Functional Specification of ‘Delivery’

Put author(s) here
(This document was generated by Ampersand vs. 2.1.0.66)

Sat May 7 09:45:17 West-Europa (standaardtijd) 2011

Contents

1 Introduction **3**

2 Shared Language **4**

2.1 Deliveries 4

2.2 Delivery 5

3 Diagnosis **7**

4 Conceptual Analysis **10**

4.1 Deliveries 10

4.2 Sessions 12

5 Process Analysis **13**

5.1 Delivery 14

6 Function Point Analysis **17**

7 Data structure **18**

7.1 Invoice 18

7.2 Order 19

7.3 Delivery 19

7.4 Session 19

7.5 item1 20

7.6 item2 20

8 login **22**

9 create orders **24**

10 accept orders **26**

11 ship orders **28**

12 pay invoices **31**

13 receive goods **33**

14 send invoices **36**

15 ECA rules **39**

Chapter 1

Introduction

This document defines the functionality of an information system called 'Delivery'. It defines business services in a system where people and applications work together in order to fullfill their commitments. A number of these rules have been used as functional requirement to assemble this functional specification[1]. Those rules are listed in chapter 2, ordered by theme.

The diagnosis in chapter 3 is meant to help the authors identify shortcomings in their Ampersand script.

The conceptual analysis in chapter 4 is meant for requirements engineers and architects to validate and formalize the requirements from chapter 2. It is also meant for testers to come up with correct test cases. The formalization in this chapter makes consistency of the functional specification provable. It also yields an unambiguous interpretation of all requirements.

Chapters that follow have the builders of 'Delivery' as their intended audience. The data analysis in chapter 7 describes the data sets upon which 'Delivery' is built. Each subsequent chapter defines one business service. This allows builders focus on a single service at a time. Together, these services fulfill all commitments from chapter 2. By disclosing all functionality exclusively through these services, 'Delivery' ensures compliance to all rules from chapter 2.

[1] To use agreements as functional requirements characterizes the Ampersand approach, which has been used to produce this document.

Chapter 2

Shared Language

This chapter defines the natural language, in which functional requirements of 'Delivery' can be discussed and expressed. The purpose of this chapter is to create shared understanding among stakeholders. The language of 'Delivery' consists of concepts and basic sentences. All functional requirements are expressed in these terms. When stakeholders can agree upon this language, at least within the scope of 'Delivery', they share precisely enough language to have meaningful discussions about functional requirements. All definitions have been numbered for the sake of traceability.

2.1 Deliveries

We want orders to be delivered correctly, with only items that are mentioned on the order.

Requirement 1 (correct delivery): Each item in a delivery is mentioned on the order.

We want orders to be delivered completely, with no items missing.

Requirement 2 (complete delivery): Every item on an order must also be in the corresponding delivery.

Accepting an order is always done by the provider to whom the order was addressed. To prevent an order to be accepted or rejected by anyone else, we need this requirement.

Requirement 3 (proper address): A provider can only accept or reject orders that are addressed to that provider.

In this context, providers only deliver when there is an order. So, if a delivery is made by a provider, we assume the existence of an order that is accepted by that provider.

Requirement 4 (order based delivery): For every delivery a provider has made, there exists an accepted order.

To prevent arbitrary payments, we enforce that every invoice is paid by the client to whom it was sent.

Requirement 5 (correct payments): Payments are made only for invoices sent.

To make sure that deliveries are billed to the right customer, there must be a delivery for each invoice sent.

Requirement 6 (correct invoices): Invoices are sent to a customer only if there is a delivery made to that customer.

2.2 Delivery

This paragraph describes the theme 'Delivery'.

Requirement 7 (login): If there exists a Session called s, (If there exists a Provider called p, s is being run by p and True) and (If there exists a Client called c, not(s is being run by c) and True) or (If there exists a Client called c, s is being run by c and True) or (If there exists a Provider called p, not(s is being run by p) and True).

Orders should be deliverable and payable. For such purposes, it is necessary to know the client (customer) that created the order

Requirement 8 (create orders): Each order is created by precisely one client.

Not every order received by a provider leads to a delivery. The provider may decide to accept or to reject an order. Eventually, all orders must be acknowledged, either positively (accept) or negatively (reject).

Orders are issued by clients for the purpose of making a sales transaction. At some point in time, a provider must accept or reject the order. In this simplistic model, every order will be accepted. A more realistic model should at least ensure that orders will not be lost.

Requirement 9 (accept orders): Orders addressed to a provider must be accepted or rejected by that provider.

Ultimately, each order accepted must be shipped by the provider who has accepted that order. The provider will be signalled of orders waiting to be shipped.

Requirement 10 (ship orders): Each order that has been accepted by a provider must (eventually) be shipped by that provider.

A client who receives an invoice must eventually pay.

Requirement 11 (pay invoices): All invoices sent to a customer must be paid by that customer.

The ordered goods must be delivered at some point in time to the client. This is done in one delivery.

Requirement 12 (receive goods): Every delivery must be acknowledged by the client who placed the corresponding order.

In order to induce payment, a provider sends an invoice for deliveries made.

Requirement 13 (send invoices): After a delivery has been made to a customer, an invoice must be sent.

Chapter 3

Diagnosis

This chapter provides an analysis of the Ampersand script of 'Delivery'. This analysis is intended for the authors of this script. It can be used to complete the script or to improve possible flaws.

Delivery assigns rules to roles. The following table shows the rules that are being maintained by a given role.

rule	Client	Provider
login	×	×
create orders	×	
accept orders		×
ship orders		×
pay invoices	×	
receive goods	×	
send invoices		×

Concepts Order, Client, Product, Delivery, Provider, Invoice, and Session remain without a definition.

Relations $sProvider$, $sClient$, $from_{[Order\times Client]}$, $addressedTo_{[Order\times Provider]}$, $item_{[Order\times Product]}$, $rejected$, $accepted$, $addressedTo$, $of_{[Delivery\times Order]}$, $accepted_{[Provider\times Order]}$, $delivery$, $from_{[Invoice\times Provider]}$, $sentTo$, $paid_{[Client\times Invoice]}$, $item_{[Delivery\times Product]}$, $provided$, and $deliveredTo$ are explained by an automated explanation generator. If these explanations are not appropriate, add PURPOSE RELATION statements to your relations.

Relation $from_{[Invoice\times Provider]}$ is not being used in any rule.

Figure 3.1 shows a conceptual diagram with all relations declared in 'Deliveries'.

Figure 3.2 shows a conceptual diagram with all relations declared in 'Sessions'.

The purpose of rule $I_{[Session]} \vdash sProvider;V_{[Provider\times Session]}\cap\overline{sClient;V_{[Client\times Session]}}\cup sClient;V_{[Client\times Sessio}$ (line 142 of file Delivery.adl) is not documented.

All rules in process Delivery are linked to roles.

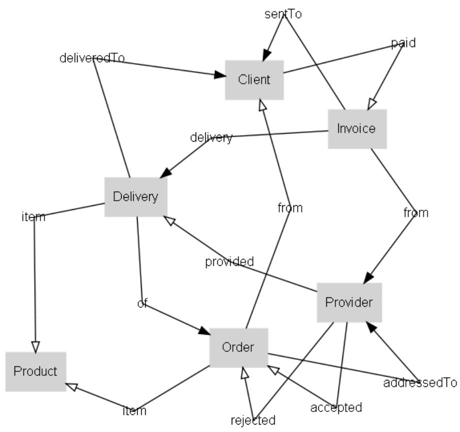

Figure 3.1: Concept analysis of relations in Deliveries

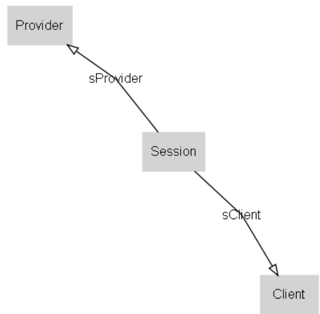

Figure 3.2: Concept analysis of relations in Sessions

All role-rule assigments involve rules that are defined in process 'Delivery'.

The following table represents the population of various relations.

Concept	Population
Order	3
Client	3
Product	3
Delivery	3
Provider	2
Invoice	3

Relation	Population
$\mathit{from} : \mathit{Order} \times \mathit{Client}$	3
$\mathit{item} : \mathit{Order} \times \mathit{Product}$	3
$\mathit{item} : \mathit{Delivery} \times \mathit{Product}$	3
$\mathit{of} : \mathit{Delivery} \times \mathit{Order}$	3
$\mathit{provided} : \mathit{Provider} \times \mathit{Delivery}$	3
$\mathit{accepted} : \mathit{Provider} \times \mathit{Order}$	3
$\mathit{rejected} : \mathit{Provider} \times \mathit{Order}$	3
$\mathit{addressedTo} : \mathit{Order} \times \mathit{Provider}$	3
$\mathit{deliveredTo} : \mathit{Delivery} \times \mathit{Client}$	3
$\mathit{sentTo} : \mathit{Invoice} \times \mathit{Client}$	3
$\mathit{delivery} : \mathit{Invoice} \times \mathit{Delivery}$	3
$\mathit{from} : \mathit{Invoice} \times \mathit{Provider}$	3
$\mathit{paid} : \mathit{Client} \times \mathit{Invoice}$	3

The population in this script does not specify any work in progress.

The population in this script violates no rule.

Chapter 4

Conceptual Analysis

This chapter provides an analysis of the principles described in chapter 2. Each section in that chapter is analysed in terms of relations and each principle is then translated in a rule.

4.1 Deliveries

Figure 4.1 shows a conceptual diagram of this theme.

