Functional Specification of 'Delivery'

Put author(s) here
(This document was generated by Ampersand vs. 2.1.0.52)

Sun Apr 24 13:14:37 W. Europe Daylight Time 2011

Contents

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 shows remaining fact types and concepts that have not been used in previous paragraphs.

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.

2.3 Loose ends...

This paragraph shows remaining fact types and concepts that have not been used in previous paragraphs.

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.

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 F:\RJ$\Prive\CC model repository\Adlfiles\Delivery.adl) is not documented.

All rules in process Delivery are linked to roles.

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

The following table represents the population of various relations.

Relation	Population
$from: \ Order \times Client$	3
$item: \ Order \times Product$	3
$item: \ Delivery \times Product$	3
$of: \ Delivery \times Order$	3
$provided: \ Provider \times Delivery$	3
$accepted: \ Provider \times Order$	3
$rejected: \ Provider \times Order$	3
$addressedTo: \ Order \times Provider$	3
$deliveredTo: \ Delivery \times Client$	3
$sentTo: \ Invoice \times Client$	3
$delivery: \ Invoice \times Delivery$	3
$from: \ Invoice \times Provider$	3
$paid: \ Client \times Invoice$	3
$I_{[Order]}$	3
$I_{[Client]}$	3
$I_{[Product]}$	3
$I_{[Delivery]}$	3
$I_{[Provider]}$	2
$I_{[Invoice]}$	3
$I_{[Session]}$	0

The population in this script violates no rule, nor does it specify any work in progress.

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.

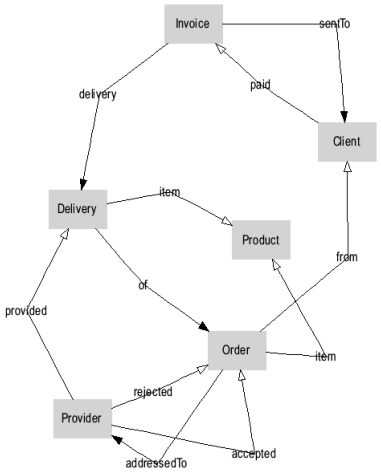

Figure 4.1: Conceptual model of Deliveries

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.

$$item : Delivery \times Product \tag{4.1}$$
$$item : Order \times Product \tag{4.2}$$
$$of : Delivery \rightarrow Order \tag{4.3}$$

$$item \cap \overline{(of ; item)} \tag{4.4}$$

This means: Each item in a delivery is mentioned on the order.

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).

$$of ; item \cap \overline{item} \tag{4.5}$$

This means: Every item on an order must also be in the corresponding delivery.

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.

$$rejected : Provider \times Order \tag{4.6}$$
$$accepted : Provider \times Order \tag{4.7}$$
$$addressedTo : Order \rightarrow Provider \tag{4.8}$$

$$accepted \cap \overline{addressedTo} \cup rejected \cap \overline{addressedTo} \tag{4.9}$$

This means: A provider can only accept or reject orders that are addressed to that provider.

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):

$$provided : Provider \times Delivery \tag{4.10}$$

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

$$provided \cap \overline{(accepted ; of\ )} \tag{4.11}$$

This means: For every delivery a provider has made, there exists an accepted order.

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.

$$paid \quad : \quad Client \times Invoice \tag{4.12}$$
$$sentTo \quad : \quad Invoice \rightarrow Client \tag{4.13}$$

$$paid \cap \overline{sentTo} \tag{4.14}$$

This means: Payments are made only for invoices sent.

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.

$$from \quad : \quad Order \times Client \tag{4.15}$$
$$delivery \quad : \quad Invoice \rightarrow Delivery \tag{4.16}$$

We also use definitions 4.13 ($sentTo$) and 4.3 (of).

$$sentTo \cap \overline{(delivery; of; from)} \tag{4.17}$$

This means: Invoices are sent to a customer only if there is a delivery made to that customer.

This corresponds to requirement 2.1 on page 5.

4.2 Sessions

Figure 4.2 shows a conceptual diagram of this theme.

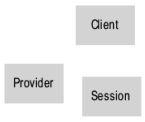

Figure 4.2: Conceptual model of Sessions

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.

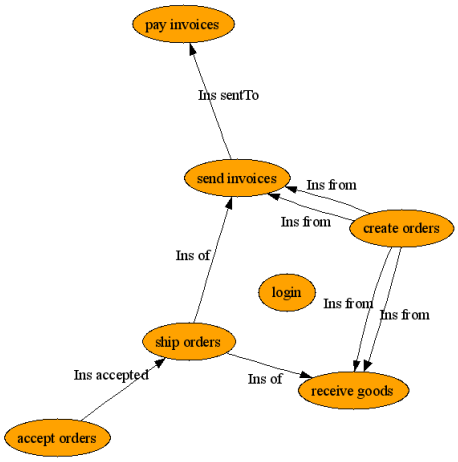

Figure 5.1: Process model of Delivery

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

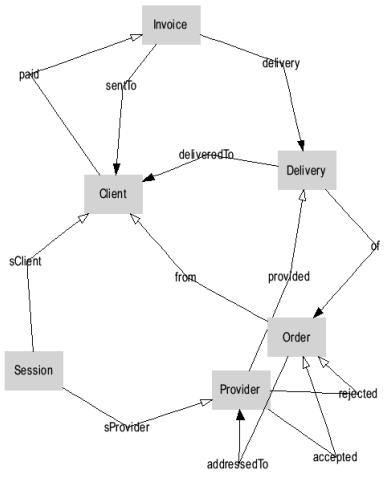

Figure 5.2: Basic sentences of Delivery

login We use definitions **??** ($sProvider$) and **??** ($sClient$). Activity is signalled by:

$$I_{[Session]} \cap (\overline{(sProvider; V_{[Provider \times Session]})} \cup \overline{(sClient; V_{[Client \times Session]})}) \cap \overline{(sClient; V_{[Client \times Session]})} \cap (\overline{sP} \quad (5.1)$$

This means: 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).

