# Ein Ereignis-basiertes Modell zur Formalisierung von Request-Reply Objektinteraktionen

Raúl Monge, Franz J. Hauck

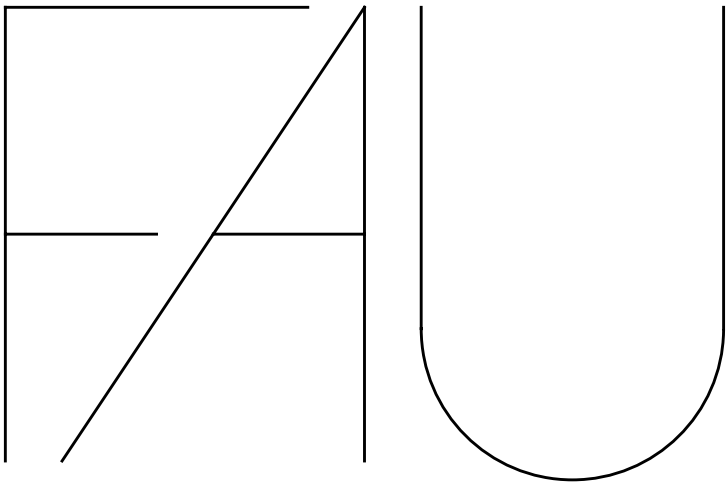
September 1993 TR-I4-10-93

Technical Report

Computer
Science Department

Operating Systems — IMMD IV

Friedrich-Alexander-University
Erlangen-Nürnberg, Germany



Dieser Report wurde auch veröffentlicht in:

Raúl Monge, Franz J. Hauck: “Ein Ereignis-basiertes Modell zur Formalisierung von Request-Reply Objektinteraktionen”; In: H. Wedekind [Hrsg.] *Verteilte Systeme*, Grundl. und zukünft. Entw. aus d. Sicht d. SFB 182; Bibliographisches Institut, Zürich; erscheint 1994

# Ein Ereignis-basiertes Modell zur Formalisierung von *Request-Reply* Objektinteraktionen

Raúl Monge[1], Franz J. Hauck

E-Mail: rmonge@inf.utfsm.cl, hauck@informatik.uni-erlangen.de

Teilprojekt B2
Entwurf und Implementierung eines an Hardwarearchitektur und Aufgabenklassen adaptierbaren Multiprozessorsystems

## Zusammenfassung

Die häufigste Form der möglichen Interaktionen zwischen verteilten Objekten ist die *Client-Server*- oder *Request-Reply*-Interaktion. Sie wird Beauftragung genannt. Um die Ausprägungen, d.h. die möglichen Protokolle dieser Art von Objektinteraktion besser klassifizieren und untersuchen zu können, wurde ein Modell für die Beauftragung entworfen, das auf Ereignisse während des Kommunikationsvorgangs basiert. Es wird das Modell mit den möglichen Ereignissen definiert und darauf aufbauend werden mehrere Ausprägungen von Interaktionsformen formal erfaßt. Kommunikationsprimitive erlauben die Beschreibung der Interaktionsformen eines bestimmten Objektsystems und verstecken die Ereignisse des Modells in übersichtlichen programmiersprachlichen Einheiten.

## 1 Einführung

Objekte und Objektmodelle eignen sich besonders gut zur Modellierung verteilter Systeme, da sich in einem Objektsystem bereits von Anfang an Einheiten der Verteilung erkennen lassen. Diese kommunizieren miteinander auf Nachrichtenbasis. Die Form der möglichen Interaktionen zwischen den Einheiten eines verteilten Systems ist zwar nicht unmittelbar durch das objektorientierte Paradigma festgelegt, doch werden in solchen Modellen meist Interaktionen in Form von Beauftragung verwendet. Darunter versteht man *Request-Reply*- oder *Client-Server*-Interaktionen. Es besteht nun der Wunsch nach formaler Grundlage für die Definition verschiedener Protokolle und Begriffe im Zusammenhang mit Beauftragung.