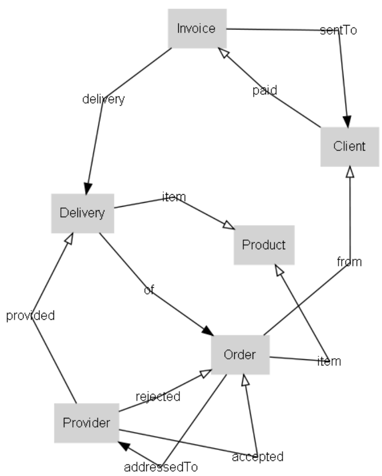

Figure 4.1: Conceptual model of Deliveries

correct delivery We want orders to be delivered correctly, with only items that are mentioned on the order.

To arrive at the formalization in equation 4.4, the following three relations are introduced.

$$\begin{aligned} item &: Delivery \times Product && (4.1)\\ item &: Order \times Product && (4.2)\\ of &: Delivery \rightarrow Order && (4.3) \end{aligned}$$

This means:

$$\forall d :: Delivery; p :: Product : d\ item\ p \Rightarrow of(d)\ item\ p \qquad (4.4)$$

This corresponds to requirement 2.1 on page 4.

complete delivery We want orders to be delivered completely, with no items missing.

We use definitions 4.1 (*item*), 4.2 (*item*), and 4.3 (*of*). This means:

$$\forall d :: Delivery; p :: Product : of(d)\ item\ p \Rightarrow d\ item\ p \qquad (4.5)$$

proper address Accepting an order is always done by the provider to whom the order was addressed. To prevent an order to be accepted or rejected by anyone else, we need this requirement.

To arrive at the formalization in equation 4.9, the following three relations are introduced.

$$\begin{aligned} rejected &: Provider \times Order && (4.6)\\ accepted &: Provider \times Order && (4.7)\\ addressedTo &: Order \rightarrow Provider && (4.8) \end{aligned}$$

This means:

$$\forall p :: Provider; o :: Order : \neg(p\ accepted\ o) \wedge \neg(p\ rejected\ o) \vee p\ =\ addressedTo(o) \qquad (4.9)$$

This corresponds to requirement 2.1 on page 4.

order based delivery In this context, providers only deliver when there is an order. So, if a delivery is made by a provider, we assume the existence of an order that is accepted by that provider.

In order to formalize this, a relation provided is introduced (4.10):

$$provided : Provider \times Delivery \qquad (4.10)$$

We also use definitions 4.7 (*accepted*) and 4.3 (*of*) to formalize requirement 2.1 (page 5): This means:

$$\forall p :: Provider; d :: Delivery : p\ =\ provided(d) \Rightarrow (\exists o :: Order : p\ accepted\ o \wedge o\ =\ of(d)) \qquad (4.11)$$

correct payments To prevent arbitrary payments, we enforce that every invoice is paid by the client to whom it was sent.

To arrive at the formalization in equation 4.14, the following two relations are introduced.

$$\begin{aligned} \mathit{paid} &: \mathit{Client} \times \mathit{Invoice} && (4.12) \\ \mathit{sentTo} &: \mathit{Invoice} \rightarrow \mathit{Client} && (4.13) \end{aligned}$$

This means:

$$\forall c :: Client; i :: Invoice : c \ \mathit{paid} \ i \Rightarrow c \ = \ \mathit{sentTo}(i) \qquad (4.14)$$

This corresponds to requirement 2.1 on page 5.

correct invoices To make sure that deliveries are billed to the right customer, there must be a delivery for each invoice sent.

To arrive at the formalization in equation 4.17, the following two relations are introduced.

$$\begin{aligned} \mathit{from} &: \mathit{Order} \times \mathit{Client} && (4.15) \\ \mathit{delivery} &: \mathit{Invoice} \rightarrow \mathit{Delivery} && (4.16) \end{aligned}$$

We also use definitions 4.13 (sentTo) and 4.3 (of). This means:

$$\forall i :: Invoice; c :: Client : \mathit{sentTo}(i) \ = \ c \Rightarrow \mathit{of}(\mathit{delivery}(i)) \ \mathit{from} \ c \qquad (4.17)$$

This corresponds to requirement 2.1 on page 5.

4.2 Sessions

Figure 4.2 shows a conceptual diagram of this theme.

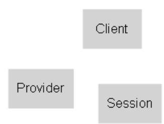

Figure 4.2: Conceptual model of Sessions

Chapter 5

Process Analysis

Delivery assigns rules to roles. The following table shows the rules that are being maintained by a given role.

Role	Rule
Client	login create orders pay invoices receive goods
Provider	login accept orders ship orders send invoices

Delivery assigns roles to relations. The following table shows the relations, the content of which can be altered by anyone who fulfills a given role.

Role	Relation
Client	$sClient_{[Session \times Client]}$ $from_{[Order \times Client]}$ $item_{[Order \times Product]}$ $paid$ $deliveredTo$
Provider	$sProvider_{[Session \times Provider]}$ $accepted$ $rejected$ $item_{[Delivery \times Product]}$ $item_{[Delivery \times Product]}$
	$of : Delivery \times Order$ $provided : Provider \times Delivery$ $addressedTo : Order \times Provider$ $sentTo : Invoice \times Client$ $delivery : Invoice \times Delivery$ $from : Invoice \times Provider$

5.1 Delivery

Figure 5.1 shows the process model.

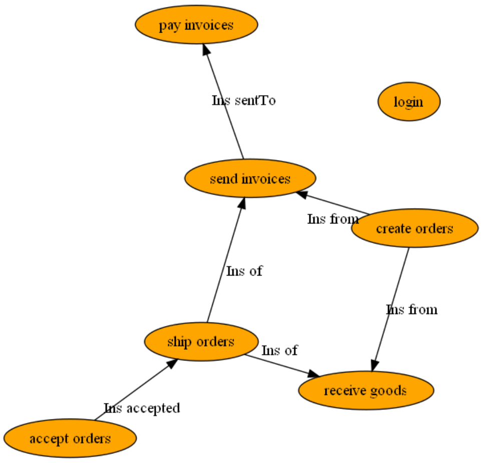

Figure 5.1: Process model of Delivery

The conceptual diagram of figure 5.2 provides an overview of the language in which this process is expressed.

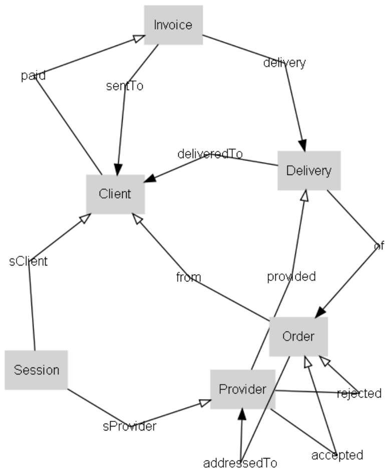

Figure 5.2: Basic sentences of Delivery

login We use definitions **??** (*sProvider*) and **??** (*sClient*). Activities that are defined by this rule are finished when:

$$\forall s :: Session : (\exists p :: Provider : s\ sProvider\ p \wedge V) \wedge (\exists c :: Client : \neg(s\ sClient\ c) \wedge V) \vee (\exists c :: Client : s\ sC \tag{5.1}$$

These activities are signalled by:

$$\forall s, s' :: Session : s = s' \wedge (\neg(\exists p :: Provider : s\ sProvider\ p \wedge V) \vee \neg(\exists c :: Client : \neg(s\ sClient\ c) \wedge V)) \wedge \neg(\tag{5.2}$$

create orders Orders should be deliverable and payable. For such purposes, it is necessary to know the client (customer) that created the order

We use definition 4.15 (from). Activities that are defined by this rule are finished when:

$$\forall o :: Order : (\exists c :: Client : o\ \mathit{from}\ c \wedge o\ \mathit{from}\ c) \tag{5.3}$$

These activities are signalled by:

$$\forall o, o' :: Order : o = o' \wedge \neg(\exists c :: Client : o\ \mathit{from}\ c \wedge o'\ \mathit{from}\ c) \tag{5.4}$$

accept orders Not every order received by a provider leads to a delivery. The provider may decide to accept or to reject an order. Eventually, all orders must be acknowledged, either positively (accept) or negatively (reject).

Orders are issued by clients for the purpose of making a sales transaction. At some point in time, a provider must accept or reject the order. In this simplistic model, every order will be accepted. A more realistic model should at least ensure that orders will not be lost.

We use definitions 4.7 (*accepted*), 4.6 (*rejected*), and 4.8 (*addressedTo*). Activities that are defined by this rule are finished when:

$$\forall p :: Provider; o :: Order : p = addressedTo(o) \Rightarrow p\ accepted\ o \vee p\ rejected\ o \tag{5.5}$$

These activities are signalled by:

$$\forall p :: Provider; o :: Order : p = addressedTo(o) \wedge \neg(p\ accepted\ o) \wedge \neg(p\ rejected\ o) \tag{5.6}$$

ship orders Ultimately, each order accepted must be shipped by the provider who has accepted that order. The provider will be signalled of orders waiting to be shipped.

We use definitions 4.3 (*of*), 4.10 (*provided*), and 4.7 (*accepted*). Activities that are defined by this rule are finished when:

$$\forall p :: Provider; o :: Order : p\ accepted\ o \Rightarrow (\exists d :: Delivery : p = provided(d) \wedge of(d) = o) \tag{5.7}$$

These activities are signalled by:

$$\forall p :: Provider; o :: Order : p\ accepted\ o \wedge \neg(\exists d :: Delivery : p = provided(d) \wedge of(d) = o) \tag{5.8}$$

pay invoices A client who receives an invoice must eventually pay.