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). Activity is signalled by:

$$I_{[Order]} \cap \overline{(from; I_{[Client]}; from\ )} \tag{5.2}$$

This means: Each order is created by precisely one client.

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*). Activity is signalled by:

$$addressedTo \cap \overline{accepted} \cap \overline{rejected} \tag{5.3}$$

This means: Orders addressed to a provider must be accepted or rejected by that provider.

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*). Activity is signalled by:

$$accepted \cap \overline{(provided; of)} \tag{5.4}$$

This means: Each order that has been accepted by a provider must (eventually) be shipped by that provider.

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

We use definitions 4.13 (*sentTo*) and 4.12 (*paid*). Activity is signalled by:

$$sentTo \cap \overline{paid} \tag{5.5}$$

This means: All invoices sent to a customer must be paid by that customer.

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*). Activity is signalled by:

$$of; from \cap \overline{deliveredTo} \tag{5.6}$$

This means: Every delivery must be acknowledged by the client who placed the corresponding order.

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*). Activity is signalled by:

$$delivery; of; from \cap \overline{sentTo} \tag{5.7}$$

This means: After a delivery has been made to a customer, an invoice must be sent.

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

service	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

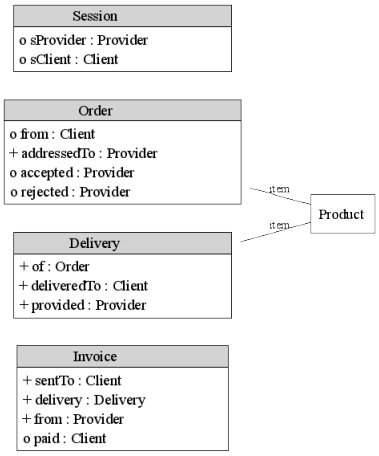

Figure 7.2: Data model of Delivery

7.1 Invoice

The attributes in Invoice have the following multiplicity constraints.

attribute	type	mandatory	unique
key	Invoice	√	√
sentTo	Client	√	
delivery	Delivery	√	
from	Provider	√	
paid	Client		

7.2 Order

The attributes in Order have the following multiplicity constraints.

attribute	type	mandatory	unique
key	Order	√	√
from	Client		
addressedTo	Provider	√	
accepted	Provider		
rejected	Provider		

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

$$\overline{I_{[Order]}} \cup from; I_{[Client]}; from$$

7.3 Delivery

The attributes in Delivery have the following multiplicity constraints.

attribute	type	mandatory	unique
key	Delivery	√	√
of	Order	√	
deliveredTo	Client	√	
provided	Provider	√	

7.4 Session

The attributes in Session have the following multiplicity constraints.

attribute	type	mandatory	unique
key	Session	√	√
sProvider	Provider		
sClient	Client		

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

$$\overline{I_{[Session]}} \cup sProvider; V_{[Provider \times Session]} \cap \overline{sClient}; V_{[Client \times Session]} \cup sClient; V_{[Client \times Session]} \cup \overline{sProvider}; V_{[Prov}$$

7.5 item1

The attributes in item1 have the following multiplicity constraints.

attribute	type	mandatory	unique
key	Order	√	
Product	Product	√	

7.6 item2

The attributes in item2 have the following multiplicity constraints.

attribute	type	mandatory	unique
key	Delivery	√	
Product	Product	√	

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	I[Session]	ECA rule 1
Del	I[Session]	error: rule 'login'
Ins	sProvider[Session*Provider]	error: rule 'login'
Del	sProvider[Session*Provider]	ECA rule 4
Ins	sClient[Session*Client]	error: rule 'login'
Del	sClient[Session*Client]	ECA rule 6

Figure 8.1 shows the knowledge graph of this service.

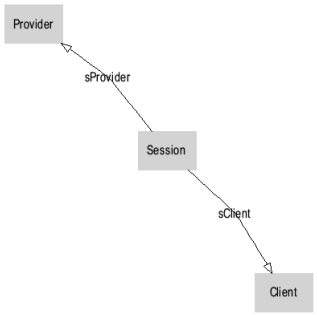

Figure 8.1: Language diagram of login

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 service.