Diese Arbeit entstand im Rahmen des PM-Projekts als Folge von Untersuchungen, die sich mit der Kommunikation in einem verteilten, objektbasierten System befaßt haben [Mong92]. Ein wichtiges Ziel war es, ein formales Modell zu entwickeln, um verschie-

1. Aktuelle Adresse: Depto. de Informática, Universidad Santa María, Casilla 110-V, Valparaíso, Chile

dene Ausprägungen von Objektinteraktionen genauer zu beschreiben. Auf dieser Basis können Auftragsprotokolle dann besser verstanden und in einer geeigneter Weise entworfen werden.

Zunächst wird ein Modell für die Beauftragung vorgestellt. Dabei beschränken wir uns auf die Interaktionsform der Delegierung, d.h. Auftraggeber und Auftragnehmer sind immer verschiedene Aktivitätsträger und Aktivitätsträger bleiben den Objekten zugeordnet. Es folgt eine Darstellung des Ereignis-basierten Auftragsmodells, mit dessen Hilfe im darauf folgenden Kapitel verschiedene Protokolle diskutiert werden. Schließlich wird die Arbeit zusammengefaßt und ein Ausblick gegeben.

# 2 Die Beauftragung

Ein Modell für Interaktionen zwischen Objekten ist die Beauftragung: Objekte können hierbei andere Objekte mit bestimmten Aufgaben beauftragen, wobei die Abarbeitung eines Auftrags jedoch nicht unbedingt Ergebnisse zurückliefern muß. Dieses Modell unterstützt die Kapselung der Objekte, da Objektinteraktionen nur durch den Austausch von Aufträgen und eventuell von Ergebnissen stattfinden.

## 2.1 Das Auftragsmodell

In der Beauftragung findet eine Objektinteraktion zwischen einem Auftraggeber und mindestens einem Auftragnehmer statt. Es sind zwei Phasen der Interaktion erkennbar, nämlich die *Beauftragungs-* und die *Beantwortungsphase*. Eine solche Interaktion wird als *Auftragstransaktion* bezeichnet. Die entsprechende Botschaften dieser Phasen werden als Auftrag bzw. als Antwort definiert. Zwischen beiden Phasen findet in einem Auftragnehmer die Bearbeitung des Auftrags statt (siehe Abb. 2.1).

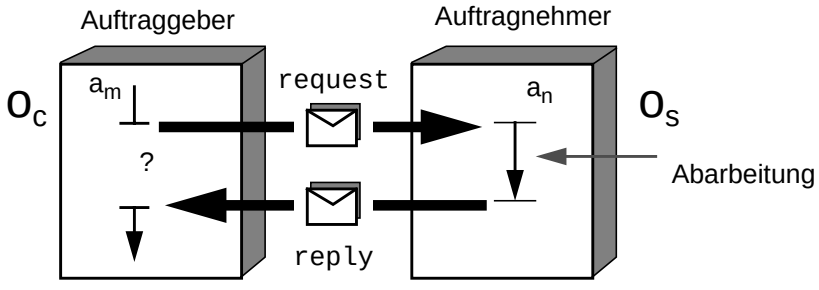



Abb. 2.1 Auftragsmodell

Wenn man sich nach dem Verhalten der Aktivitätsträger während einer Objektinteraktion fragt, dann lassen sich im wesentlichen zwei Modelle unterscheiden:

- **Selbstbedienung**: Der Aktivitätsträger, der im Auftraggeber einen Auftragnehmer beauftragt, wandert zusammen mit dem Auftrag zum Auftragnehmer, wo er den Auftrag abarbeitet, und kehrt dann wieder zum Auftraggeber zurück; eventuell mit einer Antwort.

- **Delegierung**: Der Aktivitätsträger, der im Auftraggeber einen Auftragnehmer beauftragt, delegiert die Abarbeitung des Auftrags an einen anderen Aktivitätsträger, der sich im Auftragnehmer befindet.