We use definitions 4.13 (*sentTo*) and 4.12 (*paid*). Activities that are defined by this rule are finished when:

$$\forall c :: Client; i :: Invoice : c = sentTo(i) \Rightarrow c\ paid\ i \tag{5.9}$$

These activities are signalled by:

$$\forall c :: Client; i :: Invoice : c \;=\; sentTo(i) \wedge \neg(c \; paid \; i) \tag{5.10}$$

receive goods The ordered goods must be delivered at some point in time to the client. This is done in one delivery.

We use definitions 4.15 (*from*), 4.3 (*of*), and **??** (*deliveredTo*). Activities that are defined by this rule are finished when:

$$\forall d :: Delivery; c :: Client : of(d) \; from \; c \Rightarrow deliveredTo(d) \;=\; c \tag{5.11}$$

These activities are signalled by:

$$\forall d :: Delivery; c :: Client : of(d) \; from \; c \wedge \neg(deliveredTo(d) \;=\; c) \tag{5.12}$$

send invoices In order to induce payment, a provider sends an invoice for deliveries made.

We use definitions 4.15 (*from*), 4.3 (*of*), 4.13 (*sentTo*), and 4.16 (*delivery*). Activities that are defined by this rule are finished when:

$$\forall i :: Invoice; c :: Client : of(delivery(i)) \; from \; c \Rightarrow sentTo(i) \;=\; c \tag{5.13}$$

These activities are signalled by:

$$\forall i :: Invoice; c :: Client : of(delivery(i)) \; from \; c \wedge \neg(sentTo(i) \;=\; c) \tag{5.14}$$

Chapter 6

Function Point Analysis

The specification of 'Delivery' has been analysed by counting function points[?]. This has resulted in an estimated total of 49 function points.

data set	analysis	FP
Invoice	ILGV Eenvoudig	7
Order	ILGV Eenvoudig	7
Delivery	ILGV Eenvoudig	7
Session	ILGV Eenvoudig	7
Provider	ILGV Eenvoudig	7
Product	ILGV Eenvoudig	7
Client	ILGV Eenvoudig	7

interface	analysis	FP
login	NO	0
create orders	NO	0
accept orders	NO	0
ship orders	NO	0
pay invoices	NO	0
receive goods	NO	0
send invoices	NO	0

Chapter 7

Data structure

The requirements, which are listed in chapter 2, have been translated into the data model in figure 7.2. There are four data sets, two associations, no generalisations, and no aggregations. Delivery has a total of 7 concepts.

Figure 7.1: Classification of Delivery

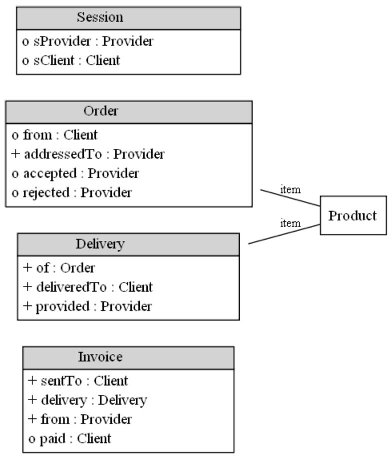

Figure 7.2: Data model of Delivery

7.1 Invoice

The attributes in Invoice have the following multiplicity constraints.

attribute	type	mandatory	unique
key	Invoice	$\surd$	$\surd$
sentTo	Client	$\surd$	
delivery	Delivery	$\surd$	
from	Provider	$\surd$	
paid	Client		

7.2 Order

The attributes in Order have the following multiplicity constraints.

attribute	type	mandatory	unique
key	Order	$\surd$	$\surd$
from	Client		
addressedTo	Provider	$\surd$	
accepted	Provider		
rejected	Provider		

The following rule defines the integrity of data within this data set. It must remain true at all times.

$$\forall o, o' :: Order : \neg(o = o') \vee (\exists c :: Client : o \ from \ c \wedge o' \ from \ c)$$

7.3 Delivery

The attributes in Delivery have the following multiplicity constraints.

attribute	type	mandatory	unique
key	Delivery	$\surd$	$\surd$
of	Order	$\surd$	
deliveredTo	Client	$\surd$	
provided	Provider	$\surd$	

7.4 Session

The attributes in Session have the following multiplicity constraints.

attribute	type	mandatory	unique
key	Session	$\surd$	$\surd$
sProvider	Provider		
sClient	Client		

The following rule defines the integrity of data within this data set. It must remain true at all times.

$$\forall s, s' :: Session : \neg(s = s') \vee (\exists p :: Provider : s \ sProvider \ p \wedge V) \wedge (\exists c :: Client : \neg(s \ sClient \ c) \wedge V) \vee (\exists c :: Clie$$

7.5 item1

The attributes in item1 have the following multiplicity constraints.

attribute	type	mandatory	unique
key	Order	$\surd$	
Product	Product	$\surd$	

7.6 item2

The attributes in item2 have the following multiplicity constraints.

attribute	type	mandatory	unique
key	Delivery	$\surd$	
Product	Product	$\surd$	

Figure 7.3 shows the switchboard diagram.This is used in designing the database functionality.

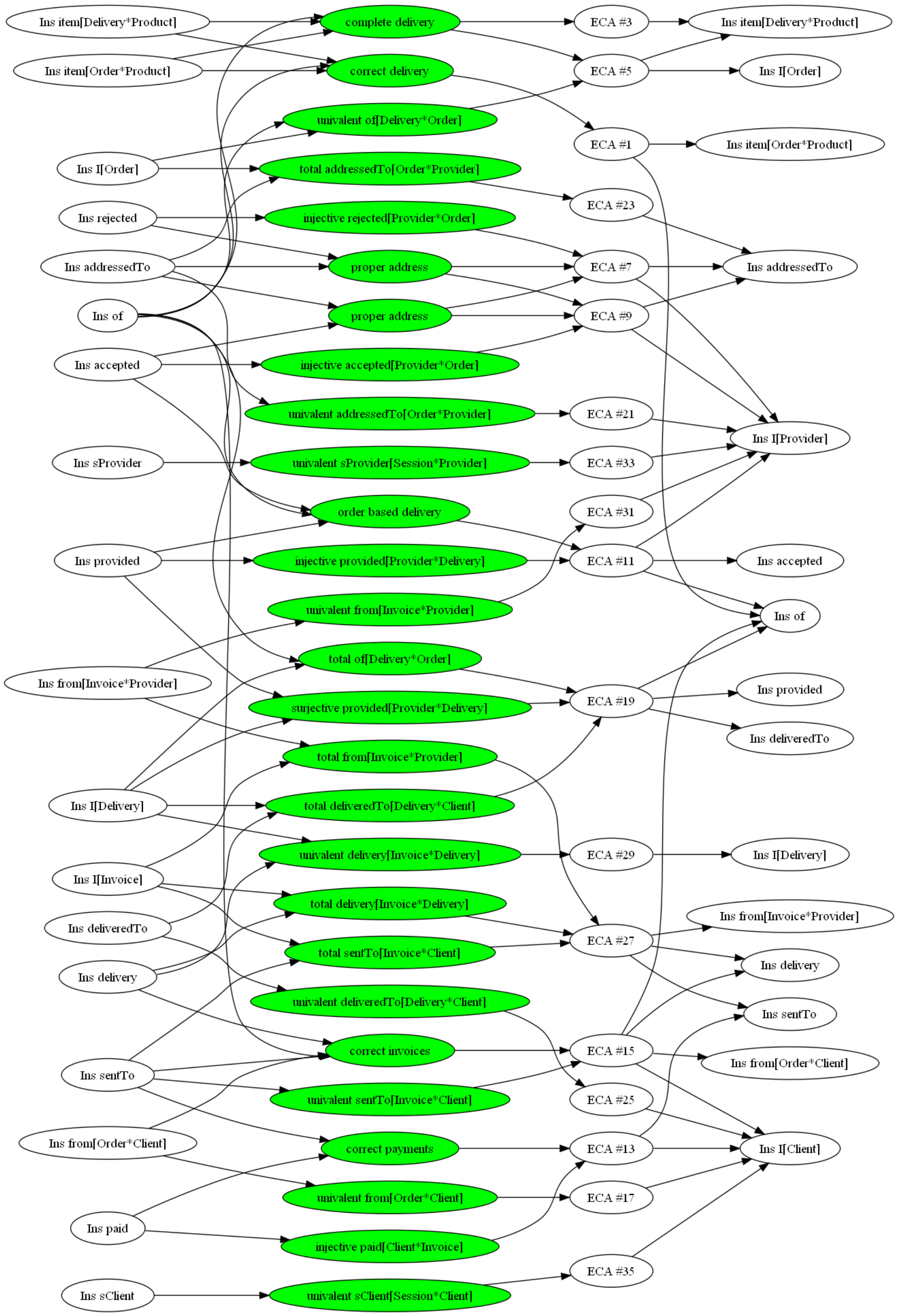

Figure 7.3: Switchboard of Delivery

Chapter 8

login

For what purpose activity 'login' exists remains undocumented. Activity 'login' must be performed by a user with role Client or Provider. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	sProvider[Session*Provider]	ECA rule 33
Del	sProvider[Session*Provider]	error: rule 'univalent sProvider[Session*Provider]'
Ins	sClient[Session*Client]	ECA rule 35
Del	sClient[Session*Client]	error: rule 'univalent sClient[Session*Client]'

Figure 8.1 shows the knowledge graph of this interface.