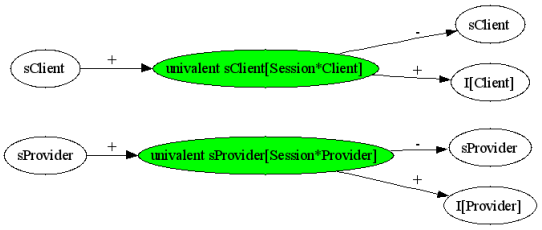

Figure 8.2: Switchboard of login

ECA rules:
temporarily not documented

```
ON INSERT Delta IN I[Session] EXECUTE    -- (ECA rule 1)
     ALL of
            ONE of
                   CREATE x:Provider;
                       ALL of
                              INSERT INTO V[Provider*Session] SELECTFROM 'x'[Provider
                              INSERT INTO sProvider SELECTFROM ((I[Session] \/ Delta)
                   SELECT x:Provider FROM codomain(sProvider);
                       INSERT INTO V[Provider*Session] SELECTFROM 'x'[Provider];V[Pro
                   SELECT x:Provider FROM codomain(V[Session*Provider]);
                       INSERT INTO sProvider SELECTFROM ((I[Session] \/ Delta)/\(I[Se
                   CREATE x:Client;
                       ALL of
                              INSERT INTO V[Client*Session] SELECTFROM 'x'[Client];V[
                              INSERT INTO sClient SELECTFROM ((I[Session] \/ Delta)/\
                   SELECT x:Client FROM codomain(sClient);
                       INSERT INTO V[Client*Session] SELECTFROM 'x'[Client];V[Client*
                   SELECT x:Client FROM codomain(V[Session*Client]);
                       INSERT INTO sClient SELECTFROM ((I[Session] \/ Delta)/\(I[Sess
                   SELECT x:Provider FROM codomain(-sProvider);
                       INSERT INTO V[Provider*Session] SELECTFROM 'x'[Provider];V[Pro
                   SELECT x:Provider FROM codomain(V[Session*Provider]);
                       DELETE FROM sProvider SELECTFROM ((I[Session] \/ Delta)/\(I[Se
            ONE of
                   CREATE x:Client;
```

```
                        ALL of
                               INSERT INTO V[Client*Session] SELECTFROM 'x'[Client];V[
                               DELETE FROM sClient SELECTFROM ((I[Session] \/ Delta)/\
                    SELECT x:Client FROM codomain(-sClient);
                        INSERT INTO V[Client*Session] SELECTFROM 'x'[Client];V[Client*
                    SELECT x:Client FROM codomain(V[Session*Client]);
                        DELETE FROM sClient SELECTFROM ((I[Session] \/ Delta)/\(I[Sess
                    SELECT x:Client FROM codomain(sClient);
                        INSERT INTO V[Client*Session] SELECTFROM 'x'[Client];V[Client*
                    SELECT x:Client FROM codomain(V[Session*Client]);
                        INSERT INTO sClient SELECTFROM ((I[Session] \/ Delta)/\(I[Sess
                    CREATE x:Provider;
                        ALL of
                               INSERT INTO V[Provider*Session] SELECTFROM 'x'[Provider]
                               DELETE FROM sProvider SELECTFROM ((I[Session] \/ Delta),
                    SELECT x:Provider FROM codomain(-sProvider);
                        INSERT INTO V[Provider*Session] SELECTFROM 'x'[Provider];V[Pro
                    SELECT x:Provider FROM codomain(V[Session*Provider]);
                        DELETE FROM sProvider SELECTFROM ((I[Session] \/ Delta)/\(I[Se
    (MAINTAINING -I[Session] \/ sProvider;V[Provider*Session]/\-sClient;V[Client*Sess

ON DELETE Delta FROM I[Session] EXECUTE    -- (ECA rule 2)
    BLOCK
    (CANNOT CHANGE -I[Session] \/ sProvider;V[Provider*Session]/\-sClient;V[Client*S

ON INSERT Delta IN sProvider EXECUTE    -- (ECA rule 3)
    BLOCK
    (CANNOT CHANGE -I[Session] \/ sProvider;V[Provider*Session]/\-sClient;V[Client*S

ON DELETE Delta FROM sProvider EXECUTE    -- (ECA rule 4)
    DELETE FROM I[Session] SELECTFROM I[Session]/\(-(sProvider;V[Provider*Session])/
    (TO MAINTAIN -I[Session] \/ sProvider;V[Provider*Session]/\-sClient;V[Client*Ses

ON INSERT Delta IN sClient EXECUTE    -- (ECA rule 5)
    BLOCK
    (CANNOT CHANGE -I[Session] \/ sProvider;V[Provider*Session]/\-sClient;V[Client*S

ON DELETE Delta FROM sClient EXECUTE    -- (ECA rule 6)
    DELETE FROM I[Session] SELECTFROM I[Session]/\(-(sProvider;V[Provider*Session])
    (TO MAINTAIN -I[Session] \/ sProvider;V[Provider*Session]/\-sClient;V[Client*Ses
```

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. Rule ‘‘correct invoices’’ will be maintained automatically, without intervention of a user. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	I[Order]	ECA rule 7
Del	I[Order]	error: rule ‘create orders’
Ins	from[Order*Client]	ECA rule 15
Del	from[Order*Client]	error: rule ‘receive goods’ and ‘send invoices’

Figure 9.1 shows the knowledge graph of this service.

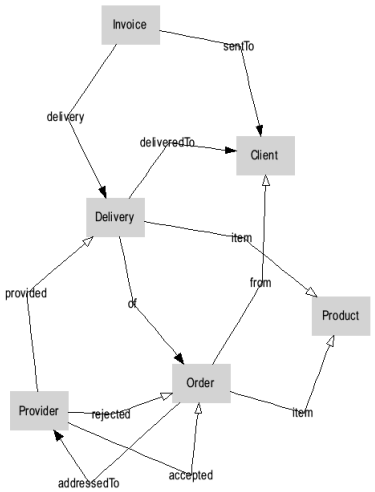

Figure 9.1: Language diagram of create orders

Figure 9.2 shows the switchboard diagram of this service.

