Functional Specification of 'Webshop'

Put author(s) here
(This document was generated by Ampersand vs. 2.1.0.59)

Sat Apr 30 13:56:52 W. Europe Daylight Time 2011

Contents

Chapter 1

Introduction

This document defines the functionality of an information system called 'Webshop'. 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 'Webshop' as their intended audience. The data analysis in chapter 7 describes the data sets upon which 'Webshop' 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, 'Webshop' 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 'Webshop' can be discussed and expressed. The purpose of this chapter is to create shared understanding among stakeholders. The language of 'Webshop' 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 'Webshop', they share precisely enough language to have meaningful discussions about functional requirements. All definitions have been numbered for the sake of traceability.

2.1 Ordering

This pattern provides the language needed for clients to convey to a webshop what it is they wish to be delivered. Also, it provides guarantees to limit needless work.

This section introduces concepts Order, Client, Address, Product, and Processor.

In order to sell products, the webshop must allow clients to specify the products they wish to be delivered. The term 'order' refers to a set of products that a client has specified for delivery.

An *order* is a set of products that a clients wishes the webshop to deliver — *order*

The purpose of a webshop is to sell products. The term 'client' is used to designate parties to whom such products are being sold.

A *client* is an entity that can place orders, receive deliveries and pay invoices for ordered products — *client*

In order to send deliveries or invoices, the destination location must be known. We use the term 'address' for identifiers of such locations.

An *address* is an identifier of a location to which deliveries can be shipped and invoices can be sent — *address*

In order for a webshop to sell items, it should have items to sell. The term 'Product' refers to items that can be sold.

A *product* is an item that the webshop is capable of selling *product*

Processing orders consists of accepting or rejecting them, picking them so as to assemble a delivery, etcetra. The term 'Processor' refers to a person that performs such tasks.

A *processor* is an actor that is capable of handling orders *processor*

For shipping ordered products, sending a corresponding invoice etc., providers must know (details about) the client who placed that order. For the sake of simplicity, we have left out name, address, zipcode, etcetera.

Requirement 1: each order has been placed by zero or one client .

In order to improve the quality of order handling, a single order processor will be responsible for all activities pertaining to the handling of an order.

Requirement 2: each order is to be handled by exactly one processor .

Whether or not an order is to be carried out is a decision by the order processor. The consequence of rejecting an order is that the order will not be processed further and the client will not receive any products it may have ordered. An order processor may reject orders for a number of reasons, e.g. when the product is not carried, or when the amount payable would be so large that the risk of not receiving payment is too big.

Because of such consequences, it must be known whether or not an order is rejected. In order to put responsibility of handling an order into one hand, we need to know which order processor rejected the order.

Requirement 3: each order has been rejected by zero or one processor .

Whether or not an order is to be carried out is a decision by the order processor. The consequence of accepting an order will be that all products ordered by a client will be assembled into a delivery that is subsequently shipped to the client.

Because of such consequences, it must be known whether or not an order is accepted. In order to put responsibility of handling an order into one hand, we need to know which order processor accepted the order.

Requirement 4: each order has been accepted by zero or one processor .

Before it can be decided whether or not to accept an order, all information must be available (a) to make this decision and (b) to handle the order if it were accepted. An order that has the property 'cleanOrder' satisfies this condition.

Requirement 5: The sentence: "o satisfies the 'cleanOrder' conditions o' " is meaningful (i.e. it is either true or false) for any order o and order o'.

In order to send invoices (as well as other client-related communications), an address is needed to send such messages to.

Requirement 6: All invoices that each client should pay, must be sent to zero or one address.

Commitment to an order should only take place if it is known where the delivery is to be shipped to. Hence, a delivery address should be part of an order

Requirement 7: The delivery for each order must be shipped to zero or one address.

In order to decide whether or not to accept an order, all information must be available (a) to make this decision and (b) to handle the order if it were accepted. This is the case when:

- at least one product is ordered;
- the address to which the delivery is to be sent is known;
- the address to which the invoice is to be sent is known.

Requirement 8 (clean order): If there exist Orders called o, o', o satisfies the 'cleanOrder' conditions o' is equivalent to o equals o' and (If there exists a Product called p, o mentions p as an order item and True) and (If there exists a Address called a, The delivery for o must be shipped to a and True) and (If there exists a Client called c and there exists a Address called a, o has been placed by c and All invoices that c should pay, must be sent to a and True).

Orders can only be accepted if they are clean, and have not been rejected.

Requirement 9 (accepting orders): Orders that are accepted must have the property 'cleanOrder', and may not have been rejected.

Orders can only be rejected if they are clean, and have not been accepted.

Requirement 10 (rejecting orders): Orders that are rejected must have the property 'cleanOrder', and may not have been accepted.

The responsibility for handling an order must be assigned such that it is unambiguous. One way of doing this is to designate the order processor that decides acceptance or rejection of an order as the handler for that order.

Requirement 11 (assigning the order processor): The order processor that decides on acceptance or rejection of an order, is the designated handler for that order.

2.2 Order Picking

This pattern ensures that after an order has been accepted, a delivery (package) is created that consists of all ordered items (and no more than that), so that it can be sent to the customer.

This section introduces concept *Delivery*.

After an order has been accepted, the items on the order must be picked to form what is called a 'Delivery', which can subsequently be sent to the client that placed the order.

A *delivery* is a set of picked (physical) items that correspond to the items listed on a single order *delivery*

In order to be able to tell whether or not a delivery is ready for shipment, it is necessary to know of which products a delivery consists. Each delivery item corresponds to a line on the delivery form. For the sake of simplicity, we have left out quantity, product codes, etcetera.