Ein Ziel verteilter Systeme ist es, die potentielle Nebenläufigkeit zu erhöhen, um somit die Ressourcen im System besser auszunutzen und dadurch die Leistung der Anwendungen zu verbessern. Die Kommunikationsmechanismen, die das System auf der Programmierebene bietet, können hier unterstützend wirken. Unter der Zielsetzung möglichst hoher Nebenläufigkeit erscheint die Verwendung von *Selbstbedienung* wenig sinnvoll, da diese eine Sequentialisierung zur Folge hat. Die Delegierung dagegen ermöglicht Möglichkeiten, die Nebenläufigkeit im System zu fördern:

- **Mehrfache Beauftragung**: Ein Auftrag wird gleichzeitig von einem Auftraggeber an mehrere Auftragnehmer übergeben. *Multicast*-Protokolle unterstützen diese Form in geeigneter Weise.
- **Nicht-blockierende Delegierung**: Der Auftraggeber kann parallel mit der Abarbeitung des Auftrags durch einen Auftragnehmer weiterarbeiten und insbesondere weitere Auftragstransaktionen anstoßen. Durch das Entgegennehmen der Antwort eines Auftrags wird der Auftraggeber mit dem Ende der Auftragsbearbeitung synchronisiert.
- **Mitteilende Beauftragung**: Der Auftraggeber schickt einen Auftrag ab, von dem er keine Antwort erwartet. Die Synchronisation kann hier abgeschwächt werden. Im Auftragsmodell fällt die gesamte Beantwortungsphase weg.

Bei blockierender Delegierung bleibt der Auftraggeber solange blockiert, bis der Auftragnehmer die Antwort zurückgeschickt hat. Diese Interaktionsform modelliert den Fall der Selbstbedienung.

In diesem Modell bleiben also Aktivitätsträger in ihren Objekten verankert und Objekte interagieren nur durch den Austausch von Botschaften. Ein Objekt kann jedoch mehrere Aktivitätsträger besitzen, die verschiedenen Auftragstransaktionen zuzuordnen sind. Das verwendete Auftragsprotokoll wird in gewisser Weise das Verhalten der Aktivitätsträger bestimmen, die in einer Objektinteraktion verwickelt sind.

## 2.2 Kommunikationsereignisse

In einer Auftragstransaktion – kurz Transaktion – können zwei Kommunikationsphasen unterschieden werden: Beauftragungs- und Beantwortungsphase. In jeder Kommunikationsphase lassen sich drei wichtige Ereignisse erkennen:

- **Abgabe**: Die Botschaft wird vom Sender-Objekt an das Kommunikationssystem zum Senden abgegeben.
- **Empfang**: Die Botschaft kommt am Empfänger-Objekt an.
- **Zustellen**: Die Botschaft wird an einen Aktivitätsträger des Empfänger-Objekt für ihre Bearbeitung zugestellt.

Einige nachrichtenbasierte Kommunikationsmechanismen lassen es zu, daß der Empfänger dem Kommunikationssystem explizit seine Bereitschaft für den Empfang einer Nachricht mitteilt. Es wird dann von einem expliziten Empfang gesprochen, der hier als viertes Ereignis eingeführt und als "**Anfordern**" bezeichnet wird.

# 3 Ein Ereignis-basiertes Auftragsmodell

Um die Ausprägungen, d. h. die möglichen Protokolle dieser Art von Objektinteraktionen besser klassifizieren und untersuchen zu können, wurde ein Modell für die Beauftragung entworfen. Das Modell definiert Ereignisse auf der Auftraggeberseite (Client) und auf der Auftragnehmerseite (Server), die im Zuge einer Interaktion auftreten. Diese Ereignisse stehen in einem kausalen Zusammenhang, d. h. sie müssen sequentiell bearbeitet werden.

## 3.1 Ereignisse

Ausgehend von den drei Kommunikationsereignisse und der Zulassung eines expliziten Anforder-Ereignisses kommt man zu folgendem Modell für die Beauftragung:

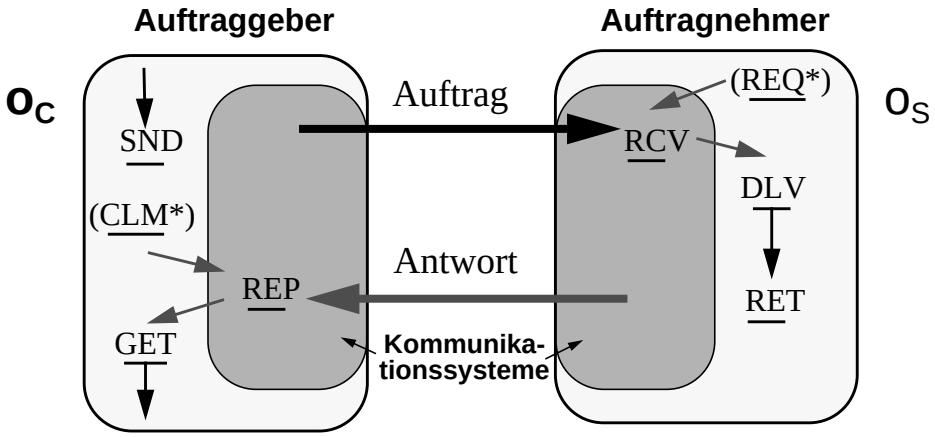



Abb. 3.1 Ereignisse in der Beauftragung

Die Namen der definierten Ereignisse der Abb. 3.1 haben folgende Bedeutung:

| **Beauftragungsphase** | | **Beantwortungsphase** | |
|---|---|---|---|
| SND | Abgabe des Auftrags | RET | Abgabe der Antwort |
| RCV | Empfang des Auftrags | REP | Empfang der Antwort |
| DLV | Zustellen des Auftrags | GET | Zustellen der Antwort |
| REQ | explizites Anfordern des Auftrags | CLM | explizites Anfordern der Antwort |

Die Ereignisse SND, REP, CLM und GET finden auf der Auftraggeberseite, die Ereignisse REQ, RCV, DLV und RET auf der Auftragnehmerseite statt. Die Beauftragungs- und Beantwortungsphase laufen völlig symmetrisch ab.

## 3.2 Definitionen

Die wichtigsten Definitionen im Modell können folgendermaßen vereinfacht dargestellt werden:

| | |
|---|---|
| $O$ | Menge der Objekte; typisches Element: $o_i$ |
| $A$ | Menge der Aktivitätsträger; typisches Element: $a_m$ |
| $R$ | Menge der Transaktionen; typisches Element: $r_p$ |
| $E$ | Menge der Ereignisse des Objektsystems |
| $H(o_i)$ | Ereignisspur im Objekt $o_i$ |
| $H(a_m)$ | Ereignisspur des Aktivitätsträgers $a_m$ |
| $H\_Wrk(r_q, o_j)$ | Bearbeitungsspur des Auftrags $r_q$ in Auftragnehmer $o_j$ |
| $Servers(r_q)$ | Menge der Auftragnehmer (Teilmenge von $O$) |
| $Workers(r_q)$ | Menge der Bearbeiter (Aktivitätsträger), die den Auftrag der Transaktion $r_q$ bearbeiten (Teilmenge von $A$) |

Ereignisse können u.a. mit folgenden Relationen verknüpft werden, die zur Definition der später vorgestellten Protokolle benötigt werden:

| | |
|---|---|
| $e \rightarrow f$ | Ereignis f folgt kausal abhängig auf Ereignis e |
| $e \mid\mid f$ | Ereignis e findet parallel zu f statt |
| $e \rightarrowtail f$ | Ereignis f folgt direkt auf Ereignis e |

Mit Hilfe der definierten Ereignisse in der Beauftragung und mit Berücksichtigung von mehreren Auftragnehmern ist es möglich, die Ereignisse der Beauftragung folgendermaßen eindeutig zu identifizieren:

| | |
|---|---|
| $snd(r_q)$ | Ereignis SND der Transaktion $r_q$ |
| $rcv(r_q, o_j)$ | Ereignis RCV der Transaktion $r_q$ in Auftragnehmer $o_j$ |
| $dlv(r_q, o_j)$ | Ereignis DLV der Transaktion $r_q$ in Auftragnehmer $o_j$ |
| $ret(r_q, o_j)$ | Ereignis RET der Transaktion $r_q$ in Auftragnehmer $o_j$ |
| $rep(r_q, o_j)$ | Ereignis REP der Transaktion $r_q$ im Zusammenhang mit der Antwort von Auftragnehmer $o_j$ |
| $get(r_q, o_j)$ | Ereignis GET der Transaktion $r_q$ im Zusammenhang mit der Antwort von Auftragnehmer $o_j$ |