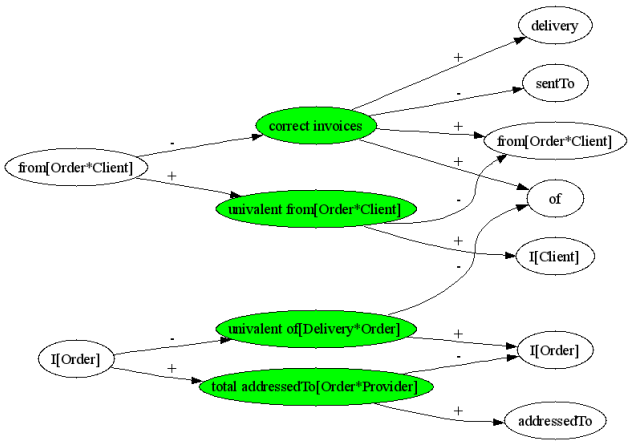

Figure 9.2: Switchboard of create orders

ECA rules:
temporarily not documented

```
ON INSERT Delta IN I[Order] EXECUTE    -- (ECA rule 7)
     ONE of
            CREATE x:Client;
                INSERT INTO from[Order*Client]~ SELECTFROM V[Client*Order];((I[Order]
            (MAINTAINING -I[Order] \/ from[Order*Client];I[Client];from[Order*Client]
            SELECT x:Client FROM codomain(from[Order*Client]);
                INSERT INTO from[Order*Client]~ SELECTFROM V[Client*Order];((I[Order]
            (MAINTAINING -I[Order] \/ from[Order*Client];I[Client];from[Order*Client]
     (MAINTAINING -I[Order] \/ from[Order*Client];I[Client];from[Order*Client]~ FROM

ON DELETE Delta FROM I[Order] EXECUTE    -- (ECA rule 8)
     BLOCK
     (CANNOT CHANGE -I[Order] \/ from[Order*Client];I[Client];from[Order*Client]~ FRO

ON INSERT Delta IN from[Order*Client] EXECUTE    -- (ECA rule 15)
     ALL of
            ONE of
                   INSERT INTO deliveredTo SELECTFROM (of;from[Order*Client] \/ of;De
                   (TO MAINTAIN -(of;from[Order*Client]) \/ deliveredTo FROM R6)
                   INSERT INTO I[Client] SELECTFROM (deliveredTo~;of;from[Order*Clien
                   (TO MAINTAIN -(of;from[Order*Client]) \/ deliveredTo FROM R6)
            (MAINTAINING -(of;from[Order*Client]) \/ deliveredTo FROM R6)
            ONE of
                   INSERT INTO sentTo SELECTFROM (delivery;of;from[Order*Client] \/ d
                   (TO MAINTAIN -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
                   INSERT INTO I[Client] SELECTFROM (sentTo~;delivery;of;from[Order*C
```

```
                    (TO MAINTAIN -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
               (MAINTAINING -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
     (MAINTAINING -(of;from[Order*Client]) \/ deliveredTo FROM R6)
     (MAINTAINING -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)

ON DELETE Delta FROM from[Order*Client] EXECUTE    -- (ECA rule 16)
     BLOCK
     (CANNOT CHANGE -(of;from[Order*Client]) \/ deliveredTo FROM R6)
     (CANNOT CHANGE -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
```

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. Rules 'proper address' and 'order based delivery' will be maintained automatically, without intervention of a user. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	addressedTo[Order*Provider]	ECA rule 9
Del	addressedTo[Order*Provider]	no op
Ins	accepted[Provider*Order]	ECA rule 11
Del	accepted[Provider*Order]	error: rule 'ship orders'

Figure 10.1 shows the knowledge graph of this service.

Figure 10.2 shows the switchboard diagram of this service.

ECA rules:
temporarily not documented

```
ON INSERT Delta IN addressedTo EXECUTE    -- (ECA rule 9)
     ONE of
            INSERT INTO accepted SELECTFROM (addressedTo~ \/ Delta~)/\(addressedTo~ \
            (TO MAINTAIN -addressedTo~ \/ accepted \/ rejected FROM R3)
            INSERT INTO rejected SELECTFROM (addressedTo~ \/ Delta~)/\(addressedTo~ \
            (TO MAINTAIN -addressedTo~ \/ accepted \/ rejected FROM R3)
     (MAINTAINING -addressedTo~ \/ accepted \/ rejected FROM R3)
```

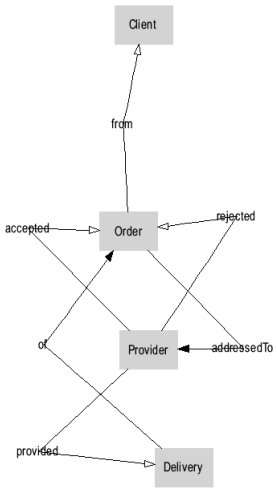

Figure 10.1: Language diagram of accept orders

```
ON INSERT Delta IN accepted EXECUTE    -- (ECA rule 11)
     ONE of
            CREATE x:Delivery;
                ALL of
                       INSERT INTO of SELECTFROM V[Delivery*Provider];((accepted \/ D
                       INSERT INTO provided SELECTFROM ((accepted \/ Delta)/\-(provid
                (MAINTAINING -accepted \/ provided;of FROM R4)
            (MAINTAINING -accepted \/ provided;of FROM R4)
            SELECT x:Delivery FROM codomain(provided);
                INSERT INTO of SELECTFROM V[Delivery*Provider];((accepted \/ Delta)/\
            (MAINTAINING -accepted \/ provided;of FROM R4)
            SELECT x:Delivery FROM codomain(of~);
                INSERT INTO provided SELECTFROM ((accepted \/ Delta)/\-(provided;of))
            (MAINTAINING -accepted \/ provided;of FROM R4)
     (MAINTAINING -accepted \/ provided;of FROM R4)

ON DELETE Delta FROM accepted EXECUTE    -- (ECA rule 12)
     BLOCK
     (CANNOT CHANGE -accepted \/ provided;of FROM R4)
```