Requirement 12: The sentence: "d contains p " is meaningful (i.e. it is either true or false) for any delivery d and product p.

It is necessary to distinguish deliveries that contain all items mentioned in the corresponding orders from those that do not. A delivery is said to have the property 'readyForShipping' if this is the case.

Requirement 13: *readyForShipping* is a property of deliveries.

In order to ensure that all handling of a single order is put in one hand, it is necessary to know who shipped a delivery.

In order to ensure that all deliveries are shipped to the client that ordered them, it is necessary to know to which client a delivery is shipped.

Requirement 14: each delivery has been shipped by zero or one processor .

Requirement 15 (ready for shipping): A delivery is ready for shipping iff each item in that delivery is mentioned on the corresponding order.

Deliveries should only be shipped if they are ready for shipping. Shipping of a delivery may only be done by the order handler that accepted the corresponding order, so as to ensure that responsibilities remain in one hand.

Requirement 16 (shipping): Deliveries may only be shipped (1) by the person that accepted the corresponding order and (2) if the delivery is ready for shipping.

2.3 Billing

In order to do business (make a profit) it is necessary to bill clients for the deliveries that are shipped to their orders. This pattern ensures that the correct bills are sent and payment is received.

This section introduces concept *Invoice*.

To enable a client to pay for the products in a delivery, the client must be informed about the costs thereof. We use the term 'Invoice' to refer to a message that spells out what the client is expected to pay for the products that the client ordered.

An *invoice* is a specification of the amount a client is expected to pay for the delivered products corresponding to a specific order *invoice*

In order to make sure that every delivery will ultimately be paid for, and also in order to convince a client to pay for a delivery, it is necessary to link a delivery with an invoice.

Requirement 17: each invoice covers exactly one delivery , but not for each delivery there must be a invoice.

In order to ensure that the responsibility for handling an order remains in one hand, it must be known which order processor has sent which invoices.

Requirement 18: each invoice has been sent by exactly one processor .

In order to receive payment for a delivery, the client that ordered this delivery must be informed about the cost. To this end, it must be possible to send invoices to a client's (invoice)address.

Requirement 19: each invoice was sent to exactly one client .

In order to receive payment for a delivery, the client that ordered this delivery must be informed about the cost. To this end, it must be possible to send invoices to a client's (invoice)address.

Requirement 20: each invoice was sent to exactly one client .

In order to establish that a client has paid for a delivery, it must be possible to know whether or not payment has been received for a particular invoice.

Requirement 21: There is at most one client (a) for each invoice (b), for which: Payment by b has been received for a.

In order to send invoices (as well as other client-related communications), an address is needed to send such messages to.

Requirement 22: All invoices that each client should pay, must be sent to zero or one address.

In order to send an invoice to a client, the invoice address must be known.

Requirement 23 (invoice address is available): If there exists a Invoice called i and there exists a Client called c, If $hasBeenSentTo(i)$ equals c, then i equals c and (If there exists a Address called a, All invoices that i should pay, must be sent to a and All invoices that c should pay, must be sent to a).

In order to make order processors accountable for the orders they process, we must ensure that the the order processor that handles an order also sends the corresponding invoice.

Requirement 24 (sending of invoices): If there exists a Invoice called i and there exists a Processor called p, If $hasBeenSentBy(i)$ equals p, then $correspondsTo(covers(i))$ has been accepted by p .

In order to prevent conflicts about erroneously sent invoices, it must be ensured that they are sent to the appropriate clients.

Requirement 25 (sending invoices to clients): An invoice that covers a delivery may only be sent to (the invoice address) of the client that has ordered the delivery.

To prevent conflicts about wrong payments (and to uphold the brand of the webshop), payments by clients shall only be accepted for invoices that correspond to orders placed by such clients.

Requirement 26 (correct payments): Payments by a client shall only be received if they are made for an invoice that has been sent to that client.

2.4 Delivery

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

This section introduces concept *Session*.

In order to provide ordering- and other services to clients, as well as provide order handling services to order processors, in such a way that they will only be provided services that are relevant to them, we need to distinguish the timeframes within which a specific actor interacts with the webshop. We use the term 'Session' to refer to such timeframes.

A *session* is a timeframe within which a specific actor interacts with the webshop *session*

In order to know which services may be executed in a session, it is necessary to know which, if any, order processor is logged in.

Requirement 27: each session is being run by zero or one processor .

In order to know which services may be executed in a session, it is necessary to know which, if any, client is logged in.

Requirement 28: each session is being run by zero or one client .

For shipping ordered products, sending a corresponding invoice etc., providers must know (details about) the client who placed that order. For the sake of simplicity, we have left out name, address, zipcode, etcetera.

Requirement 29: each order has been placed by zero or one client .

Whether or not an order is to be carried out is a decision by the order processor. The consequence of accepting an order will be that all products ordered by a client will be assembled into a delivery that is subsequently shipped to the client.

Because of such consequences, it must be known whether or not an order is accepted. In order to put responsibility of handling an order into one hand, we need to know which order processor accepted the order.

Requirement 30: each order has been accepted by zero or one processor .

In order to make sure that every delivery will ultimately be paid for, and also in order to convince a client to pay for a delivery, it is necessary to link a delivery with an invoice.

Requirement 31: each invoice covers exactly one delivery , but not for each delivery there must be a invoice.

In order to establish that a client has paid for a delivery, it must be possible to know whether or not payment has been received for a particular invoice.

Requirement 32: There is at most one client (a) for each invoice (b), for which: Payment by b has been received for a.

Requirement 33 (login): If there exists a Session called s, (If there exists a Processor 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 Processor called p, not(s is being run by p) and True).