Wenn ein expliziter Empfang eines Auftrags bzw. einer Antwort zugelassen wird, dann müssen folgende zwei Funktionen definiert werden:

| | |
|---|---|
| $req(r_q, o_j)$ | Ereignis REQ der Transaktion $r_q$ in Auftragnehmer $o_j$ |
| $clm(r_q, o_j)$ | Ereignis CLM der Transaktion $r_q$ im Zusammenhang mit der Antwort von Auftragnehmer $o_j$ |

## 3.3 Kausale Relationen

Die Ereignisse, die im Objektsystem erzeugt werden, sind kausal verknüpft, wenn ein möglicher Informationsfluß zwischen ihnen möglich ist [Lamp78]. Die folgenden Ausdrücke definieren im Modell die drei möglichen kausalen Abhängigkeiten, die durch den Aktvitätsträger selbst, die Kommunikation zwischen Objekten und durch expliziten Empfang oder explizites Anfordern entstehen:

a) Ereignisse, die vom selben Aktivitätsträger erzeugt wurden, sind trivialerweise miteinander kausal verknüpft. Ereignisse sind kausal abhängig von früheren Ereignissen des gleichen Aktivitätsträgers:

$$\forall\, a_m \in A\,,\ \exists\, e_i, e_j \in E : e_i \in H(a_m) \text{ und } e_j \in H(a_m) \text{ und } e_i < e_j \Rightarrow e_i \rightarrow e_j$$

Zwei Ereignisse stehen dabei in der Relation „<“ wenn das erste Ereignis zeitlich vor dem zweiten Ereignis stattfindet.

b) Kommunikation zwischen Objekten:

$$\forall\, r_p \in R\,,\ \forall\, o_j \in O :\quad o_j \in \text{Workers}(r_p) \Rightarrow \quad snd(r_p) \rightarrow dlv(r_p, o_j) \rightarrow ret(r_p, o_j) \rightarrow get(r_p, o_j)$$

c) Expliziter Empfang bzw. Anfordern:

$$\forall\, r_p \in R\,,\ \forall\, o_j \in O :\quad o_j \in \text{Workers}(r_p) \Rightarrow \quad req(r_p, o_j) \rightarrow dlv(r_p, o_j) \text{ und } clm(r_p, o_j) \rightarrow get(r_p, o_j)$$

Diese drei Relationen gelten für alle möglichen Auftragsprotokolle, die durch das Modell definierbar sind. Einige speziellere Protokolle können jedoch zusätzliche kausale Abhängigkeiten definieren oder auf bestimmte Ereignisse verzichten, wie im folgenden Kapitel zu sehen sein wird.

# 4 Auftragsprotokolle

Die Semantik der Auftragsprotokolle kann von der Seite des Auftraggebers und des Auftragnehmers spezifiziert werden. Eine kleine Menge von Primitiven kann definiert werden, um Auftragsprotokolle mit unterschiedlichen Semantiken auf der Seite des Auftraggebers und des Auftragnehmers zu benutzen.

## 4.1 Definitionen auf der Auftraggeberseite

### Blockierende Delegierung

Um die Semantik eines Auftragsprotokolls aus der Seite des Auftraggebers zu spezifizieren, muß zuerst festgelegt werden, wie sich die Aktivitätsträger während einer Interaktion verhalten. In einer blockierenden Delegierung bleibt der Aktivitätsträger,

der einen Auftrag abgibt, solange blockiert, bis eine Antwort zugestellt wird. Formal:

$$\forall\, r_p \in R\,, \forall\, o_j \in O\,, \forall\, a_m \in A\,:$$
$$get(r_p, o_j) \in H(a_m) \Rightarrow H(a_m)\colon snd(r_p) \twoheadrightarrow get(r_p, o_j)$$

Ein Fernaufruf oder RPC [BiNe84] verhält sich in dieser Weise. Nebenläufigkeit ist im Auftraggeber nur möglich, wenn sich mehrere Aktivitätsträger in ihm befinden können. In [Mong89] wurde für den PM-Prototyp ein solches Protokoll implementiert, der Programmierer hat jedoch keine explizite Kontrolle über die Aktivitätsträger, da diese dem Scheduling des Laufzeitsystems unterliegen. Es kann deshalb für den Programmierer in einigen besonderen Fällen wünschenswert sein, daß der Auftraggeber sich in kontrollierter Weise während einer Beauftragung mit einer anderen Aktivität beschäftigt.