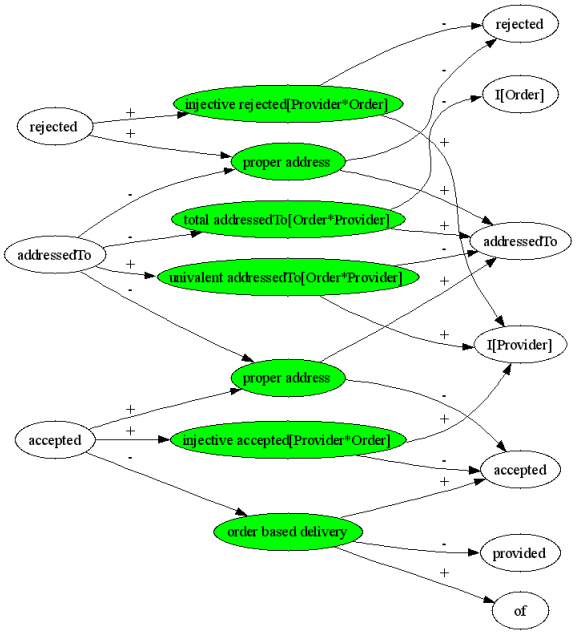

Figure 10.2: Switchboard of accept orders

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. Rules 'correct delivery', 'complete delivery', 'proper address', 'order based delivery', and 'correct invoices' will be maintained automatically, without intervention of a user. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	accepted[Provider*Order]	ECA rule 11
Del	accepted[Provider*Order]	error: rule 'ship orders'
Ins	of[Delivery*Order]	ECA rule 17
Del	of[Delivery*Order]	error: rule 'receive goods' and 'send invoices'

Figure 11.1 shows the knowledge graph of this service.

Figure 11.2 shows the switchboard diagram of this service.

ECA rules:
temporarily not documented

```
ON INSERT Delta IN accepted EXECUTE    -- (ECA rule 11)
     ONE of
            CREATE x:Delivery;
                ALL of
                       INSERT INTO of SELECTFROM V[Delivery*Provider];((accepted \/ D
                       INSERT INTO provided SELECTFROM ((accepted \/ Delta)/\-(provid
                (MAINTAINING -accepted \/ provided;of FROM R4)
            (MAINTAINING -accepted \/ provided;of FROM R4)
            SELECT x:Delivery FROM codomain(provided);
                INSERT INTO of SELECTFROM V[Delivery*Provider];((accepted \/ Delta)/\
            (MAINTAINING -accepted \/ provided;of FROM R4)
            SELECT x:Delivery FROM codomain(of~);
                INSERT INTO provided SELECTFROM ((accepted \/ Delta)/\-(provided;of))
```

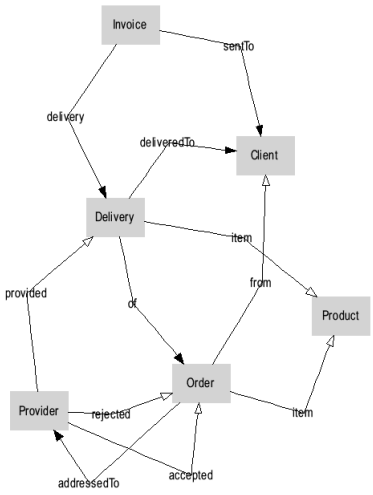

Figure 11.1: Language diagram of ship orders

```
            (MAINTAINING -accepted \/ provided;of FROM R4)
      (MAINTAINING -accepted \/ provided;of FROM R4)

ON DELETE Delta FROM accepted EXECUTE    -- (ECA rule 12)
      BLOCK
      (CANNOT CHANGE -accepted \/ provided;of FROM R4)

ON INSERT Delta IN of EXECUTE    -- (ECA rule 17)
      ALL of
            ONE of
                  INSERT INTO deliveredTo SELECTFROM (of;from[Order*Client] \/ Delta
                  (TO MAINTAIN -(of;from[Order*Client]) \/ deliveredTo FROM R6)
                  INSERT INTO I[Client] SELECTFROM (deliveredTo~;of;from[Order*Clien
                  (TO MAINTAIN -(of;from[Order*Client]) \/ deliveredTo FROM R6)
            (MAINTAINING -(of;from[Order*Client]) \/ deliveredTo FROM R6)
            ONE of
                  INSERT INTO sentTo SELECTFROM (delivery;of;from[Order*Client] \/ d
                  (TO MAINTAIN -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
                  INSERT INTO I[Client] SELECTFROM (sentTo~;delivery;of;from[Order*C
                  (TO MAINTAIN -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
            (MAINTAINING -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
      (MAINTAINING -(of;from[Order*Client]) \/ deliveredTo FROM R6)
      (MAINTAINING -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)

ON DELETE Delta FROM of EXECUTE    -- (ECA rule 18)
      BLOCK
      (CANNOT CHANGE -(of;from[Order*Client]) \/ deliveredTo FROM R6)
      (CANNOT CHANGE -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
```

