägt neben seinem Namen beliebige Daten.
- Ein *Command* trägt ebenfalls seine Bedeutung im Namen und optional weiterere Daten.
- Eine *Notification* enthält den ersten Aspekt der Änderung im Namen[17] und einen weiteren Aspekt[18] der Änderung mit vereinbarter Semantik im Daten-Schlüsselwort. Die Aspekte der Änderung[19] beziehen sich immer auf den Absender der Notification. Bei Requests und Commands dagegen sollte die Bedeutung des Namens oder des Daten-Schlüsselworts nicht vom Absender abhängen, obwohl dies prinzipiell auch möglich wäre.

[17] zum Beispiel den Namen eines geänderten Datums oder Objekts

[18] zum Beispiel den Wert des geänderten Datums oder den Namen des geänderten Attributs

[19] Im einfachsten Fall ist dies ein Datum und sein neuer Wert. In komplexeren Fällen muss der Empfänger der Notification anhand der geänderten Aspekte die eigentliche Änderung noch erfragen.

Der Dateninhalt selbst ist immer eine Bytefolge bzw. ein String. Dieser String muss vom Empfänger interpretiert werden. Heute wird dafür meist XML-Schema als Typbeschreibung eingesetzt[20]. Diese Typbeschreibung hat einen Namen, nämlich das Schlüsselwort für die Daten, das in der Spezifikation der Nachricht verwendet wird.

[20] Historisch überwiegen programmiersprachengebundene Typbeschreibungen für Intra-System-Kommunikation und EDI für Inter-System-Kommunikation

Bis auf diesen Zusammenhang ist allerdings die Typbeschreibung des Nachrichteninhalts orthogonal zu der Spezifikation der Nachrichten selbst und der Nachrichten-Muster, d. h. sie ist völlig unabhängig davon.

Damit wird klar, dass – anders als z. B. in den Web Service Spezifikationen[21] – die Spezifikation der Schnittstellen von der Spezifikation der Nachrichteninhaltstypen getrennt werden sollte. Für die Nachrichteninhalte gibt es auch bereits eine weit verwendete DSL: XML-Schema. Eine spezielle DSL für die Nachrichteninhalte erschlösse allerdings Vorteile in Bezug auf Lesbarkeit, Überschaubarkeit und Einfachheit.

[21] OMG: *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, Juni 2007. `http://www.w3.org/TR/wsdl20/`

4.2 Austauschmuster des Referenzmodells

Wir charakterisieren Kommunikationsmuster und Nachrichtentypen aus mehreren Blickwinkeln:

- Eine Nachricht geht entweder an genau einen Empfänger oder an eine Verteilerliste mit vielen Empfängern. Im zweiten Fall gibt es einen Dienst, der die Verteilerliste verwaltet und die Nachrichten an die eigentlichen Empfänger verteilt.
- Nachrichten sind synchron (der Sender wartet auf eine Reaktion) oder asynchron (der Sender arbeitet nach dem Senden weiter).
- Im Fall des asynchronen Sendens kann es eine Zustellgarantie geben. Zustellgarantie besagt, dass eine einmal gesendete Nachricht nicht verloren geht.

Wir erlauben die folgenden Austauschmuster aus Sicht der Anwendung:

- Ein *Request-Reply* ist synchron mit je einem Absender und Empfänger. Ein Request ändert beim Empfänger keinen Zustand und kann deshalb beliebig oft wiederholt werden. Es gibt eine Zeitüberwachung, d. h. einen Sendefehler, wenn die Antwort nicht rechtzeitig eintrifft. Ein Request braucht deshalb keine Zustellgarantie.
- Ein *Command* ist asynchron und geht an einen Empfänger. Ein Command löst eine Aktion beim Empfänger aus und ändert seinen Zustand. Ein weiteres Command kann als Antwort spezifiziert werden, sei es als Teil einer Command-Kette oder als einfache Quittung. In dem Fall obliegt dem Sender die Überwachung des Eintreffens der Antwort. Der Sender muss aber nicht auf eine Antwort warten (send&forget). In dem Fall kann er eine Zustellgarantie in Anspruch nehmen.
- Eine *Notification* geht an eine Verteilerliste und ist als Spezialfall des Commands ebenfalls asynchron. Sie enthält Information

über eine Änderung des Absenders. Die Verteilerliste bietet einen Abonnement-Dienst an. Ein potenzieller Empfänger abonniert die ihn interessierenden Nachrichten, mit oder ohne Zustellgarantie. Bei Interesse erfragt der Empfänger Details der Änderung per Anfrage (Request-Reply) beim ursprünglichen Sender.