**Nicht-blockierende Delegierung**

Eine explizite Antwortannahme löst dieses Problem, da zwischen der Auftragsabgabe und dem Anfordern der Antwort der entsprechende Aktivitätsträger frei ist, um andere Berechnungen durchzuführen. Das Anfordern der Antwort kann jedoch zu einer Blokkade führen. Diese Mechanismen sind als *Futures* oder als *Promises* bekannt [LiSh88]. Formal:

$$\forall\, r_p \in R\,, \forall\, o_j \in O, \forall\, a_m \in A\,:$$
$$get(r_p, o_j) \in H(a_m) \Rightarrow H(a_m)\colon clm(r_{p,}\, o_j) \twoheadrightarrow get(r_p, o_j)$$

**Mehrfachbeauftragung**

Bei der mehrfachen Beauftragung ergibt sich das zusätzliche Problem, daß mehrere Antworten erwartet werden. Einige Systeme synchronisieren sich mit der Zustellung der ersten Antwort. Weitere Antworten müssen explizit angenommen werden (z.B. Multicast im *V-Kernel* [Cher88]). Eine elegante Lösung ist der Einsatz eines Meta-Objekt, das die Rolle einer *Handle* spielt, über die Antworten angefordert werden können. Dieser Mechanismus wird in [Coop90] vorgeschlagen und hat den Vorteil, daß er mit einer expliziten Antwortannahme gut zusammenpaßt. Die Antworten, die von verschiedenen Auftragnehmer kommen, werden im allgemeinen parallel empfangen und abgeliefert, d.h.:

$$\forall\, r_p \in R\,:$$
$$\exists\, o_j, o_k \in Workers(r_p) \text{ und } o_j \neq o_k \Rightarrow$$
$$rep(r_q, o_j) \parallel rep(r_q, o_k) \text{ und } get(r_p, o_j) \parallel get(r_p, o_k)$$

Man beachte, daß die Ereignisse bei einer Mehrfachbeauftragung alle zur gleichen Auftragstransaktion gehören, jedoch nicht im selben Objekt stattfinden. Wenn man auf eine Antwort verzichtet oder wenn überhaupt keine Antwort erwartet wird, dann kann der Ablauf des Protokolls vereinfacht werden.

## 4.2 Definitionen auf der Auftragnehmerseite

Beim Auftragnehmer ergeben sich ebenfalls mehrere Möglichkeiten für den Empfang, das Zustellen und die Abarbeitung eines Auftrags. Wenn kein expliziter Empfang eines Auftrags genutzt wird, d.h. Aufträge nicht angefordert werden, dann müssen Aufträge vom Laufzeitsystem an einen Aktivitätsträger zugeteilt werden. Der Programmierer hat daher keine direkte Kontrolle über diese Zuteilung. Einige Protokolle benutzen einen einzigen Aktivitätsträger, der die Aufträge sequentiell abarbeitet. Dieser Ansatz hat jedoch den Nachteil, die Leistung des Auftragnehmers zu verringern, wenn dieser mit weiteren Objekten interagieren muß, um den Auftrag zu Ende abzuarbeiten. Der Effekt ist, daß der Auftragnehmer solange blockiert bleibt, bis eine Antwort vom dritten Objekt vorliegt und andere Aufträge dadurch unnötig verdrängt werden. Eine Lösung wäre es, Multiplexing zuzulassen, jedoch wird dadurch die Struktur unübersichtlich und es besteht die Gefahr eines *Deadlocks*. Aus diesem Grund schlägt [LHG86] eine dynamische Struktur von Prozessen (Aktivitätsträger) für einen Auftragnehmer vor.