Orders that are created should record all information necessary for the web-shop to decide whether or not to accept or reject it; i.e.: it should have the 'cleanOrder' property.

Requirement 34 (create orders): Each order that is created must have the 'cleanOrder' property.

Orders are issued by clients for the purpose of making a sales transaction. At some point in time, a order processor must accept or reject the order.

Requirement 35 (accept orders): For every (clean) order, it must be decided whether to accept or reject it.

For every orders that is accepted, a delivery should be assembled consisting of all ordered products. The order processor will be signalled of all accepted orders for which a corresponding delivery is not yet (correctly) assembled.

Requirement 36 (pick orders): If there exists a Order called o and there exists a Processor called p, If o has been accepted by p , then (If there exist Deliveries called d, d', o equals $correspondsTo(d)$ and d is ready for shipping, meaning that d' contains all items that are mentioned on the corresponding order and True).

When a delivery is ready for shipping, it must be sent to the client that has ordered it. The order processor will be signalled of deliveries that await shipping.

Requirement 37 (ship deliveries): Each delivery that is ready for shipping shall be sent to the client that placed the corresponding order.

When a delivery is ready for shipping, an invoice for it should be sent to the client that has ordered it. The order processor will be signalled of deliveries for which an invoice needs to be sent.

Requirement 38 (invoice deliveries): For each delivery that is ready for shipping, an invoice covering this delivery shall be sent to the client that placed the corresponding order.

A client to which an invoice has been sent, should pay this. The order processor will be signalled all invoices for which payment has not yet been received.

Requirement 39 (pay invoices): All invoices sent to a customer must be paid by that customer.

2.5 Loose ends...

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

In order to know which services may be executed in a session, it is necessary to know which, if any, order processor is logged in.

Requirement 40: each session is being run by zero or one processor .

In order to know which services may be executed in a session, it is necessary to know which, if any, client is logged in.

Requirement 41: each session is being run by zero or one client .

In order to improve the quality of order handling, a single order processor will be responsible for all activities pertaining to the handling of an order.

Requirement 42: each order is to be handled by exactly one processor .

In order to ensure that the responsibility for handling an order remains in one hand, it must be known which order processor has sent which invoices.

Requirement 43: each invoice has been sent by exactly one processor .

Chapter 3

Diagnosis

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

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

rule	Client	Processor
login	×	×
create orders	×	
accept orders		×
pick orders		×
ship deliveries		×
invoice deliveries		×
pay invoices	×	×

All concepts in this document have been provided with a definition.

Relations $orderedItem_{[Order \times Product]}$, $correspondsTo_{[Delivery \times Order]}$, $orderedItem$, and *hasBeenShippedTo* are explained by an automated explanation generator. If these explanations are not appropriate, add PURPOSE RELATION statements to your relations.

All relations in this document are being used in one or more rules.

On line numbers line 155, file "F:\\RJ$\\Prive\\CC model repository\\Adlfiles\\Webshop.adl" and line 268, file "F:\\RJ$\\Prive\\CC model repository\\Adlfiles\\Webshop.adl" of file F:\RJ$\Prive\CC model repository\Adlfiles\Webshop.adl, rules are defined without a proper explanation of their purpose.

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
$isPlacedBy : Order \times Client$	4
$deliveryAddress : Order \times Address$	4
$orderedItem : Order \times Product$	5
$cleanOrder : Order \times Order$	4
$acceptedBy : Order \times Processor$	3
$rejectedBy : Order \times Processor$	1
$orderProcessor : Order \times Processor$	4
$correspondsTo : Delivery \times Order$	3
$containsPickedItem : Delivery \times Product$	4
$readyForShipping : Delivery \times Delivery$	3
$hasBeenShippedBy : Delivery \times Processor$	3
$hasBeenShippedTo : Delivery \times Client$	3
$invoiceAddress : Client \times Address$	4
$hasBeenSentTo : Invoice \times Client$	3
$covers : Invoice \times Delivery$	3
$hasBeenSentBy : Invoice \times Processor$	3
$hasPaid : Client \times Invoice$	3
$I_{[Order]}$	4
$I_{[Client]}$	4
$I_{[Address]}$	4
$I_{[Product]}$	5
$I_{[Processor]}$	3
$I_{[Delivery]}$	3
$I_{[Invoice]}$	3
$I_{[Session]}$	0

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

Chapter 4

Conceptual Analysis

The purpose of a webshop is to try and sell products. To this end, it must have processes in place that allow clients to inform the webshop about the products they want (ordering), ship such products (delivery) and ensure that these are paid for (billing).

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 Ordering

This pattern provides the language needed for clients to convey to a webshop what it is they wish to be delivered. Also, it provides guarantees to limit needless work.

Figure 4.1 shows a conceptual diagram of this theme.

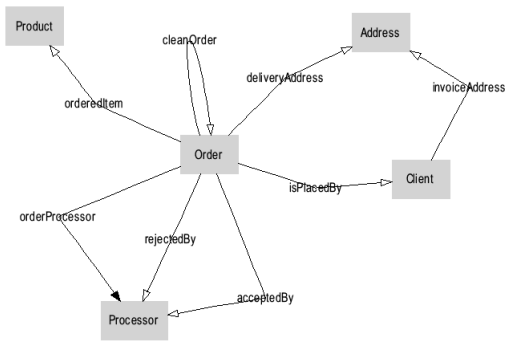


Figure 4.1: Conceptual model of Ordering

clean order In order to decide whether or not to accept an order, all information must be available (a) to make this decision and (b) to handle the order if it were accepted. This is the case when:

- at least one product is ordered;
- the address to which the delivery is to be sent is known;
- the address to which the invoice is to be sent is known.