Ein Beispiel für die DSL-Spezifikation der Austauschmuster zeigt Anhang A.2.

Mit dem Request-Reply-Muster läßt sich eine zustandsfreie Schnittstelle implementieren. In den Daten, die mit dem Request mitgeschickt werden, sind dann alle erforderlichen Zustandsinformationen enthalten.

Steuerungssysteme brauchen ebenfalls Request-Reply, aber vor allem Commands. Mit Commands werden explizite Abläufe über Teilsystemgrenzen hinweg gesteuert. Steuerungssysteme können aber auch Notifications nutzen, die sie aus der Umwelt beziehen oder selbst in Teilen erzeugen. Prominente Steuerungssysteme wie z. B. das des Containerterminals Altenwerder in Hamburg nutzen genau und ausschließlich diese drei Muster.[22]

[22] KRASEMANN, Hartmut ; SPINDEL, Ulrich: Softwarearchitektur für einen Containerterminal. In: *Tagungsband Net.ObjectDays*, Netobjectdays Forum, September 2003

4.3 API für das Anwendungsprogramm

Aus dem Programmtext einer geeigneten Schnittstellen-DSL, bestehend aus der Spezifikation der Nachrichten und der Austauschmuster, kann das API für die Anwendung vollständig erzeugt werden.

Da in der Anwendung die Namen der Nachricht nicht ein zweites Mal deklariert werden sollen, enthält das API nachrichtenspezifische Methoden. Zum API gehören

- je eine Sende-Methode pro Command und Notification, diese kehren nach dem Senden ohne Antwort zurück;
- je eine Sende-Methode pro Request, diese antworten mit dem Inhalt des jeweiligen Reply oder einer Exception bei Timeout;
- je eine Hook-Methode pro empfangener Nachricht, die von der Laufzeitmaschine beim Empfang der Nachricht aufgerufen wird;
- zwei Methoden zum Abonnieren und Abbestellen von Notifications.

Die Anwendung ruft die nachrichtenspezifische Sendemethode mit dem Empfänger[23] und dem Nachrichteninhalt auf. Die Version der Nachricht und eine eventuelle Zustellgarantie von Commands sind mit der DSL spezifiziert.

[23] Weder Nachrichteninhalt noch Empfänger sind mit der DSL spezifiziert. Die Anwendung muss die URL sowie den Port bzw. die Queue des Empfängers bzw. Abonnement-Dienstes kennen.

Beim Empfang einer Nachricht wird die Hook-Methode von der Schnittstelle aufgerufen. In der Hook-Methode sind für den Anwendungsprogrammierer neben dem Nachrichteninhalt sowohl die Nachrichtenversion als auch der Sender verfügbar. Die Hook-Methode eines Requests muss mit dem Inhalt des zugehörigen Reply antworten.

Die Hook-Methoden für empfangene Nachrichten sind die einzigen generierten Elemente, in die der Anwendungsprogrammierer Code hineinschreibt.

Die Autoren zeigen ein konkretes, in Smalltalk implementiertes Beispiel für die SStDSL.[24] Anhang A.3 zeigt ein API, das aus dem DSL-Text für Nachrichten (Anhang A.1) und für die Austauschmuster (Anhang A.2) erzeugt werden kann.

[24] KRASEMANN, Hartmut ; BRAUER, Johannes ; CRASEMANN, Christoph: SStDSL, eine DSL für Schnittstellen mit PetitParser / NORDAKADEMIE Hochschule der Wirtschaft. 2012 (04). – Forschungsbericht

5 *Eigenschaften und Anmerkungen*

COMMANDS können von Anwendung zu Anwendung beliebig verkettet werden, sie orchestrieren damit verteilte Verarbeitung. Diese Sequenzen sind Teil der Anwendungssemantik, die nicht mit einer DSL spezifiziert werden kann.[25]

Jedes einzelne Command[26] hat die transaktionalen Eigenschaften der atomaren, konsistenten und isolierten Verarbeitung. Eine Kette von Nachrichten dagegen kann partiell fehlschlagen und verliert damit die Garantie der Konsistenz. Eine Anwendung, die auf diesem Referenzmodell aufbaut, muss deshalb ihre Transaktionen auf jeweils ein Command ausführen oder zusätzliche Konsistenzmechanismen implementieren.

Häufig kommen Anfragen vor, die in der antwortenden Anwendung Zustand ändern wie z. B. eine Reservierungsanfrage. Solch eine Anfrage kann nicht als Request-Reply, sie muss als Folge zweier Commands spezifiziert werden.