### Dynamische Prozeßstruktur

Es ist wünschenswert, daß ein Aktivitätsträger sich nur mit einem Auftrag befaßt, d.h.:

$$\forall\, r_p \in R\,,\ \forall\, o_j \in \mathrm{Servers}(r_p),\ \exists!\, a_n \in A :$$
$$\mathrm{H\_Wrk}(r_p, o_j) \text{ liegt in } \mathrm{H}(a_n)$$

Ob ein Aktivitätsträger nach der Abarbeitung einen neuen Auftrag abarbeitet, ist bei einem impliziten Empfang für den Programmierer transparent, da diese Entscheidung vom Laufzeitsystem getroffen wird. Die ausgewählte Struktur beim Auftragnehmer bleibt auch für den Auftraggeber transparent.

### Mehrfachbeauftragung

Ein Auftrag kann an mehrere Auftragnehmer gleichzeitig zugestellt werden. Jedoch kann ein Protokoll mit einer mehrfachen Beauftragung für den Auftragnehmer transparent bleiben, wenn die Ablieferung der Aufträge nicht kausal verknüpft sind. Formal:

$$\forall\, r_p \in R\,,\ \forall\, o_j, o_k \in O :$$
$$o_j \neq o_k \text{ und } o_j, o_k \in \mathrm{Servers}(r_p) \Rightarrow \mathrm{dlv}(r_q, o_j) \parallel \mathrm{dlv}(r_q, o_k)$$

### Auftragsende

Die Abarbeitung eines Auftrags kann mit der Antwortabgabe beendet werden. Ein Prozeduraufruf entspricht genau diese Semantik. Formal:

$$\forall\, r_p \in R\,,\ \forall\, o_j \in O :$$
$$o_j \in \mathrm{Servers}(r_p) \Rightarrow \mathrm{ret}(r_p, o_j) \text{ ist letztes Ereignis in } \mathrm{H\_Wrk}(r_p, o_j)$$

Einige Protokolle halten sich nicht an diese Bedingung, um nach der Antwortabgabe weitere Aktivitäten durchzuführen und dadurch den Auftrag vollständig abzuarbeiten. So wird mehr Nebenläufigkeit ermöglicht, da die Beendigung der Abarbeitung parallel zur Antwortannahme stattfindet. Es wird von einer *frühen Antwort* gesprochen.

### 4.3 Primitive

Für ein bestimmtes objektbasiertes System mit bestimmten Formen der Objektinteraktion lassen sich sogenannte Primitive definieren, die elementaren Kommunikationsoperationen der Objektinteraktionen entsprechen. Diese Primitive können als Aktionen betrachtet werden, die mehrere der oben erwähnten Ereignisse auslösen. Der Zusammenhang zwischen den möglichen Ereignissen eines Primitives wird ähnlich formal beschrieben wie z.B. die blockierende Delegierung. Auf eine formale Definition bestimmter Primitive soll hier verzichtet werden, Beispiele dafür finden sich in [Mong92]. Im folgenden soll lediglich aufgezeigt werden, welche Primitive für die bisher genannten Interaktionsformen bei Beauftragung nötig sind.

Der Auftraggeber benötigt ein einziges Primitiv, wenn nur blockierende und einfache Delegierung definiert ist. Dieses Primitiv wird als *Inv* (invoke) bezeichnet. Im Fall einer nicht-blockierenden Delegierung werden zwei Primitive benötigt: *Call* und *Claim*, die auch für eine mehrfache Beauftragung ausreichen. Man kann argumentieren, daß *Inv* überflüssig ist, wenn *Call* und *Claim* hintereinander benutzt werden. Die Zusammenfassung beider Primitive in ein einziges ermöglicht mehrere Optimierungen, die aus der Implementierung von RPC-Protokolle bekannt sind (z.B. bessere Verwaltung von Puffern, weniger Austausch von Kontrollnachrichten, etc.). In einer mitteilenden Beauftragung wird man auf die Beantwortungsphase verzichten. Da auch hier Optimierungen möglich sind, wird auf der Seite des Auftraggebers ein viertes Primitiv eingeführt: *Send*.