Before it can be decided whether or not to accept an order, all information must be available (a) to make this decision and (b) to handle the order if it were accepted. An order that has the property 'cleanOrder' satisfies this condition.

In order to send invoices (as well as other client-related communications), an address is needed to send such messages to.

For shipping ordered products, sending a corresponding invoice etc., providers must know (details about) the client who placed that order. For the sake of simplicity, we have left out name, address, zipcode, etcetera.

Commitment to an order should only take place if it is known where the delivery is to be shipped to. Hence, a delivery address should be part of an order

To arrive at the formalization in equation 4.6, the following five relations are introduced.

$$cleanOrder \quad : \quad Order \times Order \tag{4.1}$$
$$invoiceAddress \quad : \quad Client \times Address \tag{4.2}$$
$$isPlacedBy \quad : \quad Order \times Client \tag{4.3}$$
$$deliveryAddress \quad : \quad Order \times Address \tag{4.4}$$
$$orderedItem \quad : \quad Order \times Product \tag{4.5}$$

$$\overline{cleanOrder} \cap I_{[Order]} \cap orderedItem; V_{[Product \times Order]} \cap deliveryAddress; V_{[Address \times Order]} \cap isPlacedBy; invoi \tag{4.6}$$

This means: If there exist Orders called o, o', o satisfies the 'cleanOrder' conditions o' is equivalent to o equals o' and (If there exists a Product called p, o mentions p as an order item and True) and (If there exists a Address called a, The delivery for o must be shipped to a and True) and (If there exists a Client called c and there exists a Address called a, o has been placed by c and All invoices that c should pay, must be sent to a and True). This corresponds to requirement 2.1 on page 5.

accepting orders Orders can only be accepted if they are clean, and have not been rejected.

Whether or not an order is to be carried out is a decision by the order processor. The consequence of accepting an order will be that all products ordered by a client will be assembled into a delivery that is subsequently shipped to the client.

Because of such consequences, it must be known whether or not an order is accepted. In order to put responsibility of handling an order into one hand, we need to know which order processor accepted the order.

Whether or not an order is to be carried out is a decision by the order processor. The consequence of rejecting an order is that the order will not be processed further and the client will not receive any products it may have ordered. An order processor may reject orders for a number of reasons, e.g. when the product is not carried, or when the amount payable would be so large that the risk of not receiving payment is too big.

Because of such consequences, it must be known whether or not an order is rejected. In order to put responsibility of handling an order into one hand, we need to know which order processor rejected the order.

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

$$acceptedBy \quad : \quad Order \times Processor \tag{4.7}$$

$$rejectedBy \quad : \quad Order \times Processor \tag{4.8}$$

Beside that, we use definition 4.1 (cleanOrder).

$$acceptedBy \cap \overline{(cleanOrder; (acceptedBy \cap \overline{rejectedBy}))} \tag{4.9}$$

This means: Orders that are accepted must have the property 'cleanOrder', and may not have been rejected.

This corresponds to requirement 2.1 on page 5.

rejecting orders Orders can only be rejected if they are clean, and have not been accepted.

We use definitions 4.7 (*acceptedBy*), 4.8 (*rejectedBy*), and 4.1 (*cleanOrder*).

$$rejectedBy \cap \overline{(cleanOrder; (rejectedBy \cap \overline{acceptedBy}))} \tag{4.10}$$

This means: Orders that are rejected must have the property 'cleanOrder', and may not have been accepted.

assigning the order processor The responsibility for handling an order must be assigned such that it is unambiguous. One way of doing this is to designate the order processor that decides acceptance or rejection of an order as the handler for that order.

In order to improve the quality of order handling, a single order processor will be responsible for all activities pertaining to the handling of an order.

In order to formalize this, a function orderProcessor is introduced (4.11):

$$orderProcessor \quad : \quad Order \rightarrow Processor \tag{4.11}$$

We also use definitions 4.7 (*acceptedBy*) and 4.8 (*rejectedBy*) to formalize requirement 2.1 (page 5):

$$acceptedBy \cap \overline{orderProcessor} \cup rejectedBy \cap \overline{orderProcessor} \tag{4.12}$$

This means: The order processor that decides on acceptance or rejection of an order, is the designated handler for that order.

4.2 Order Picking

This pattern ensures that after an order has been accepted, a delivery (package) is created that consists of all ordered items (and no more than that), so that it can be sent to the customer.

Figure 4.2 shows a conceptual diagram of this theme.

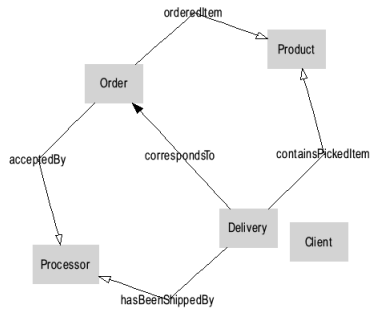


Figure 4.2: Conceptual model of Order Picking

ready for shipping It is necessary to distinguish deliveries that contain all items mentioned in the corresponding orders from those that do not. A delivery is said to have the property 'readyForShipping' if this is the case.

In order to be able to tell whether or not a delivery is ready for shipment, it is necessary to know of which products a delivery consists. Each delivery item corresponds to a line on the delivery form. For the sake of simplicity, we have left out quantity, product codes, etcetera.