[25] In der Praxis werden die zwischen den beteiligten Systemen auftretenden Sequenzen von Commands höchsten teilweise spezifiziert.

[26] aber auch jede andere Nachricht

NOTIFICATIONS unterrichten den Empfänger darüber, dass sich ein bestimmter Aspekt des Absenders geändert hat. Der Aspekt wird in zwei Teilen benannt: mit dem Namen der Nachricht und einem Schlüsselwort. Die Notification trägt i. A. nicht die geänderten Daten, die muss der Empfänger erfragen, wenn er sie braucht.

Dieses Prinzip erscheint auf den ersten Blick umständlich, werden in vielen Fällen doch drei Nachrichten gebraucht, wo eine ausreichen würde. Dennoch hat es sich in der Realität bewährt, da die Notifications unvermeidbar viel öfter gesendet werden als die geänderten Daten gebraucht werden[27]. So ist das Beobachtermuster z. B. als *update-change* eines der Grundmuster des Smalltalk-Systems[28].

[27] Z. B. führt eine Notification beim Empfänger lediglich zum Setzen eines Dirty-Flags. Erst wenn die Anwendung das Datum verwendet, wird der aktuelle Wert erfragt.

[28] GOLDBERG, Adele ; ROBSON, David: *Smalltalk-80: The Language*. Addison-Wesley Publishing Company, 1989

EINE ZUSTELLGARANTIE vereinfacht die Schnittstellenprogrammierung sehr. Wenn Commands eine Zustellgarantie tragen, dann darf sich der Sender darauf verlassen, dass sie beim Empfänger auch ankommen. Er ist damit von allen Handshake-Mechanismen entlastet. Deshalb sieht unser Referenzmodell vor, dass Commands und Notifications eine Zustellgarantie tragen können. Für jedes Command kann dies in den Austausch-Mustern des DSL-Texts einzeln spezifiziert werden. Für Notifications spezifiziert dies der potenzielle Empfänger.

Synchrone Request und Reply-Nachrichten brauchen keine Zustellgarantie, für sie wird eine Timeout-Zeit spezifiziert.

Damit gibt es die folgenden Fehlermöglichkeiten bzw. Exceptions zur Laufzeit:

- für alle Nachrichten: Fehler beim Senden,
- für einen Request: Timeout des Reply.

EINE NACHRICHT KANN IN MEHREREN VERSIONEN vorliegen. Für zu sendende Nachrichten wird deshalb im DSL-Text die Version spezifiziert, die gesendet werden soll. Für zu empfangende Nachrichten werden im DSL-Text alle Versionen, die verarbeitet werden, in einer Versionsliste spezifiziert. Request-Reply-Paare haben eine gemeinsame Version.

Damit gestaltet sich eine Schnittstellenevolution z. B. für ein Command oder eine Notification in vier Schritten:

1. Es wird eine neue Version der Nachricht spezifiziert.
2. Der Empfänger verarbeitet neben der alten auch die neue Version der Nachricht.
3. Der Sender sendet die neue Version.
4. Der Empfänger stellt die Verarbeitung der alten Version der Nachricht ein.

Die beiden Nachrichten eines Request/Reply-Paares ändern sich gemeinsam. Es ergeben sich die folgenden Evolutionsschritte[29]:

1. Es wird eine neue Version des Request-Reply-Paares spezifiziert.
2. Der Server verarbeitet neben der alten auch die neue Version des Request. Auf die neue Version des Request sendet er die neue Version des Reply.
3. Der Client sendet die neue Version des Request und verarbeitet die neue Version des Reply.
4. Der Server stellt die Verarbeitung der alten Version des Request ein.

Beim jeweiligen Empfänger, im zweiten Beispiel sowohl Client als auch Server, ist die parallele Verarbeitung von zwei verschiedenen Versionen einer Nachricht erforderlich. Folglich muss die Versionsinformation, die originär zu den Umschlagsdaten gehört[30], auch dem empfangenden Anwendungsprogramm übergeben werden. Alternativ kann natürlich auch jeweils eine neue Nachricht definiert werden. Dafür gelten die gleichen Evolutionsschritte.

[29] Der *Client* sendet den Request, der *Server* antwortet mit dem Reply

[30] Umschlagsdaten sind die Daten, die sozusagen auf dem Briefumschlag stehen. Die funktionalen Komponenten des Nachrichtensystems haben nur Zugriff auf die Umschlagsdaten, nicht aber auf den Inhalt der Nachrichten.