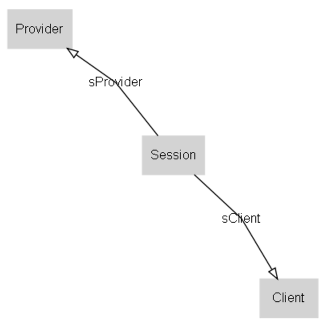

Figure 8.1: Language diagram of login

Every section in this chapter describes one activity. While performing an activity, users will insert or delete population in various relations. This may potentially violate invariants. (An invariant is a business rule rules that must remain true at all times.) The software to maintain the truth of invariant rules is generated automatically. The structure of that software is illustrated by a so called switchboard diagram, the first of which you will find in figure X. Each switchboard diagram consists of three columns: Invariant rules are drawn in the middle, and relations occur on the (right and left hand) sides. An arrow on the left hand side points from a relation that may be edited to a rule that may be violated as a consequence thereof. Each arrow on the right hand side of a rule represents an edit action that is required to restore its truth. It points to the relation that is edited for that purpose. If that arrow is labeled '+', it involves an insert event; if labeled '-' it refers to a delete event. This yields an accurate perspective on the way in which invariants are maintained.

Figure 8.2 shows the switchboard diagram of this interface.

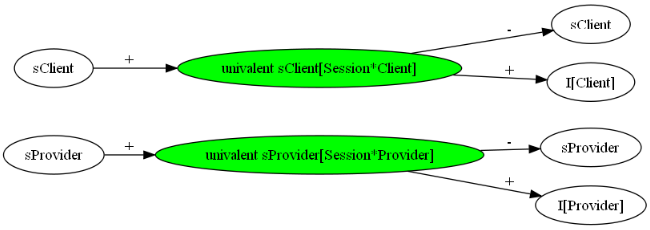

Figure 8.2: Switchboard of login

Chapter 9

create orders

Orders should be deliverable and payable. For such purposes, it is necessary to know the client (customer) that created the order

Activity ‘create orders’ must be performed by a user with role Client. During that activity, rule ‘correct invoices’ will be maintained without intervention of a user. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	from[Order*Client]	ECA rule 17
Del	from[Order*Client]	error: rule ‘univalent from[Order*Client]’
Ins	I	ECA rule 23
Del	I	error: rule ‘total addressedTo[Order*Provider]’

Figure 9.1 shows the knowledge graph of this interface.

Figure 9.2 shows the switchboard diagram of this interface.

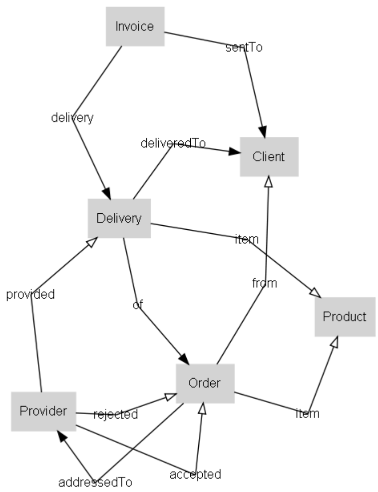

Figure 9.1: Language diagram of create orders

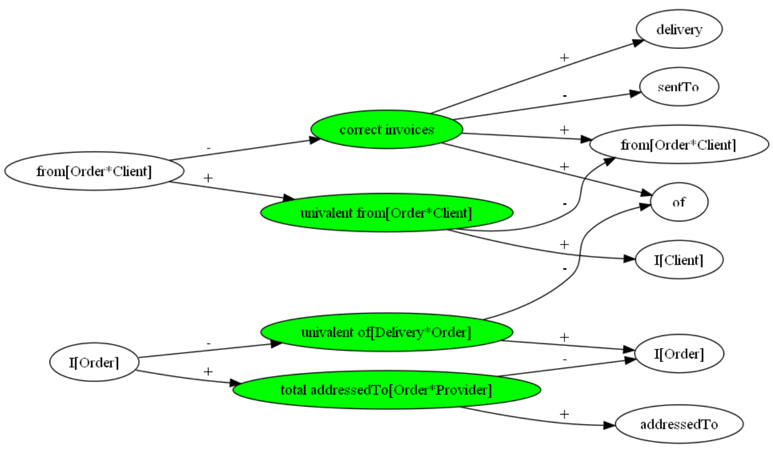

Figure 9.2: Switchboard of create orders

Chapter 10

accept orders

Not every order received by a provider leads to a delivery. The provider may decide to accept or to reject an order. Eventually, all orders must be acknowledged, either positively (accept) or negatively (reject).

Orders are issued by clients for the purpose of making a sales transaction. At some point in time, a provider must accept or reject the order. In this simplistic model, every order will be accepted. A more realistic model should at least ensure that orders will not be lost.

Activity 'accept orders' must be performed by a user with role Provider. During that activity, rules 'proper address' and 'order based delivery' will be maintained without intervention of a user. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	rejected[Provider*Order]	ECA rule 7
Del	rejected[Provider*Order]	error: rule 'proper address' and 'injective rejected[Provider*Order
Ins	accepted[Provider*Order]	ECA rule 9
Del	accepted[Provider*Order]	error: rule 'proper address' and 'injective accepted[Provider*Orde
Ins	addressedTo[Order*Provider]	ECA rule 21
Del	addressedTo[Order*Provider]	error: rule 'univalent addressedTo[Order*Provider]'

Figure 10.1 shows the knowledge graph of this interface.

Figure 10.2 shows the switchboard diagram of this interface.

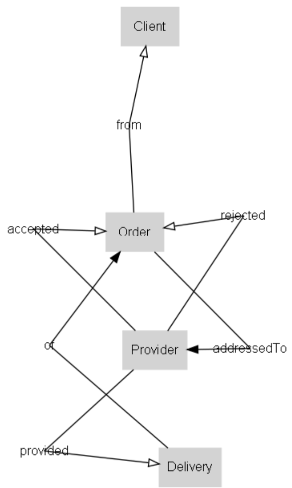

Figure 10.1: Language diagram of accept orders

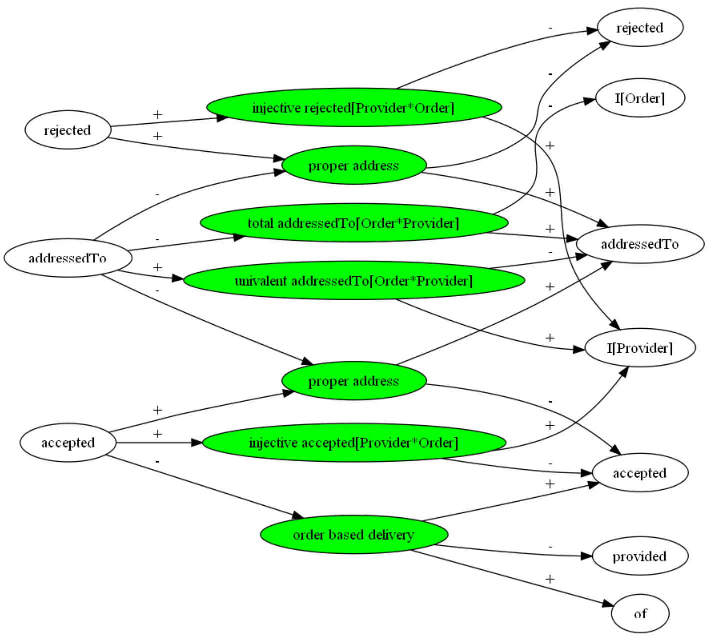

Figure 10.2: Switchboard of accept orders

Chapter 11

ship orders

Ultimately, each order accepted must be shipped by the provider who has accepted that order. The provider will be signalled of orders waiting to be shipped.

Activity ‘ship orders’ must be performed by a user with role Provider. During that activity, rules ‘correct delivery’, ‘complete delivery’, ‘proper address’, ‘order based delivery’, and ‘correct invoices’ will be maintained without intervention of a user. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	of[Delivery*Order]	ECA rule 5
Del	of[Delivery*Order]	error: rule ‘complete delivery’ and ‘univalent of[Delivery*Order]’
Ins	accepted[Provider*Order]	ECA rule 9
Del	accepted[Provider*Order]	error: rule ‘proper address’ and ‘injective accepted[Provider*Order]
Ins	provided[Provider*Delivery]	ECA rule 11
Del	provided[Provider*Delivery]	error: rule ‘injective provided[Provider*Delivery]’

Figure 11.1 shows the knowledge graph of this interface.

Figure 11.2 shows the switchboard diagram of this interface.