To arrive at the formalization in equation 4.16, the following three relations are introduced.

$$readyForShipping \quad : \quad Delivery \times Delivery \tag{4.13}$$

$$correspondsTo \quad : \quad Delivery \rightarrow Order \tag{4.14}$$

$$containsPickedItem \quad : \quad Delivery \times Product \tag{4.15}$$

Beside that, we use definition 4.5 (orderedItem).

$$\overline{readyForShipping} \cap containsPickedItem \dagger \overline{(orderedItem \ ; correspondsTo \ )} \cap \overline{containsPickedItem \dagger orderedIte} \tag{4.16}$$

This means: A delivery is ready for shipping iff each item in that delivery is mentioned on the corresponding order.

This corresponds to requirement 2.2 on page 6.

shipping Deliveries should only be shipped if they are ready for shipping. Shipping of a delivery may only be done by the order handler that accepted the corresponding order, so as to ensure that responsibilities remain in one hand.

In order to ensure that all handling of a single order is put in one hand, it is necessary to know who shipped a delivery.

In order to ensure that all deliveries are shipped to the client that ordered them, it is necessary to know to which client a delivery is shipped.

In order to formalize this, a relation hasBeenShippedBy is introduced (4.17):

$$hasBeenShippedBy \quad : \quad Delivery \times Processor \tag{4.17}$$

We also use definitions 4.13 (*readyForShipping*), 4.14 (*correspondsTo*), and 4.7 (*acceptedBy*) to formalize requirement 2.2 (page 6):

$$hasBeenShippedBy \cap \overline{(readyForShipping; correspondsTo; acceptedBy)} \tag{4.18}$$

This means: Deliveries may only be shipped (1) by the person that accepted the corresponding order and (2) if the delivery is ready for shipping.

4.3 Billing

In order to do business (make a profit) it is necessary to bill clients for the deliveries that are shipped to their orders. This pattern ensures that the correct bills are sent and payment is received.

Figure 4.3 shows a conceptual diagram of this theme.

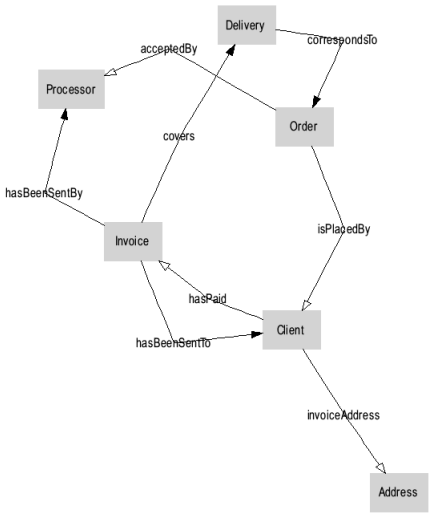


Figure 4.3: Conceptual model of Billing

invoice address is available In order to send an invoice to a client, the invoice address must be known.

In order to receive payment for a delivery, the client that ordered this delivery must be informed about the cost. To this end, it must be possible to send invoices to a client's (invoice)address.

In order to formalize this, a function hasBeenSentTo is introduced (4.19):

$$hasBeenSentTo \quad : \quad Invoice \rightarrow Client \tag{4.19}$$

Beside that, we use definition 4.2 (invoiceAddress) to formalize requirement 2.3 (page 8):

$$hasBeenSentTo \cap \overline{(hasBeenSentTo; (I_{[Client]} \cap invoiceAddress; invoiceAddress\ ))} \tag{4.20}$$

This means: If there exists a Invoice called i and there exists a Client called c, If $hasBeenSentTo(i)$ equals c, then i equals c and (If there exists a Address called a, All invoices that i should pay, must be sent to a and All invoices that c should pay, must be sent to a).

sending of invoices In order to make order processors accountable for the orders they process, we must ensure that the the order processor that handles an order also sends the corresponding invoice.

In order to ensure that the responsibility for handling an order remains in one hand, it must be known which order processor has sent which invoices.

In order to make sure that every delivery will ultimately be paid for, and also in order to convince a client to pay for a delivery, it is necessary to link a delivery with an invoice.

To arrive at the formalization in equation 4.23, the following two relations are introduced.

$$hasBeenSentBy \quad : \quad Invoice \rightarrow Processor \tag{4.21}$$

$$covers \quad : \quad Invoice \rightarrow Delivery \tag{4.22}$$

We also use definitions 4.14 ($correspondsTo$) and 4.7 ($acceptedBy$).

$$hasBeenSentBy \cap \overline{(covers; correspondsTo; acceptedBy)} \tag{4.23}$$

This means: If there exists a Invoice called i and there exists a Processor called p, If $hasBeenSentBy(i)$ equals p, then $correspondsTo(covers(i))$ has been accepted by p . This corresponds to requirement 2.3 on page 8.

sending invoices to clients In order to prevent conflicts about erroneously sent invoices, it must be ensured that they are sent to the appropriate clients.

We use definitions 4.22 ($covers$), 4.19 ($hasBeenSentTo$), 4.14 ($correspondsTo$), and 4.3 ($isPlacedBy$).

$$hasBeenSentTo \cap \overline{(covers; correspondsTo; isPlacedBy)} \tag{4.24}$$

This means: An invoice that covers a delivery may only be sent to (the invoice address) of the client that has ordered the delivery.

correct payments To prevent conflicts about wrong payments (and to uphold the brand of the webshop), payments by clients shall only be accepted for invoices that correspond to orders placed by such clients.

In order to establish that a client has paid for a delivery, it must be possible to know whether or not payment has been received for a particular invoice.

In order to formalize this, a relation hasPaid is introduced (4.25):

$$hasPaid \quad : \quad Client \times Invoice \tag{4.25}$$

Beside that, we use definition 4.19 (hasBeenSentTo) to formalize requirement 2.3 (page 8):

$$hasPaid \cap \overline{hasBeenSentTo} \tag{4.26}$$

This means: Payments by a client shall only be received if they are made for an invoice that has been sent to that client.