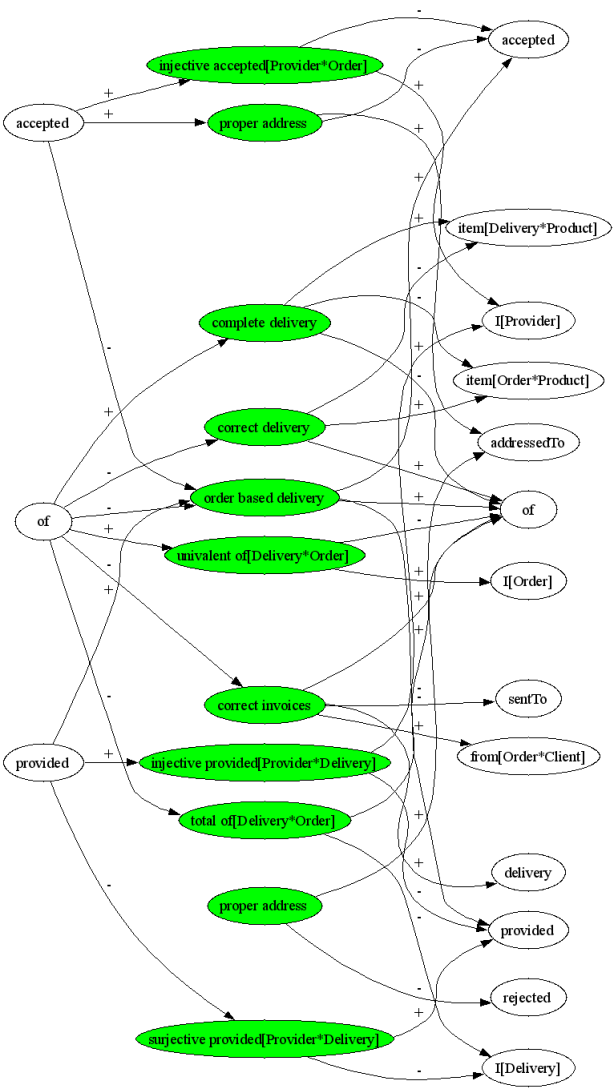

Figure 11.2: Switchboard of ship orders

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. Rules 'correct payments' and 'correct invoices' will be maintained automatically, without intervention of a user. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	sentTo[Invoice*Client]	ECA rule 13
Del	sentTo[Invoice*Client]	no op

Figure 12.1 shows the knowledge graph of this service.

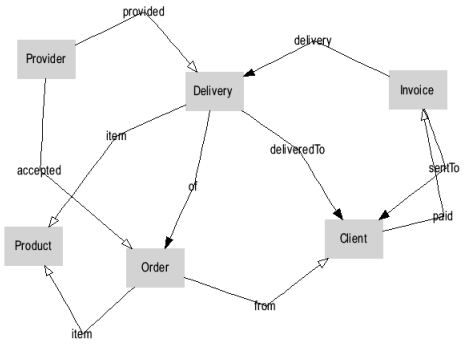

Figure 12.1: Language diagram of pay invoices

Figure 12.2 shows the switchboard diagram of this service.

ECA rules:
temporarily not documented

```
ON INSERT Delta IN sentTo EXECUTE    -- (ECA rule 13)
```

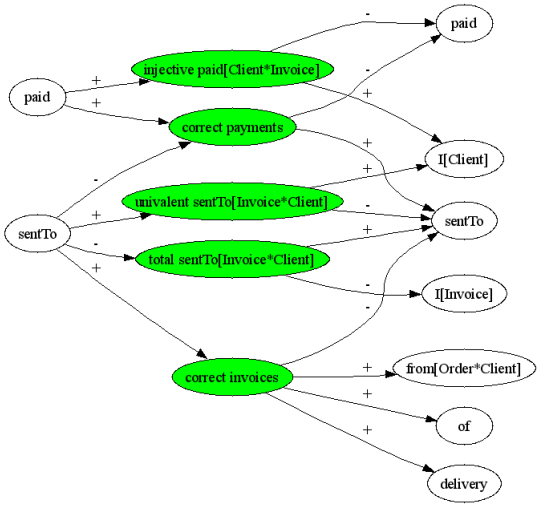

Figure 12.2: Switchboard of pay invoices

```
ONE of
       INSERT INTO paid SELECTFROM (sentTo~ \/ Delta~)/\(sentTo~ \/ -paid)/\(-pa
       (TO MAINTAIN -sentTo~ \/ paid FROM R5)
       INSERT INTO I[Client] SELECTFROM (sentTo~;paid~ \/ Delta~;paid~)/\(sentTo
       (TO MAINTAIN -sentTo~ \/ paid FROM R5)
(MAINTAINING -sentTo~ \/ paid FROM R5)
```

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. Rules 'correct delivery', 'complete delivery', 'order based delivery', and 'correct invoices' will be maintained automatically, without intervention of a user. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	from[Order*Client]	ECA rule 15
Del	from[Order*Client]	error: rule 'receive goods' and 'send invoices'
Ins	of[Delivery*Order]	ECA rule 17
Del	of[Delivery*Order]	error: rule 'receive goods' and 'send invoices'

Figure 13.1 shows the knowledge graph of this service.

Figure 13.2 shows the switchboard diagram of this service.