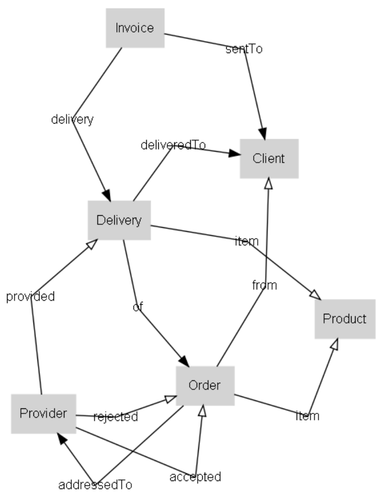

Figure 11.1: Language diagram of ship orders

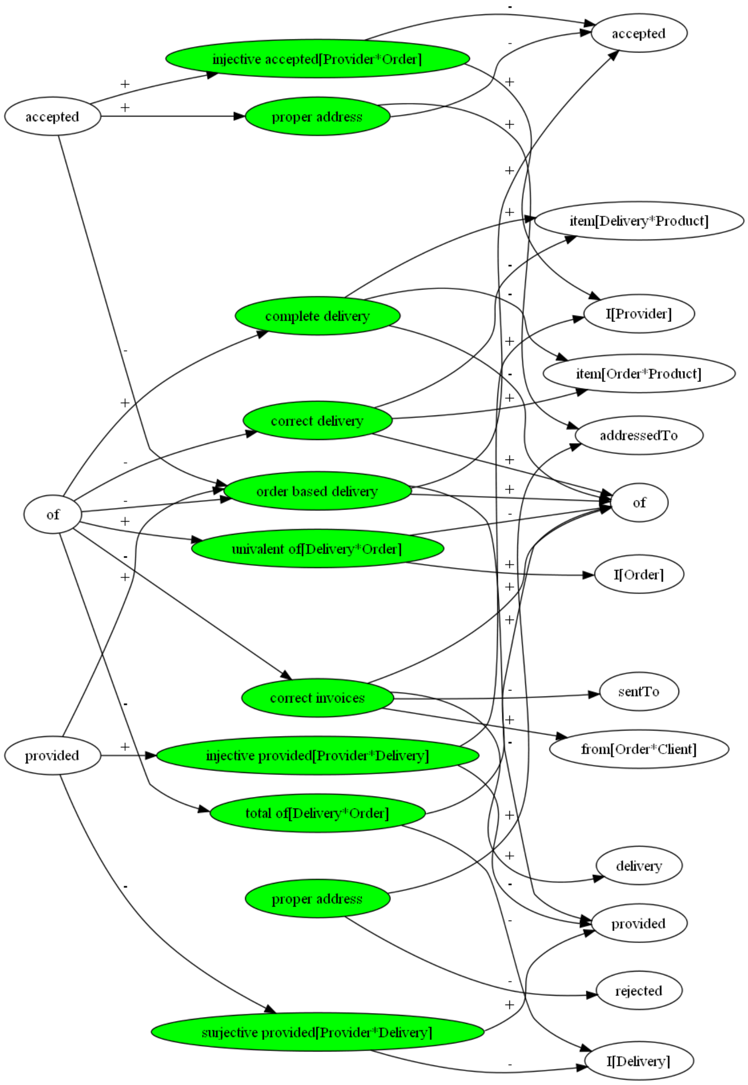

Figure 11.2: Switchboard of ship orders

Chapter 12

pay invoices

A client who receives an invoice must eventually pay.

Activity ‘pay invoices’ must be performed by a user with role Client. During that activity, rules ‘correct payments’ and ‘correct invoices’ will be maintained without intervention of a user. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	paid[Client*Invoice]	ECA rule 13
Del	paid[Client*Invoice]	error: rule ‘correct payments’ and ‘injective paid[Client*Invoice]’
Ins	sentTo[Invoice*Client]	ECA rule 15
Del	sentTo[Invoice*Client]	error: rule ‘univalent sentTo[Invoice*Client]’

Figure 12.1 shows the knowledge graph of this interface.

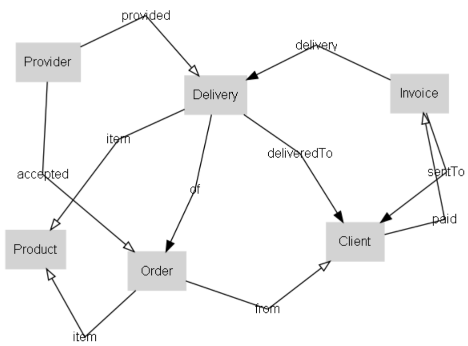

Figure 12.1: Language diagram of pay invoices

Figure 12.2 shows the switchboard diagram of this interface.

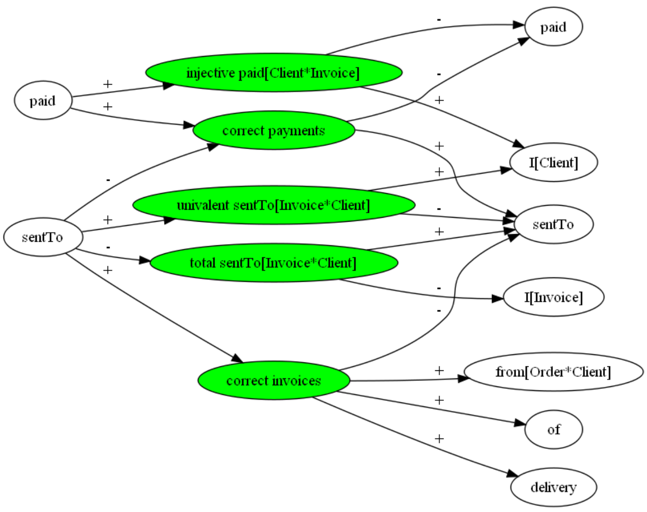

Figure 12.2: Switchboard of pay invoices

Chapter 13

receive goods

The ordered goods must be delivered at some point in time to the client. This is done in one delivery.

Activity ‘receive goods’ must be performed by a user with role Client. During that activity, rules ‘correct delivery’, ‘complete delivery’, ‘order based delivery’, and ‘correct invoices’ will be maintained without intervention of a user. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	of[Delivery*Order]	ECA rule 5
Del	of[Delivery*Order]	error: rule ‘complete delivery’ and ‘univalent of[Delivery*Order]’
Ins	from[Order*Client]	ECA rule 17
Del	from[Order*Client]	error: rule ‘univalent from[Order*Client]’
Ins	deliveredTo[Delivery*Client]	ECA rule 25
Del	deliveredTo[Delivery*Client]	error: rule ‘univalent deliveredTo[Delivery*Client]’

Figure 13.1 shows the knowledge graph of this interface.

Figure 13.2 shows the switchboard diagram of this interface.

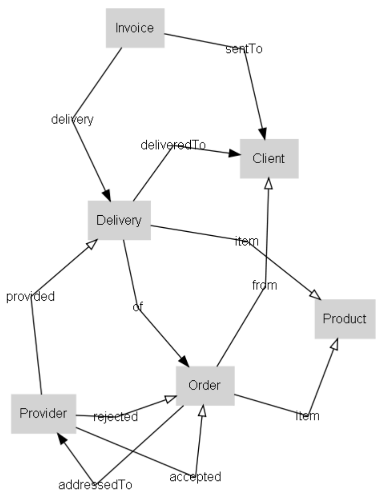

Figure 13.1: Language diagram of receive goods

Figure 13.2: Switchboard of receive goods

Chapter 14

send invoices

In order to induce payment, a provider sends an invoice for deliveries made.

Activity ‘send invoices’ must be performed by a user with role Provider. During that activity, rules ‘correct delivery’, ‘complete delivery’, ‘order based delivery’, ‘correct payments’, and ‘correct invoices’ will be maintained without intervention of a user. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	of[Delivery*Order]	ECA rule 5
Del	of[Delivery*Order]	error: rule ‘complete delivery’ and ‘univalent of[Delivery*Order]’
Ins	sentTo[Invoice*Client]	ECA rule 15
Del	sentTo[Invoice*Client]	error: rule ‘univalent sentTo[Invoice*Client]’
Ins	from[Order*Client]	ECA rule 17
Del	from[Order*Client]	error: rule ‘univalent from[Order*Client]’
Ins	delivery[Invoice*Delivery]	ECA rule 29
Del	delivery[Invoice*Delivery]	error: rule ‘univalent delivery[Invoice*Delivery]’

Figure 14.1 shows the knowledge graph of this interface.

Figure 14.2 shows the switchboard diagram of this interface.