4.4 Sessions

Figure 4.4 shows a conceptual diagram of this theme.

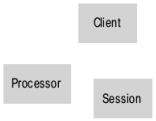


Figure 4.4: Conceptual model of Sessions

Chapter 5

Process Analysis

The purpose of a webshop is to try and sell products. To this end, it must have processes in place that allow clients to inform the webshop about the products they want (ordering), ship such products (delivery) and ensure that these are paid for (billing).

Webshop 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
Processor	login accept orders pick orders ship deliveries invoice deliveries pay invoices

Webshop 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	$sClient_{[Session \times Client]}$ $isPlacedBy_{[Order \times Client]}$ $orderedItem_{[Order \times Product]}$ $deliveryAddress$ $invoiceAddress$ $hasPaid$
Processor	$sProvider_{[Session \times Processor]}$ $acceptedBy$ $rejectedBy$ $correspondsTo$ $containsPickedItem_{[Delivery \times Product]}$ $hasBeenShippedBy$ $hasBeenShippedTo$ $hasBeenSentBy$ $hasBeenSentTo$
	$cleanOrder : Order \times Order$ $orderProcessor : Order \times Processor$ $readyForShipping : Delivery \times Delivery$ $covers : Invoice \times Delivery$

5.1 Delivery

Figure 5.1 shows the process model.

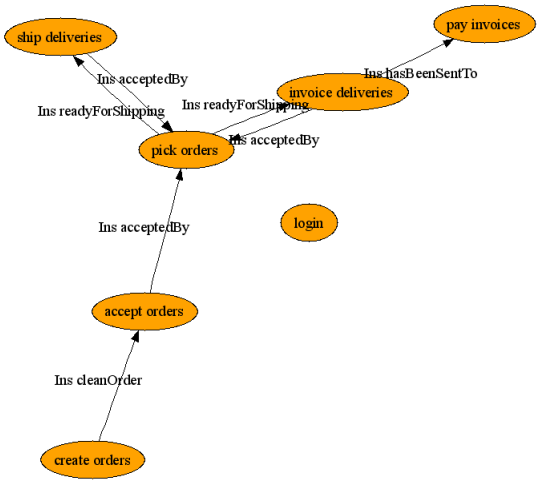


Figure 5.1: Process model of Delivery

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

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

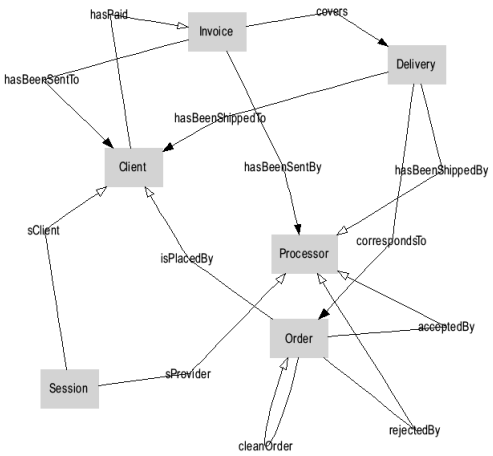


Figure 5.2: Basic sentences of Delivery

by:

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

This means: If there exists a Session called s, (If there exists a Processor 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 Processor called p, not(s is being run by p) and True).

create orders Orders that are created should record all information necessary for the webshop to decide whether or not to accept or reject it; i.e.: it should have the 'cleanOrder' property.

We use definition 4.1 (cleanOrder). Activity is signalled by:

$$I_{[Order]} \cap \overline{cleanOrder} \tag{5.2}$$

This means: Each order that is created must have the 'cleanOrder' property.

accept orders Orders are issued by clients for the purpose of making a sales transaction. At some point in time, a order processor must accept or reject the order.

We use definitions 4.1 (*cleanOrder*), 4.7 (*acceptedBy*), and 4.8 (*rejectedBy*). Activity is signalled by:

$$cleanOrder \cap \overline{(acceptedBy; V_{[Processor \times Order]})} \cap \overline{(rejectedBy; V_{[Processor \times Order]})} \tag{5.3}$$

This means: For every (clean) order, it must be decided whether to accept or reject it.

pick orders For every orders that is accepted, a delivery should be assembled consisting of all ordered products. The order processor will be signalled of all accepted orders for which a corresponding delivery is not yet (correctly) assembled.

We use definitions 4.7 (*acceptedBy*), 4.14 (*correspondsTo*), and 4.13 (*readyForShipping*). Activity is signalled by:

$$acceptedBy \cap \overline{(correspondsTo \ ; readyForShipping; V_{[Delivery \times Processor]})} \tag{5.4}$$

This means: If there exists a Order called o and there exists a Processor called p, If o has been accepted by p , then (If there exist Deliveries called d, d', o equals $correspondsTo(d)$ and d is ready for shipping, meaning that d' contains all items that are mentioned on the corresponding order and True).

ship deliveries When a delivery is ready for shipping, it must be sent to the client that has ordered it. The order processor will be signalled of deliveries that await shipping.

We use definitions 4.3 (*isPlacedBy*), 4.7 (*acceptedBy*), 4.14 (*correspondsTo*), 4.13 (*readyForShipping*), 4.17 (*hasBeenShippedBy*), and **??** (*hasBeenShippedTo*). Activity is signalled by:

$$readyForShipping \cap \overline{((hasBeenShippedBy; acceptedBy \cap hasBeenShippedTo; isPlacedBy \ ); correspondsTo \ )} \tag{5.5}$$

This means: Each delivery that is ready for shipping shall be sent to the client that placed the corresponding order.

invoice deliveries When a delivery is ready for shipping, an invoice for it should be sent to the client that has ordered it. The order processor will be signalled of deliveries for which an invoice needs to be sent.