6 Zusammenfassung und Ausblick

Das vorliegende Referenzmodell für Schnittsstellen ist bewusst einfach gehalten und auf das Notwendige reduziert. Insbesondere bietet es keine Ansatzpunkte für eine anwendungsspezifische Optimierung des Nachrichtenverkehrs.

Wir haben vier Nachrichten mit festgelegter Semantik (synchron und asynchron) definiert. Alle Nachrichten tragen eine Versionskennung für die Schnittstellenevolution. Die asynchronen Nachrichten können mit oder ohne Zustellgarantie ausgetauscht werden. Diese vier Nachrichten reichen aus, die Kommunikation sowohl für Informationssysteme als auch für reaktive, ereignisgesteuerte Systeme zu entwerfen.

Von der Schnittstellentechnik selbst kann die Anwendung mit diesem Referenzmodell vollständig befreit werden. Eine gegebene Schnittstelle lässt sich mit jeder Technik, sei es FTP oder JMS oder irgendetwas anderes, implementieren.

Mit dem Entwurf und der Implementierung einer DSL für Schnittstellen, der SStDSL[31], wurde ein „proof-of-concept“ erbracht. Das hier vorgestellte Referenzmodell für Schnittstellen erwuchs aus den Arbeiten zur SStDSL.

[31] KRASEMANN, Hartmut ; BRAUER, Johannes ; CRASEMANN, Christoph: SStDSL, eine DSL für Schnittstellen mit PetitParser / NORDAKADEMIE Hochschule der Wirtschaft. 2012 (04). – Forschungsbericht

Literatur

[1] BRAUER, Johannes ; CRASEMANN, Christoph ; KRASEMANN, Hartmut: Auf dem Weg zu idealen Programmierwerkzeugen – Bestandsaufnahme und Ausblick. In: *Informatik-Spektrum* 31 (2008), Dezember, Nr. 6, S. 580 – 590

[2] FOWLER, Martin: *Domain-Spezific Languages*. Addison-Wesley, 2010

[3] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995

[4] GOLDBERG, Adele ; ROBSON, David: *Smalltalk-80: The Language*. Addison-Wesley Publishing Company, 1989

[5] HOHPE, Gregor ; WOOLF, Bobby: *Enterprise Integration Patterns*. Pearson Education, 2004 (A Martin Fowler Signature Book)

[6] KRASEMANN, Hartmut ; BRAUER, Johannes ; CRASEMANN, Christoph: DSL mit Parser-Kobinatoren: Mit wenig Code zu einer Harel-Statechart-DSL. In: *OBJEKTspektrum* (2011), Juli/August, Nr. 04

[7] KRASEMANN, Hartmut ; BRAUER, Johannes ; CRASEMANN, Christoph: SStDSL, eine DSL für Schnittstellen mit PetitParser / NORDAKADEMIE Hochschule der Wirtschaft. 2012 (04). – Forschungsbericht

[8] KRASEMANN, Hartmut ; SPINDEL, Ulrich: Softwarearchitektur für einen Containerterminal. In: *Tagungsband Net.ObjectDays*, Netobjectdays Forum, September 2003

[9] MEYER, Bertrand: *Object-oriented Software Construction*. Prentice Hall, 1988 (C.A.R. Hoare Series)

[10] OASIS: *Web Service Notification*. `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn`. Version: 2006

[11] OMG: *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, Juni 2007. `http://www.w3.org/TR/wsdl20/`

A Anhang: DSL-Beispiel

A.1 DSL-Text für die Nachrichtenspezifikation

Wir betrachten ausschnittsweise die Kollisionswarnung auf einem Schiff, die unter anderem mit der Maschine und dem Navigationssystem des Schiffes kommuniziert. Dies geschieht mit den Nachrichten in Abbildung 2:

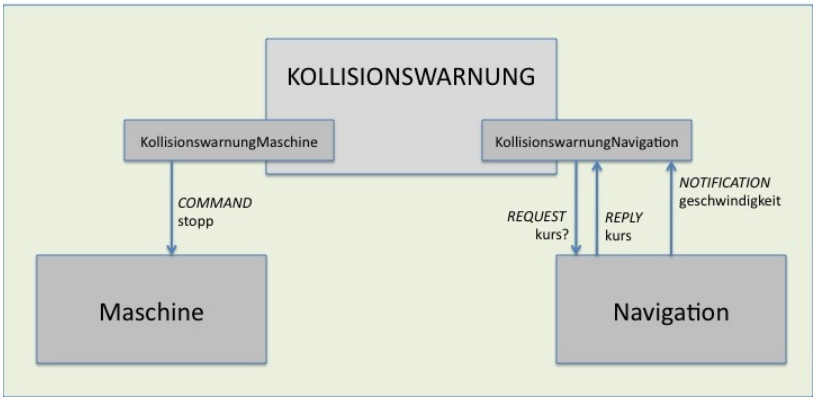

Abbildung 2: Kleines Beispiel für DSL-spezifizierte Schnittstellen

Dafür spezifizieren wir folgende Nachrichten:[32]