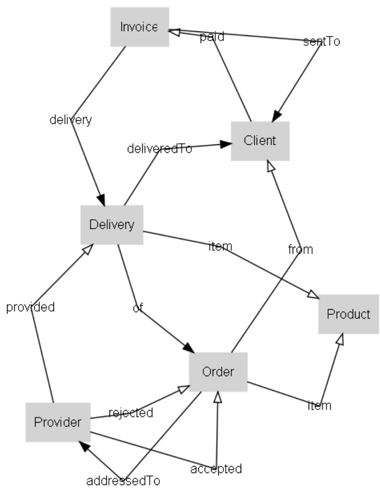

Figure 14.1: Language diagram of send invoices

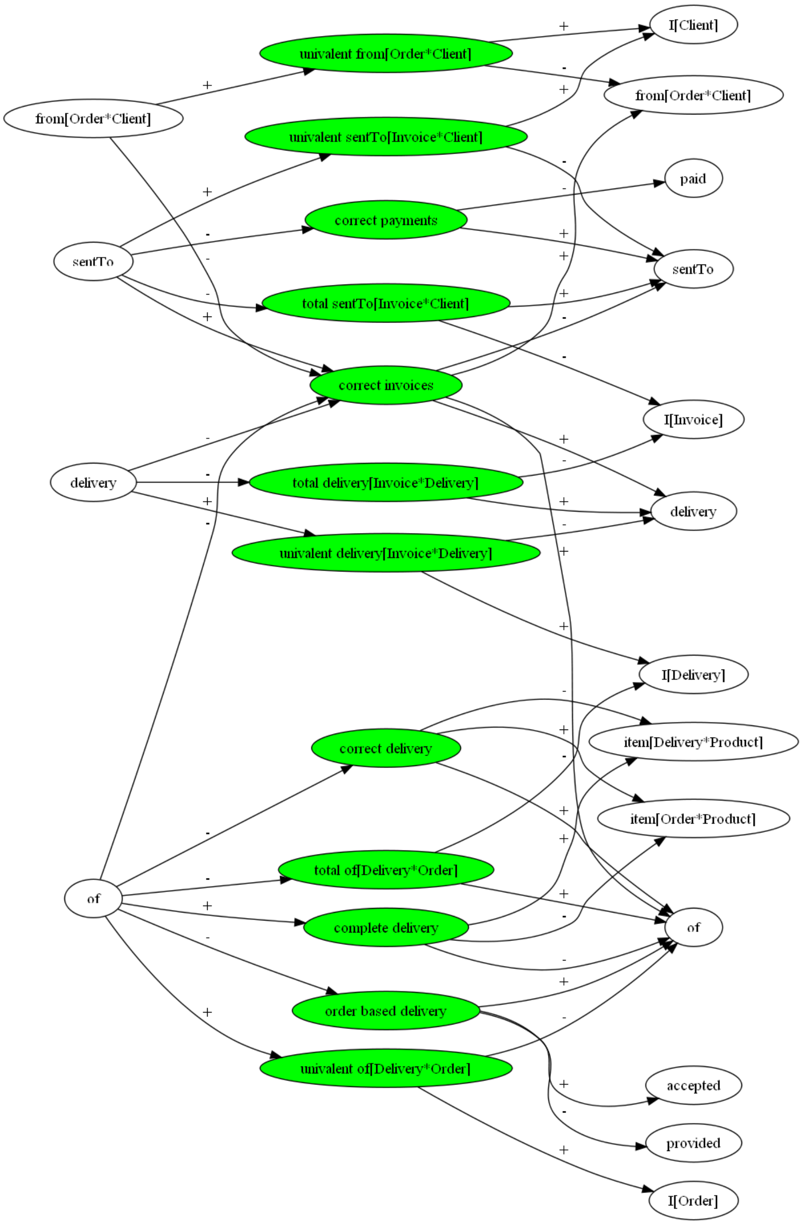

Figure 14.2: Switchboard of send invoices

Chapter 15

ECA rules

This chapter lists the ECA rules. ECA rules:
temporarily not documented

```
ON INSERT Delta IN item[Delivery*Product] EXECUTE    -- (ECA rule 1)
ONE of CREATE x:Order;
         ALL of INSERT INTO item[Order*Product] SELECTFROM
                  V[Order*Delivery];((item[Delivery*Product] \/ Delta)/\-(of;item[Ord
                INSERT INTO of SELECTFROM
                  ((item[Delivery*Product] \/ Delta)/\-(of;item[Order*Product]));V[Pr
         (MAINTAINING -item[Delivery*Product] \/ of;item[Order*Product] FROM R8)
       (MAINTAINING -item[Delivery*Product] \/ of;item[Order*Product] FROM R8)
       SELECT x:Order FROM codomain(of);
         INSERT INTO item[Order*Product] SELECTFROM
           V[Order*Delivery];((item[Delivery*Product] \/ Delta)/\-(of;item[Order*Prod
       (MAINTAINING -item[Delivery*Product] \/ of;item[Order*Product] FROM R8)
       SELECT x:Order FROM codomain(item[Order*Product]~);
         INSERT INTO of SELECTFROM
           ((item[Delivery*Product] \/ Delta)/\-(of;item[Order*Product]));V[Product*Or
       (MAINTAINING -item[Delivery*Product] \/ of;item[Order*Product] FROM R8)
       INSERT INTO item[Order*Product] SELECTFROM
         (of~;item[Delivery*Product] \/ of~;Delta)/\(of~;item[Delivery*Product] \/ -it
       (TO MAINTAIN -item[Delivery*Product] \/ of;item[Order*Product] FROM R8)
(MAINTAINING -item[Delivery*Product] \/ of;item[Order*Product] FROM R8)

ON DELETE Delta FROM item[Delivery*Product] EXECUTE    -- (ECA rule 2)
BLOCK
(CANNOT CHANGE -item[Delivery*Product] \/ of;item[Order*Product] FROM R8)

ON INSERT Delta IN item[Order*Product] EXECUTE    -- (ECA rule 3)
INSERT INTO item[Delivery*Product] SELECTFROM
  (of;item[Order*Product] \/ of;Delta)/\(of;item[Order*Product] \/ -item[Delivery*Prod
(TO MAINTAIN -(of;item[Order*Product]) \/ item[Delivery*Product] FROM R9)
```

```
ON DELETE Delta FROM item[Order*Product] EXECUTE    -- (ECA rule 4)
BLOCK
(CANNOT CHANGE -(of;item[Order*Product]) \/ item[Delivery*Product] FROM R9)

ON INSERT Delta IN of EXECUTE    -- (ECA rule 5)
ALL of INSERT INTO item[Delivery*Product] SELECTFROM
         (of;item[Order*Product] \/ Delta;item[Order*Product])/\(of;item[Order*Produc
       (TO MAINTAIN -(of;item[Order*Product]) \/ item[Delivery*Product] FROM R9)
       INSERT INTO I[Order] SELECTFROM
         (of~;of \/ of~;Delta \/ Delta~;of \/ Delta~;Delta)/\(of~;of \/ of~;Delta \/ I
       (TO MAINTAIN -(of~;of) \/ I[Order] FROM R0)
(MAINTAINING -(of;item[Order*Product]) \/ item[Delivery*Product] FROM R9)
(MAINTAINING -(of~;of) \/ I[Order] FROM R0)

ON DELETE Delta FROM of EXECUTE    -- (ECA rule 6)
BLOCK
(CANNOT CHANGE -(of;item[Order*Product]) \/ item[Delivery*Product] FROM R9)
(CANNOT CHANGE -(of~;of) \/ I[Order] FROM R0)

ON INSERT Delta IN rejected EXECUTE    -- (ECA rule 7)
ALL of ONE of INSERT INTO addressedTo~ SELECTFROM
                (rejected \/ Delta)/\(rejected \/ -addressedTo~)/\(-addressedTo~ \/ De
              (TO MAINTAIN -rejected \/ addressedTo~ FROM R10)
              INSERT INTO I[Provider] SELECTFROM
                (rejected;addressedTo \/ Delta;addressedTo)/\(rejected;addressedTo \/
              (TO MAINTAIN -rejected \/ addressedTo~ FROM R10)
       (MAINTAINING -rejected \/ addressedTo~ FROM R10)
       INSERT INTO I[Provider] SELECTFROM
         (rejected;rejected~ \/ rejected;Delta~ \/ Delta;rejected~ \/ Delta;Delta~)/\
       (TO MAINTAIN -(rejected;rejected~) \/ I[Provider] FROM R0)
(MAINTAINING -rejected \/ addressedTo~ FROM R10)
(MAINTAINING -(rejected;rejected~) \/ I[Provider] FROM R0)

ON DELETE Delta FROM rejected EXECUTE    -- (ECA rule 8)
BLOCK
(CANNOT CHANGE -rejected \/ addressedTo~ FROM R10)
(CANNOT CHANGE -(rejected;rejected~) \/ I[Provider] FROM R0)

ON INSERT Delta IN accepted EXECUTE    -- (ECA rule 9)
ALL of ONE of INSERT INTO addressedTo~ SELECTFROM
                (accepted \/ Delta)/\(accepted \/ -addressedTo~)/\(-addressedTo~ \/ De
              (TO MAINTAIN -accepted \/ addressedTo~ FROM R10)
              INSERT INTO I[Provider] SELECTFROM
                (accepted;addressedTo \/ Delta;addressedTo)/\(accepted;addressedTo \/
              (TO MAINTAIN -accepted \/ addressedTo~ FROM R10)
       (MAINTAINING -accepted \/ addressedTo~ FROM R10)
       INSERT INTO I[Provider] SELECTFROM
         (accepted;accepted~ \/ accepted;Delta~ \/ Delta;accepted~ \/ Delta;Delta~)/\
```

```
       (TO MAINTAIN -(accepted;accepted~) \/ I[Provider] FROM R0)
(MAINTAINING -accepted \/ addressedTo~ FROM R10)
(MAINTAINING -(accepted;accepted~) \/ I[Provider] FROM R0)

ON DELETE Delta FROM accepted EXECUTE    -- (ECA rule 10)
BLOCK
(CANNOT CHANGE -accepted \/ addressedTo~ FROM R10)
(CANNOT CHANGE -(accepted;accepted~) \/ I[Provider] FROM R0)

ON INSERT Delta IN provided EXECUTE    -- (ECA rule 11)
ALL of ONE of CREATE x:Order;
                ALL of INSERT INTO of~ SELECTFROM
                         V[Order*Provider];((provided \/ Delta)/\-(accepted;of~)) \/ -
                       INSERT INTO accepted SELECTFROM
                         ((provided \/ Delta)/\-(accepted;of~));V[Delivery*Order] \/ -
                (MAINTAINING -provided \/ accepted;of~ FROM R11)
              (MAINTAINING -provided \/ accepted;of~ FROM R11)
              SELECT x:Order FROM codomain(accepted);
                INSERT INTO of~ SELECTFROM
                  V[Order*Provider];((provided \/ Delta)/\-(accepted;of~)) \/ -of~
              (MAINTAINING -provided \/ accepted;of~ FROM R11)
              SELECT x:Order FROM codomain(of);
                INSERT INTO accepted SELECTFROM
                  ((provided \/ Delta)/\-(accepted;of~));V[Delivery*Order] \/ -accepte
              (MAINTAINING -provided \/ accepted;of~ FROM R11)
              INSERT INTO accepted SELECTFROM
                (provided;of \/ Delta;of)/\(provided;of \/ -accepted)/\(-accepted \/ 
              (TO MAINTAIN -provided \/ accepted;of~ FROM R11)
              INSERT INTO I[Provider] SELECTFROM
                (provided;of;accepted~ \/ Delta;of;accepted~)/\(provided;of;accepted~
              (TO MAINTAIN -provided \/ accepted;of~ FROM R11)
       (MAINTAINING -provided \/ accepted;of~ FROM R11)
       INSERT INTO I[Provider] SELECTFROM
         (provided;provided~ \/ provided;Delta~ \/ Delta;provided~ \/ Delta;Delta~)/\
       (TO MAINTAIN -(provided;provided~) \/ I[Provider] FROM R0)
(MAINTAINING -provided \/ accepted;of~ FROM R11)
(MAINTAINING -(provided;provided~) \/ I[Provider] FROM R0)

ON DELETE Delta FROM provided EXECUTE    -- (ECA rule 12)
BLOCK
(CANNOT CHANGE -(provided;provided~) \/ I[Provider] FROM R0)

ON INSERT Delta IN paid EXECUTE    -- (ECA rule 13)
ALL of ONE of INSERT INTO sentTo~ SELECTFROM
                (paid \/ Delta)/\(paid \/ -sentTo~)/\(-sentTo~ \/ Delta)/\(-sentTo~ \/
              (TO MAINTAIN -paid \/ sentTo~ FROM R12)
              INSERT INTO I[Client] SELECTFROM
                (paid;sentTo \/ Delta;sentTo)/\(paid;sentTo \/ -I[Client])/\(-I[Clien
              (TO MAINTAIN -paid \/ sentTo~ FROM R12)
```

```