We use definitions 4.3 (*isPlacedBy*), 4.7 (*acceptedBy*), 4.14 (*correspondsTo*), 4.13 (*readyForShipping*), 4.19 (*hasBeenSentTo*), 4.22 (*covers*), and 4.21 (*hasBeenSentBy*). Activity is signalled by:

$$readyForShipping \cap \overline{(covers \ ; (hasBeenSentBy; acceptedBy \cap hasBeenSentTo; isPlacedBy \ ); correspondsTo} \tag{5.6}$$

This means: For each delivery that is ready for shipping, an invoice covering this delivery shall be sent to the client that placed the corresponding order.

pay invoices A client to which an invoice has been sent, should pay this. The order processor will be signalled all invoices for which payment has not yet been received.

We use definitions 4.19 (*hasBeenSentTo*) and 4.25 (*hasPaid*). Activity is signalled by:

$$hasBeenSentTo \cap \overline{hasPaid} \tag{5.7}$$

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

Chapter 6

Function Point Analysis

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

data set	analysis	FP
Delivery	ILGV Eenvoudig	7
Order	ILGV Eenvoudig	7
Session	ILGV Eenvoudig	7
Client	ILGV Eenvoudig	7
Processor	ILGV Eenvoudig	7
Product	ILGV Eenvoudig	7
Address	ILGV Eenvoudig	7

interface	analysis	FP
login	NO	0
create orders	NO	0
accept orders	NO	0
pick orders	NO	0
ship deliveries	NO	0
invoice deliveries	NO	0
pay 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, three associations, no generalisations, and no aggregations. Webshop has a total of 8 concepts.

Figure 7.1: Classification of Webshop

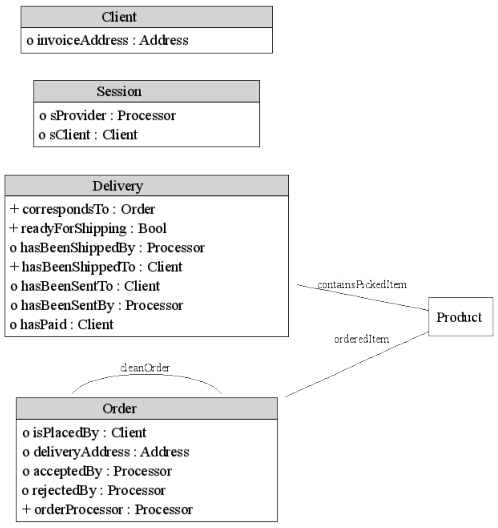


Figure 7.2: Data model of Webshop

Additionally, the endorelations come with the following properties :

relation	Rfx	Irf	Trn	Sym	Asy	Prop
cleanOrder						
readyForShipping				√	√	√

7.1 Delivery

The attributes in Delivery have the following multiplicity constraints.

attribute	type	mandatory	unique
key	Delivery	√	√
key	Invoice		√
correspondsTo	Order	√	
readyForShipping	Bool	√	
hasBeenShippedBy	Processor		
hasBeenShippedTo	Client	√	
hasBeenSentTo	Client		
hasBeenSentBy	Processor		
hasPaid	Client		

This data set generates one process rule.

$$hasBeenSentTo \vdash hasPaid$$

7.2 Order

The attributes in Order have the following multiplicity constraints.

attribute	type	mandatory	unique
key	Order	√	√
isPlacedBy	Client		
deliveryAddress	Address		
acceptedBy	Processor		
rejectedBy	Processor		
orderProcessor	Processor	√	

Within this data set, the following integrity rule shall be true at all times.

$$acceptedBy \cup rejectedBy \vdash orderProcessor$$

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

$$\overline{I_{[Order]}} \cup cleanOrder$$

7.3 Session

The attributes in Session have the following multiplicity constraints.

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

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

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

7.4 Client

The attributes in Client have the following multiplicity constraints.

attribute	type	mandatory	unique
key	Client	√	√
invoiceAddress	Address		

7.5 orderedItem

The attributes in orderedItem have the following multiplicity constraints.

attribute	type	mandatory	unique
key	Order	√	
Product	Product	√	

7.6 cleanOrder

The attributes in cleanOrder have the following multiplicity constraints.

attribute	type	mandatory	unique
key	Order	√	
tOrder	Order	√	

This data set generates one process rule.

$$I_{[Order]} \vdash cleanOrder$$

7.7 containsPickedItem

The attributes in containsPickedItem 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 Processor. 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*Processor]	error: rule 'login'
Del	sProvider[Session*Processor]	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 interface.

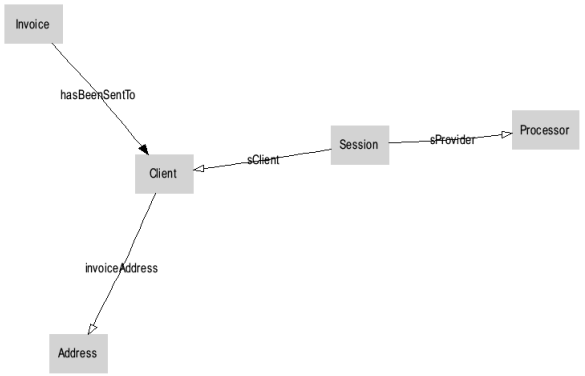


Figure 8.1: Language diagram of login

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.