Auf der Seite des Auftragnehmers wird nur ein Primitiv benötigt, wenn nur ein impliziter Empfang der Aufträge zugelassen wird. Es wird mit *Reply* bezeichnet. Dieses Primitiv gibt an den Auftraggeber eine Antwort ab. Die Synchronisation dieses Primitivs ist vom Auftragsprotokoll abhängig. Zu beachten ist, daß für einen impliziten Empfang der Aufträge ein Mechanismus existieren muß, um eine Methode oder Prozedur zu registrieren, die zuständig für eine Klasse von Anforderungen ist. Auf dieser Weise kann der Aktivitätsträger einen zugeteilten Auftrag korrekt abarbeiten. Kann der Auftragnehmer Aufträge anfordern (expliziter Empfang), benötigt man ein weiteres Primitiv: *Recv.* Dieses Primitiv kann parametrisiert werden, damit der Auftragnehmer nur Aufträge unter bestimmten Bedingungen entgegen nimmt. So können Aufträge in selektiver Weise abgearbeitet werden.

## 5 Zusammenfassung und Ausblick

Es wurde ein Modell definiert, das typische Objektinteraktionen, die auf *Request-Reply*-Interaktionen basieren, formal erfassen kann. Das Modell berücksichtigt Gruppeninteraktionen zwischen Objekten, die mit der Unterstützung von Multicast-Protokolle effizient implementiert werden können. Es wurde gezeigt, wie Auftragsprotokolle mehr Nebenläufigkeit ermöglichen können, wie z.B. mehrfache Beauftragung, nichtblockierende Delegierung und frühe Antworten. Die Objektinteraktion kann auf der Auftraggeber- und Auftragnehmerseite jeweils unabhängig voneinander definiert wer-

den. So ist eine nicht-blockierende Delegierung für den Auftragnehmer transparent oder ein Auftragnehmer erfährt nicht, ob er einen mehrfachen Auftrag abarbeitet. Eine Ausführlichere Diskussion des Modells kann in [Mong92] gefunden werden.

Das Ereignis-basierte Modell hat jedoch einige Einschränkungen, die noch gelöst werden müssen. So ist z.B. das Anfordern einer Botschaft ein Ereignis, dem nicht bekannt ist, aus welchem Objekt bzw. welcher Transaktion die entsprechende Botschaft zugeteilt wird. Trotzdem werden die Ereignisse req und clm mit der Transaktion bzw. Auftragnehmer ausgelöst.

Zu Untersuchen sind auch weitere Interaktionsformen. So sind in verteilten Systemen auch komplexere Strukturen von Objektinteraktionen zu erkennen. Im Modell wurde nur eine *One-to-many* Interaktionsstruktur berücksichtigt. Die Weiterleitung eines Auftrags an ein anderes Objekt ist ebenfalls ein Mechanismus, der im Modell nicht integriert ist.

# Literatur


[BiNe84] Birrel, A. D., Nelson, B. J., “Implementing remote procedure calls”, *ACM Transactions on Comp. Sys.* 2(1), February 1984, pp. 39-59

[Cher88] Cheriton, D. R., “The V distributed system”, *Comm. of the ACM* 31(3), March 1988, pp. 314-333

[Coop90] Cooper, E.C., “Programming language support for multicast communication in distributed systems”, *Proc. of the 10th IEEE Conf. on Distr. Comp. Sys.*, 1990, pp. 450-457

[Lamp78] Lamport, L., “Time, clocks, and ordering of events in a distributed system”, *Comm. of the ACM* 21(7), July 1978, pp. 558-565

[LHG86] Liskov, B., Herlihy, M., Gilbert, L., “Limitations of synchronous communication with static process structure in languages for distributed computing”, *Proc. of the 13th ACM Symp. on Principles of Progr. Lang.*, (St. Petersburg, FL), January 1986

[LiSh88] Liskov, B., Shrira, L. “Promises: linguistic support for efficient asynchronous procedure calls in distributed systems”, *Proc. of the SIGPLAN 88 Conf. on Progr. Lang. Design and Impl.*, SIGPLAN Notices 23(7), June 1988, pp. 260-268

[Mong89] Monge, R., “PM-RPC: an interobject communication mechanism for global objects”, *Arbeitsberichte des IMMD* 22(4), Univ. Erlangen-Nürnberg, Februar 1989

[Mong92] Monge, R., *Kommunikation in verteilten, objektbasierten Systemen*, Dissertation, Arbeitsberichte des IMMD 25(7), Univ. Erlangen-Nürnberg, September 1992