ECA rules:
temporarily not documented

```
ON INSERT Delta IN from[Order*Client] EXECUTE    -- (ECA rule 15)
     ALL of
            ONE of
                   INSERT INTO deliveredTo SELECTFROM (of;from[Order*Client] \/ of;De
                   (TO MAINTAIN -(of;from[Order*Client]) \/ deliveredTo FROM R6)
                   INSERT INTO I[Client] SELECTFROM (deliveredTo~;of;from[Order*Clien
                   (TO MAINTAIN -(of;from[Order*Client]) \/ deliveredTo FROM R6)
            (MAINTAINING -(of;from[Order*Client]) \/ deliveredTo FROM R6)
            ONE of
                   INSERT INTO sentTo SELECTFROM (delivery;of;from[Order*Client] \/ d
                   (TO MAINTAIN -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
                   INSERT INTO I[Client] SELECTFROM (sentTo~;delivery;of;from[Order*C
                   (TO MAINTAIN -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
            (MAINTAINING -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
```

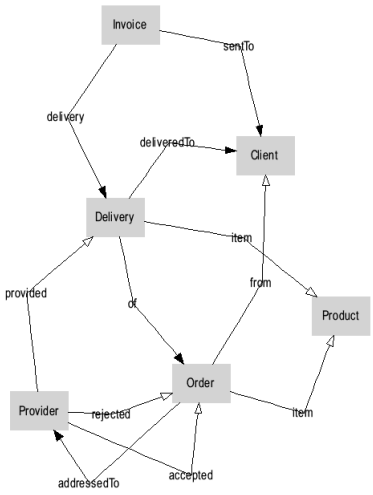

Figure 13.1: Language diagram of receive goods

```
     (MAINTAINING -(of;from[Order*Client]) \/ deliveredTo FROM R6)
     (MAINTAINING -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)

ON DELETE Delta FROM from[Order*Client] EXECUTE    -- (ECA rule 16)
     BLOCK
     (CANNOT CHANGE -(of;from[Order*Client]) \/ deliveredTo FROM R6)
     (CANNOT CHANGE -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)

ON INSERT Delta IN of EXECUTE    -- (ECA rule 17)
     ALL of
            ONE of
                   INSERT INTO deliveredTo SELECTFROM (of;from[Order*Client] \/ Delta
                   (TO MAINTAIN -(of;from[Order*Client]) \/ deliveredTo FROM R6)
                   INSERT INTO I[Client] SELECTFROM (deliveredTo~;of;from[Order*Clien
                   (TO MAINTAIN -(of;from[Order*Client]) \/ deliveredTo FROM R6)
            (MAINTAINING -(of;from[Order*Client]) \/ deliveredTo FROM R6)
            ONE of
                   INSERT INTO sentTo SELECTFROM (delivery;of;from[Order*Client] \/ d
                   (TO MAINTAIN -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
                   INSERT INTO I[Client] SELECTFROM (sentTo~;delivery;of;from[Order*C
                   (TO MAINTAIN -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
            (MAINTAINING -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
     (MAINTAINING -(of;from[Order*Client]) \/ deliveredTo FROM R6)
     (MAINTAINING -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)

ON DELETE Delta FROM of EXECUTE    -- (ECA rule 18)
     BLOCK
     (CANNOT CHANGE -(of;from[Order*Client]) \/ deliveredTo FROM R6)
     (CANNOT CHANGE -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
```

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. Rules ‘correct delivery’, ‘complete delivery’, ‘order based delivery’, ‘correct payments’, and ‘correct invoices’ will be maintained automatically, without intervention of a user. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	sentTo[Invoice*Client]	ECA rule 13
Del	sentTo[Invoice*Client]	no op
Ins	from[Order*Client]	ECA rule 15
Del	from[Order*Client]	error: rule ‘receive goods’ and ‘send invoices’
Ins	of[Delivery*Order]	ECA rule 17
Del	of[Delivery*Order]	error: rule ‘receive goods’ and ‘send invoices’
Ins	delivery[Invoice*Delivery]	ECA rule 19
Del	delivery[Invoice*Delivery]	error: rule ‘send invoices’

Figure 14.1 shows the knowledge graph of this service.

Figure 14.2 shows the switchboard diagram of this service.

ECA rules:
temporarily not documented

```
ON INSERT Delta IN sentTo EXECUTE    -- (ECA rule 13)
     ONE of
            INSERT INTO paid SELECTFROM (sentTo~ \/ Delta~)/\(sentTo~ \/ -paid)/\(-pa
            (TO MAINTAIN -sentTo~ \/ paid FROM R5)
            INSERT INTO I[Client] SELECTFROM (sentTo~;paid~ \/ Delta~;paid~)/\(sentTo
            (TO MAINTAIN -sentTo~ \/ paid FROM R5)
     (MAINTAINING -sentTo~ \/ paid FROM R5)

ON INSERT Delta IN from[Order*Client] EXECUTE    -- (ECA rule 15)
     ALL of
```