          (MAINTAINING -paid \/ sentTo~ FROM R12)
          INSERT INTO I[Client] SELECTFROM
            (paid;paid~ \/ paid;Delta~ \/ Delta;paid~ \/ Delta;Delta~)/\(paid;paid~ \/ p
          (TO MAINTAIN -(paid;paid~) \/ I[Client] FROM R0)
(MAINTAINING -paid \/ sentTo~ FROM R12)
(MAINTAINING -(paid;paid~) \/ I[Client] FROM R0)

ON DELETE Delta FROM paid EXECUTE    -- (ECA rule 14)
BLOCK
(CANNOT CHANGE -paid \/ sentTo~ FROM R12)
(CANNOT CHANGE -(paid;paid~) \/ I[Client] FROM R0)

ON INSERT Delta IN sentTo EXECUTE    -- (ECA rule 15)
ALL of ONE of CREATE x:Delivery;
                ALL of ONE of CREATE x:Order;
                               ALL of INSERT INTO from[Order*Client] SELECTFROM
                                        V[Order*Delivery];V[Delivery*Invoice];((sent
                                      INSERT INTO of SELECTFROM
                                        V[Delivery*Invoice];((sentTo \/ Delta)/\-(de
                             SELECT x:Order FROM codomain(of);
                               INSERT INTO from[Order*Client] SELECTFROM
                                 V[Order*Delivery];V[Delivery*Invoice];((sentTo \/ D
                             SELECT x:Order FROM codomain(from[Order*Client]~);
                               INSERT INTO of SELECTFROM
                                 V[Delivery*Invoice];((sentTo \/ Delta)/\-(delivery;
                       INSERT INTO delivery SELECTFROM
                         ((sentTo \/ Delta)/\-(delivery;of;from[Order*Client]));V[Cli
                 (MAINTAINING -sentTo \/ delivery;of;from[Order*Client] FROM R13)
              (MAINTAINING -sentTo \/ delivery;of;from[Order*Client] FROM R13)
              SELECT x:Delivery FROM codomain(delivery);
                ONE of CREATE x:Order;
                         ALL of INSERT INTO from[Order*Client] SELECTFROM
                                 V[Order*Delivery];V[Delivery*Invoice];((sentTo \/ D
                                INSERT INTO of SELECTFROM
                                 V[Delivery*Invoice];((sentTo \/ Delta)/\-(delivery;
                       SELECT x:Order FROM codomain(of);
                         INSERT INTO from[Order*Client] SELECTFROM
                           V[Order*Delivery];V[Delivery*Invoice];((sentTo \/ Delta)/\
                       SELECT x:Order FROM codomain(from[Order*Client]~);
                         INSERT INTO of SELECTFROM
                           V[Delivery*Invoice];((sentTo \/ Delta)/\-(delivery;of;from
              (MAINTAINING -sentTo \/ delivery;of;from[Order*Client] FROM R13)
              SELECT x:Delivery FROM codomain(from[Order*Client]~;of~);
                INSERT INTO delivery SELECTFROM
                  ((sentTo \/ Delta)/\-(delivery;of;from[Order*Client]));V[Client*Del
              (MAINTAINING -sentTo \/ delivery;of;from[Order*Client] FROM R13)
              CREATE x:Order;
                ALL of INSERT INTO from[Order*Client] SELECTFROM
                         V[Order*Invoice];((sentTo \/ Delta)/\-(delivery;of;from[Orde
```

```
                      ONE of CREATE x:Delivery;
                               ALL of INSERT INTO of SELECTFROM
                                        V[Delivery*Invoice];((sentTo \/ Delta)/\-(del
                                      INSERT INTO delivery SELECTFROM
                                        ((sentTo \/ Delta)/\-(delivery;of;from[Order
                             SELECT x:Delivery FROM codomain(delivery);
                               INSERT INTO of SELECTFROM
                                 V[Delivery*Invoice];((sentTo \/ Delta)/\-(delivery;
                             SELECT x:Delivery FROM codomain(of~);
                               INSERT INTO delivery SELECTFROM
                                 ((sentTo \/ Delta)/\-(delivery;of;from[Order*Client]
               (MAINTAINING -sentTo \/ delivery;of;from[Order*Client] FROM R13)
             (MAINTAINING -sentTo \/ delivery;of;from[Order*Client] FROM R13)
             SELECT x:Order FROM codomain(delivery;of);
               INSERT INTO from[Order*Client] SELECTFROM
                 V[Order*Invoice];((sentTo \/ Delta)/\-(delivery;of;from[Order*Clien
             (MAINTAINING -sentTo \/ delivery;of;from[Order*Client] FROM R13)
             SELECT x:Order FROM codomain(from[Order*Client]~);
               ONE of CREATE x:Delivery;
                        ALL of INSERT INTO of SELECTFROM
                                 V[Delivery*Invoice];((sentTo \/ Delta)/\-(delivery;
                               INSERT INTO delivery SELECTFROM
                                 ((sentTo \/ Delta)/\-(delivery;of;from[Order*Client]
                      SELECT x:Delivery FROM codomain(delivery);
                        INSERT INTO of SELECTFROM
                          V[Delivery*Invoice];((sentTo \/ Delta)/\-(delivery;of;from
                      SELECT x:Delivery FROM codomain(of~);
                        INSERT INTO delivery SELECTFROM
                          ((sentTo \/ Delta)/\-(delivery;of;from[Order*Client]));V[C
             (MAINTAINING -sentTo \/ delivery;of;from[Order*Client] FROM R13)
             SELECT x:Order FROM codomain(of);
               INSERT INTO from[Order*Client] SELECTFROM
                 V[Order*Delivery];((delivery~;sentTo \/ delivery~;Delta)/\-(of;from
             (MAINTAINING -sentTo \/ delivery;of;from[Order*Client] FROM R13)
             SELECT x:Order FROM codomain(from[Order*Client]~);
               INSERT INTO of SELECTFROM
                 ((delivery~;sentTo \/ delivery~;Delta)/\-(of;from[Order*Client]));V
             (MAINTAINING -sentTo \/ delivery;of;from[Order*Client] FROM R13)
             INSERT INTO from[Order*Client] SELECTFROM
               (of~;delivery~;sentTo \/ of~;delivery~;Delta)/\(of~;delivery~;sentTo \
             (TO MAINTAIN -sentTo \/ delivery;of;from[Order*Client] FROM R13)
             INSERT INTO I[Client] SELECTFROM
               (from[Order*Client]~;of~;delivery~;sentTo \/ from[Order*Client]~;of~;
             (TO MAINTAIN -sentTo \/ delivery;of;from[Order*Client] FROM R13)
      (MAINTAINING -sentTo \/ delivery;of;from[Order*Client] FROM R13)
      INSERT INTO I[Client] SELECTFROM
        (sentTo~;sentTo \/ sentTo~;Delta \/ Delta~;sentTo \/ Delta~;Delta)/\(sentTo~
      (TO MAINTAIN -(sentTo~;sentTo) \/ I[Client] FROM R0)
(MAINTAINING -sentTo \/ delivery;of;from[Order*Client] FROM R13)
(MAINTAINING -(sentTo~;sentTo) \/ I[Client] FROM R0)
```

```
ON DELETE Delta FROM sentTo EXECUTE    -- (ECA rule 16)
BLOCK
(CANNOT CHANGE -(sentTo~;sentTo) \/ I[Client] FROM R0)

ON INSERT Delta IN from[Order*Client] EXECUTE    -- (ECA rule 17)
INSERT INTO I[Client] SELECTFROM
  (from[Order*Client]~;I[Order];from[Order*Client] \/ from[Order*Client]~;Delta \/ De
(TO MAINTAIN -(from[Order*Client]~;I[Order];from[Order*Client]) \/ I[Client] FROM R0)

ON DELETE Delta FROM from[Order*Client] EXECUTE    -- (ECA rule 18)
BLOCK
(CANNOT CHANGE -(from[Order*Client]~;I[Order];from[Order*Client]) \/ I[Client] FROM R

ON INSERT Delta IN I[Delivery] EXECUTE    -- (ECA rule 19)
ALL of ONE of CREATE x:Order;
                INSERT INTO of~ SELECTFROM
                  V[Order*Delivery];((I[Delivery] \/ Delta)/\-(of;of~)) \/ -of~