[32] Die in kursiven *GROSSBUCHSTABEN* geschriebenen Worte sind Schlüsselworte der DSL.

```
MESSAGES (
   COMMAND stopp
      VERSION 1.0
      DATA noData
   REQUEST kurs?
      VERSION 1.0
      DATA noData
   RESPONSE kurs
      VERSION 1.0
      DATA kursInGrad
   NOTIFICATION geschwindigkeit
      VERSION 1.0
      DATA einBetrag
```

A.2 DSL-Text für die Spezifikation der Austauschmuster

Die Austauschmuster der Schnittstelle werden für jeden Kommunikationspartner einzeln spezifiziert.

```
INTERFACE Maschine
   VERSION 1.0
   RECEIVE zurMaschine (     /* zurMaschine = Empfängerqueue */
      COMMAND stopp VERSIONS [0.9 1.0]  )

INTERFACE Navigation
   VERSION 1.0
   SEND (
      NOTIFICATION geschwindigkeit VERSION 1.0  )
   RECEIVE zurNavigation (    /* zurNavigation = Empfängerqueue */
     REQUEST kurs? VERSIONS [0.9 1.0] RESPONSE kurs   )

INTERFACE KollisionswarnungMaschine
   VERSION 1.0
   SEND (
      COMMAND stopp VERSION 1.0 GUARANTEED ja ) /* Zustellgarantie */

INTERFACE KollisionswarnungNavigation
   VERSION 1.0
   SEND (
      REQUEST kurs? VERSION 1.0 RESPONSE kurs TIMEOUT 100 )
   RECEIVE zurKWN (            /* zurKWN = Empfängerqueue */
      NOTIFICATION geschwindigkeit VERSIONS [1.0] GUARANTEED ja )
```

A.3 API zu den vorstehenden DSL-Texten

Aus den obenstehenden DSL-Texten können für jedes *INTERFACE* die folgenden API-Methoden z. B. für Smalltalk erzeugt werden. Wir geben die Smalltalk-Methodenkategorien (methodsFor:) mit an, weil sich daraus bereits die Verwendung der Methoden erkennen läßt. *Receive hooks* sind die einzigen Methoden, die für den Empfang der Nachrichten ausprogrammiert werden müssen, *sends* und *requests* sind Methoden, die zum Senden aufgerufen werden, *subscribing* sind Methoden, mit denen Notifications abonniert werden.

```
Maschine methodsFor: 'DSL receive hooks'
stopp: noData from: aSender version: aVersion
   "(command) hook method generated by DSL"
   "please define your code"

Navigation methodsFor: 'DSL sends'
geschwindigkeit: einBetrag to: aSubscriberService
   "(notification) generated by DSL
   call this method from your code - do not change"
   self send: (self newMsg: #geschwindigkeit content: einBetrag)
          to: aSubscriberService

Navigation methodsFor: 'DSL receive hooks'
kurs?: noData from: aSender version: aVersion
   "(request) hook method generated by DSL"
   "please define your code - answer aKursInGrad "
   ^'aKursInGrad'
```

```
KollisionswarnungMaschine methodsFor: 'DSL sends'
stopp: noData to: aReceiver
   "(command) generated by DSL
   call this method from your code - do not change"
   self send: (self newMsg: #stopp content: noData) to: aReceiver

KollisionswarnungNavigation methodsFor: 'DSL requests'
kurs?: noData to: aReceiver
   "(request) generated by SStDSL
   - answers the response content of message #kurs: aKursInGrad
   call this method from your code - do not change"
   ^self request: (self newMsg: #kurs? content: noData) to: aReceiver

KollisionswarnungNavigation methodsFor: 'DSL receive hooks'
geschwindigkeit: einBetrag from: aSender version: aVersion
   "(notification) hook method generated by DSL"
   "please define your code"

KollisionswarnungNavigation methodsFor: 'DSL subscribing'
subscribe: aMessage aspect: anAspect
   "(notification) subscribe method generated by DSL
   call this method from your code - do not change"
unsubscribe: aMessage aspect: anAspect
   "(notification) unsubscribe method generated by DSL
   call this method from your code - do not change"
```