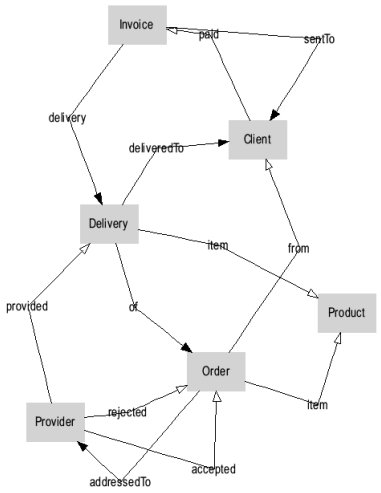

Figure 14.1: Language diagram of send invoices

```
            ONE of
                   INSERT INTO deliveredTo SELECTFROM (of;from[Order*Client] \/ of;De
                   (TO MAINTAIN -(of;from[Order*Client]) \/ deliveredTo FROM R6)
                   INSERT INTO I[Client] SELECTFROM (deliveredTo~;of;from[Order*Clien
                   (TO MAINTAIN -(of;from[Order*Client]) \/ deliveredTo FROM R6)
            (MAINTAINING -(of;from[Order*Client]) \/ deliveredTo FROM R6)
            ONE of
                   INSERT INTO sentTo SELECTFROM (delivery;of;from[Order*Client] \/ d
                   (TO MAINTAIN -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
                   INSERT INTO I[Client] SELECTFROM (sentTo~;delivery;of;from[Order*C
                   (TO MAINTAIN -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
            (MAINTAINING -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
     (MAINTAINING -(of;from[Order*Client]) \/ deliveredTo FROM R6)
     (MAINTAINING -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)

ON DELETE Delta FROM from[Order*Client] EXECUTE    -- (ECA rule 16)
     BLOCK
     (CANNOT CHANGE -(of;from[Order*Client]) \/ deliveredTo FROM R6)
     (CANNOT CHANGE -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)

ON INSERT Delta IN of EXECUTE    -- (ECA rule 17)
     ALL of
            ONE of
                   INSERT INTO deliveredTo SELECTFROM (of;from[Order*Client] \/ Delta
                   (TO MAINTAIN -(of;from[Order*Client]) \/ deliveredTo FROM R6)
                   INSERT INTO I[Client] SELECTFROM (deliveredTo~;of;from[Order*Clien
                   (TO MAINTAIN -(of;from[Order*Client]) \/ deliveredTo FROM R6)
            (MAINTAINING -(of;from[Order*Client]) \/ deliveredTo FROM R6)
            ONE of
```

```
                    INSERT INTO sentTo SELECTFROM (delivery;of;from[Order*Client] \/ d
                    (TO MAINTAIN -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
                    INSERT INTO I[Client] SELECTFROM (sentTo~;delivery;of;from[Order*C
                    (TO MAINTAIN -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
             (MAINTAINING -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
      (MAINTAINING -(of;from[Order*Client]) \/ deliveredTo FROM R6)
      (MAINTAINING -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)

ON DELETE Delta FROM of EXECUTE    -- (ECA rule 18)
      BLOCK
      (CANNOT CHANGE -(of;from[Order*Client]) \/ deliveredTo FROM R6)
      (CANNOT CHANGE -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)

ON INSERT Delta IN delivery EXECUTE    -- (ECA rule 19)
      ONE of
             INSERT INTO sentTo SELECTFROM (delivery;of;from[Order*Client] \/ Delta;of
             (TO MAINTAIN -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
             INSERT INTO I[Client] SELECTFROM (sentTo~;delivery;of;from[Order*Client]
             (TO MAINTAIN -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
      (MAINTAINING -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)

ON DELETE Delta FROM delivery EXECUTE    -- (ECA rule 20)
      BLOCK
      (CANNOT CHANGE -(delivery;of;from[Order*Client]) \/ sentTo FROM R7)
```

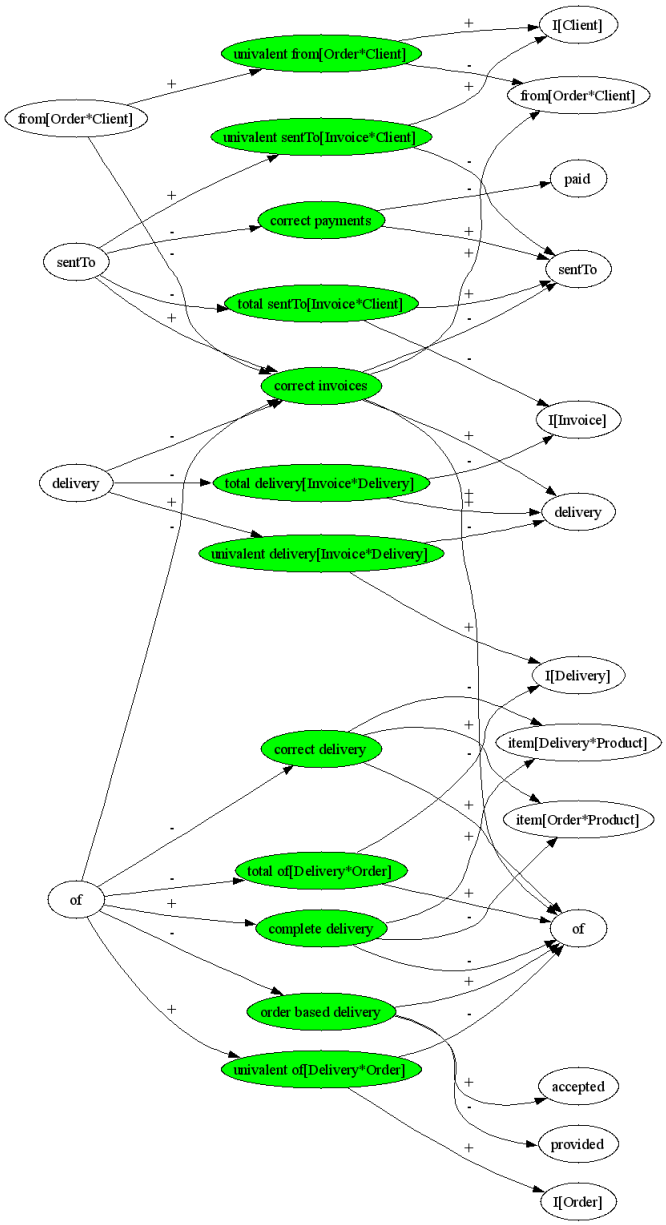

Figure 14.2: Switchboard of send invoices