              (MAINTAINING -I[Delivery] \/ of;of~ FROM R0)
              SELECT x:Order FROM codomain(of);
                INSERT INTO of~ SELECTFROM
                  V[Order*Delivery];((I[Delivery] \/ Delta)/\-(of;of~)) \/ -of~
              (MAINTAINING -I[Delivery] \/ of;of~ FROM R0)
       (MAINTAINING -I[Delivery] \/ of;of~ FROM R0)
       ONE of CREATE x:Provider;
                INSERT INTO provided SELECTFROM
                  V[Provider*Delivery];((I[Delivery] \/ Delta)/\-(provided~;provided)
              (MAINTAINING -I[Delivery] \/ provided~;provided FROM R0)
              SELECT x:Provider FROM codomain(provided~);
                INSERT INTO provided SELECTFROM
                  V[Provider*Delivery];((I[Delivery] \/ Delta)/\-(provided~;provided)
              (MAINTAINING -I[Delivery] \/ provided~;provided FROM R0)
       (MAINTAINING -I[Delivery] \/ provided~;provided FROM R0)
       ONE of CREATE x:Client;
                INSERT INTO deliveredTo~ SELECTFROM
                  V[Client*Delivery];((I[Delivery] \/ Delta)/\-(deliveredTo;delivered
              (MAINTAINING -I[Delivery] \/ deliveredTo;deliveredTo~ FROM R0)
              SELECT x:Client FROM codomain(deliveredTo);
                INSERT INTO deliveredTo~ SELECTFROM
                  V[Client*Delivery];((I[Delivery] \/ Delta)/\-(deliveredTo;delivered
              (MAINTAINING -I[Delivery] \/ deliveredTo;deliveredTo~ FROM R0)
       (MAINTAINING -I[Delivery] \/ deliveredTo;deliveredTo~ FROM R0)
(MAINTAINING -I[Delivery] \/ of;of~ FROM R0)
(MAINTAINING -I[Delivery] \/ provided~;provided FROM R0)
(MAINTAINING -I[Delivery] \/ deliveredTo;deliveredTo~ FROM R0)

ON DELETE Delta FROM I[Delivery] EXECUTE    -- (ECA rule 20)
BLOCK
(CANNOT CHANGE -I[Delivery] \/ of;of~ FROM R0)
(CANNOT CHANGE -I[Delivery] \/ provided~;provided FROM R0)
(CANNOT CHANGE -I[Delivery] \/ deliveredTo;deliveredTo~ FROM R0)
```

```
ON INSERT Delta IN addressedTo EXECUTE    -- (ECA rule 21)
INSERT INTO I[Provider] SELECTFROM
  (addressedTo~;addressedTo \/ addressedTo~;Delta \/ Delta~;addressedTo \/ Delta~;Del
(TO MAINTAIN -(addressedTo~;addressedTo) \/ I[Provider] FROM R0)

ON DELETE Delta FROM addressedTo EXECUTE    -- (ECA rule 22)
BLOCK
(CANNOT CHANGE -(addressedTo~;addressedTo) \/ I[Provider] FROM R0)

ON INSERT Delta IN I[Order] EXECUTE    -- (ECA rule 23)
ONE of CREATE x:Provider;
         INSERT INTO addressedTo~ SELECTFROM
           V[Provider*Order];((I[Order] \/ Delta)/\-(addressedTo;addressedTo~)) \/ -ad
       (MAINTAINING -I[Order] \/ addressedTo;addressedTo~ FROM R0)
       SELECT x:Provider FROM codomain(addressedTo);
         INSERT INTO addressedTo~ SELECTFROM
           V[Provider*Order];((I[Order] \/ Delta)/\-(addressedTo;addressedTo~)) \/ -ad
       (MAINTAINING -I[Order] \/ addressedTo;addressedTo~ FROM R0)
(MAINTAINING -I[Order] \/ addressedTo;addressedTo~ FROM R0)

ON DELETE Delta FROM I[Order] EXECUTE    -- (ECA rule 24)
BLOCK
(CANNOT CHANGE -I[Order] \/ addressedTo;addressedTo~ FROM R0)

ON INSERT Delta IN deliveredTo EXECUTE    -- (ECA rule 25)
INSERT INTO I[Client] SELECTFROM
  (deliveredTo~;deliveredTo \/ deliveredTo~;Delta \/ Delta~;deliveredTo \/ Delta~;Del
(TO MAINTAIN -(deliveredTo~;deliveredTo) \/ I[Client] FROM R0)

ON DELETE Delta FROM deliveredTo EXECUTE    -- (ECA rule 26)
BLOCK
(CANNOT CHANGE -(deliveredTo~;deliveredTo) \/ I[Client] FROM R0)

ON INSERT Delta IN I[Invoice] EXECUTE    -- (ECA rule 27)
ALL of ONE of CREATE x:Client;
                INSERT INTO sentTo~ SELECTFROM
                  V[Client*Invoice];((I[Invoice] \/ Delta)/\-(sentTo;sentTo~)) \/ -se
              (MAINTAINING -I[Invoice] \/ sentTo;sentTo~ FROM R0)
              SELECT x:Client FROM codomain(sentTo);
                INSERT INTO sentTo~ SELECTFROM
                  V[Client*Invoice];((I[Invoice] \/ Delta)/\-(sentTo;sentTo~)) \/ -se
              (MAINTAINING -I[Invoice] \/ sentTo;sentTo~ FROM R0)
       (MAINTAINING -I[Invoice] \/ sentTo;sentTo~ FROM R0)
       ONE of CREATE x:Delivery;
                INSERT INTO delivery~ SELECTFROM
                  V[Delivery*Invoice];((I[Invoice] \/ Delta)/\-(delivery;delivery~))
              (MAINTAINING -I[Invoice] \/ delivery;delivery~ FROM R0)
              SELECT x:Delivery FROM codomain(delivery);
```

```
                INSERT INTO delivery~ SELECTFROM
                  V[Delivery*Invoice];((I[Invoice] \/ Delta)/\-(delivery;delivery~))
              (MAINTAINING -I[Invoice] \/ delivery;delivery~ FROM R0)
       (MAINTAINING -I[Invoice] \/ delivery;delivery~ FROM R0)
       ONE of CREATE x:Provider;
                INSERT INTO from[Invoice*Provider]~ SELECTFROM
                  V[Provider*Invoice];((I[Invoice] \/ Delta)/\-(from[Invoice*Provider]
              (MAINTAINING -I[Invoice] \/ from[Invoice*Provider];I[Provider];from[Inv
              SELECT x:Provider FROM codomain(from[Invoice*Provider]);
                INSERT INTO from[Invoice*Provider]~ SELECTFROM
                  V[Provider*Invoice];((I[Invoice] \/ Delta)/\-(from[Invoice*Provider]
              (MAINTAINING -I[Invoice] \/ from[Invoice*Provider];I[Provider];from[Inv
       (MAINTAINING -I[Invoice] \/ from[Invoice*Provider];I[Provider];from[Invoice*Pr
(MAINTAINING -I[Invoice] \/ sentTo;sentTo~ FROM R0)
(MAINTAINING -I[Invoice] \/ delivery;delivery~ FROM R0)
(MAINTAINING -I[Invoice] \/ from[Invoice*Provider];I[Provider];from[Invoice*Provider]

ON DELETE Delta FROM I[Invoice] EXECUTE    -- (ECA rule 28)
BLOCK
(CANNOT CHANGE -I[Invoice] \/ sentTo;sentTo~ FROM R0)
(CANNOT CHANGE -I[Invoice] \/ delivery;delivery~ FROM R0)
(CANNOT CHANGE -I[Invoice] \/ from[Invoice*Provider];I[Provider];from[Invoice*Provide

ON INSERT Delta IN delivery EXECUTE    -- (ECA rule 29)
INSERT INTO I[Delivery] SELECTFROM
  (delivery~;delivery \/ delivery~;Delta \/ Delta~;delivery \/ Delta~;Delta)/\(delive
(TO MAINTAIN -(delivery~;delivery) \/ I[Delivery] FROM R0)

ON DELETE Delta FROM delivery EXECUTE    -- (ECA rule 30)
BLOCK
(CANNOT CHANGE -(delivery~;delivery) \/ I[Delivery] FROM R0)

ON INSERT Delta IN from[Invoice*Provider] EXECUTE    -- (ECA rule 31)
INSERT INTO I[Provider] SELECTFROM
  (from[Invoice*Provider]~;I[Invoice];from[Invoice*Provider] \/ from[Invoice*Provider]
(TO MAINTAIN -(from[Invoice*Provider]~;I[Invoice];from[Invoice*Provider]) \/ I[Provid

ON DELETE Delta FROM from[Invoice*Provider] EXECUTE    -- (ECA rule 32)
BLOCK
(CANNOT CHANGE -(from[Invoice*Provider]~;I[Invoice];from[Invoice*Provider]) \/ I[Prov

ON INSERT Delta IN sProvider EXECUTE    -- (ECA rule 33)
INSERT INTO I[Provider] SELECTFROM
  (sProvider~;sProvider \/ sProvider~;Delta \/ Delta~;sProvider \/ Delta~;Delta)/\(sP
(TO MAINTAIN -(sProvider~;sProvider) \/ I[Provider] FROM R0)

ON DELETE Delta FROM sProvider EXECUTE    -- (ECA rule 34)
BLOCK
(CANNOT CHANGE -(sProvider~;sProvider) \/ I[Provider] FROM R0)
```

```
ON INSERT Delta IN sClient EXECUTE    -- (ECA rule 35)
INSERT INTO I[Client] SELECTFROM
  (sClient~;sClient \/ sClient~;Delta \/ Delta~;sClient \/ Delta~;Delta)/\(sClient~;s
(TO MAINTAIN -(sClient~;sClient) \/ I[Client] FROM R0)

ON DELETE Delta FROM sClient EXECUTE    -- (ECA rule 36)
BLOCK
(CANNOT CHANGE -(sClient~;sClient) \/ I[Client] FROM R0)
```