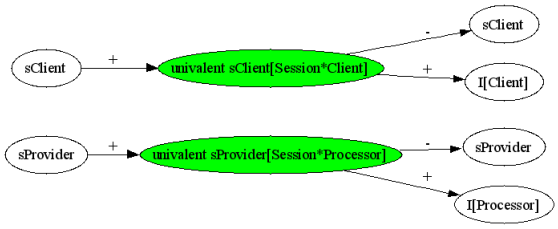


Figure 8.2: Switchboard of login

Chapter 9

create orders

Orders that are created should record all information necessary for the webshop to decide whether or not to accept or reject it; i.e.: it should have the 'cleanOrder' property.

Activity create orders must be performed by a user with role Client. Rules 'clean order', 'accepting orders', and 'rejecting orders' 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	cleanOrder[Order]	ECA rule 9
Del	cleanOrder[Order]	error: rule 'accept orders'

Figure 9.1 shows the knowledge graph of this interface.

Figure 9.2 shows the switchboard diagram of this interface.

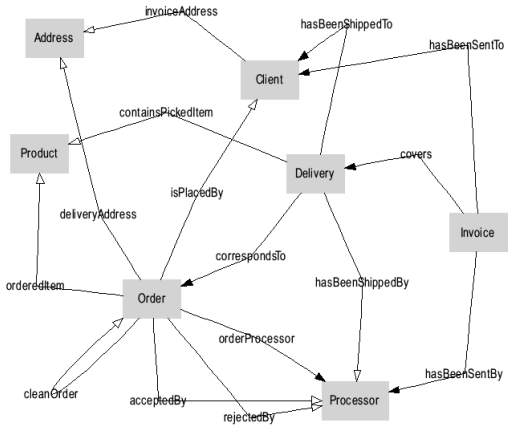


Figure 9.1: Language diagram of create orders

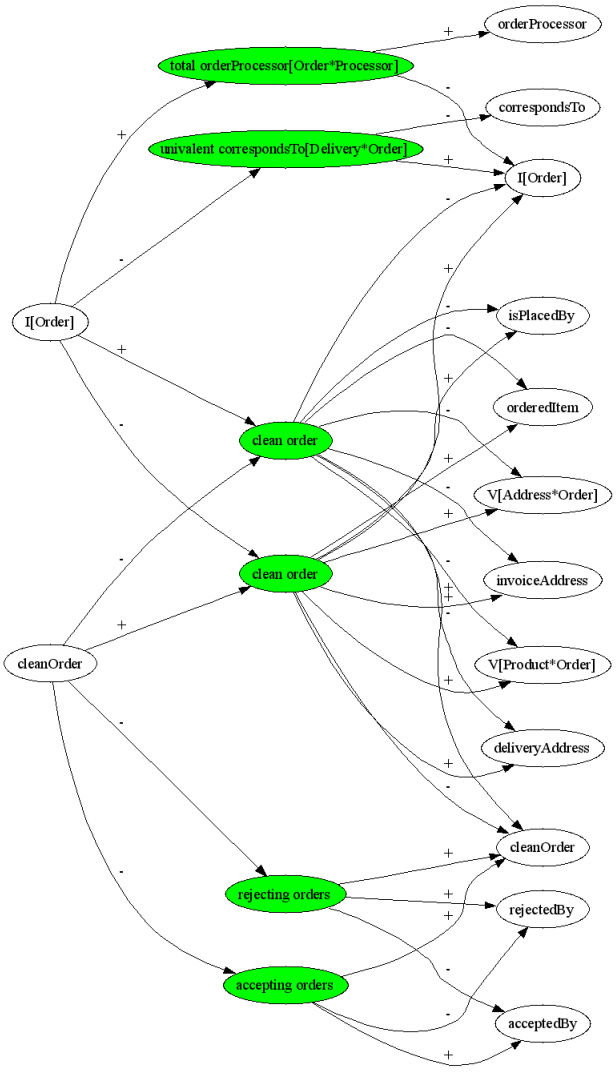


Figure 9.2: Switchboard of create orders

Chapter 10

accept orders

Orders are issued by clients for the purpose of making a sales transaction. At some point in time, a order processor must accept or reject the order.

Activity accept orders must be performed by a user with role Processor. Rules ‘clean order’, ‘accepting orders’, ‘rejecting orders’, ‘assigning the order processor’, ‘shipping’, and ‘sending of invoices’ will be maintained automatically, without intervention of a user. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	cleanOrder[Order]	ECA rule 9
Del	cleanOrder[Order]	error: rule ‘accept orders’
Ins	acceptedBy[Order*Processor]	error: rule ‘ship deliveries’ and ‘invoice deliveries’
Del	acceptedBy[Order*Processor]	error: rule ‘pick orders’

Figure 10.1 shows the knowledge graph of this interface.

Figure 10.2 shows the switchboard diagram of this interface.

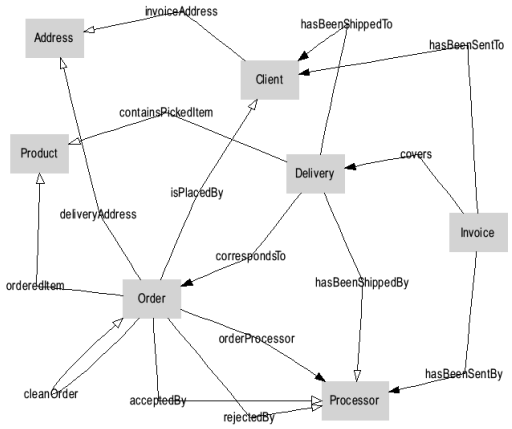


Figure 10.1: Language diagram of accept orders

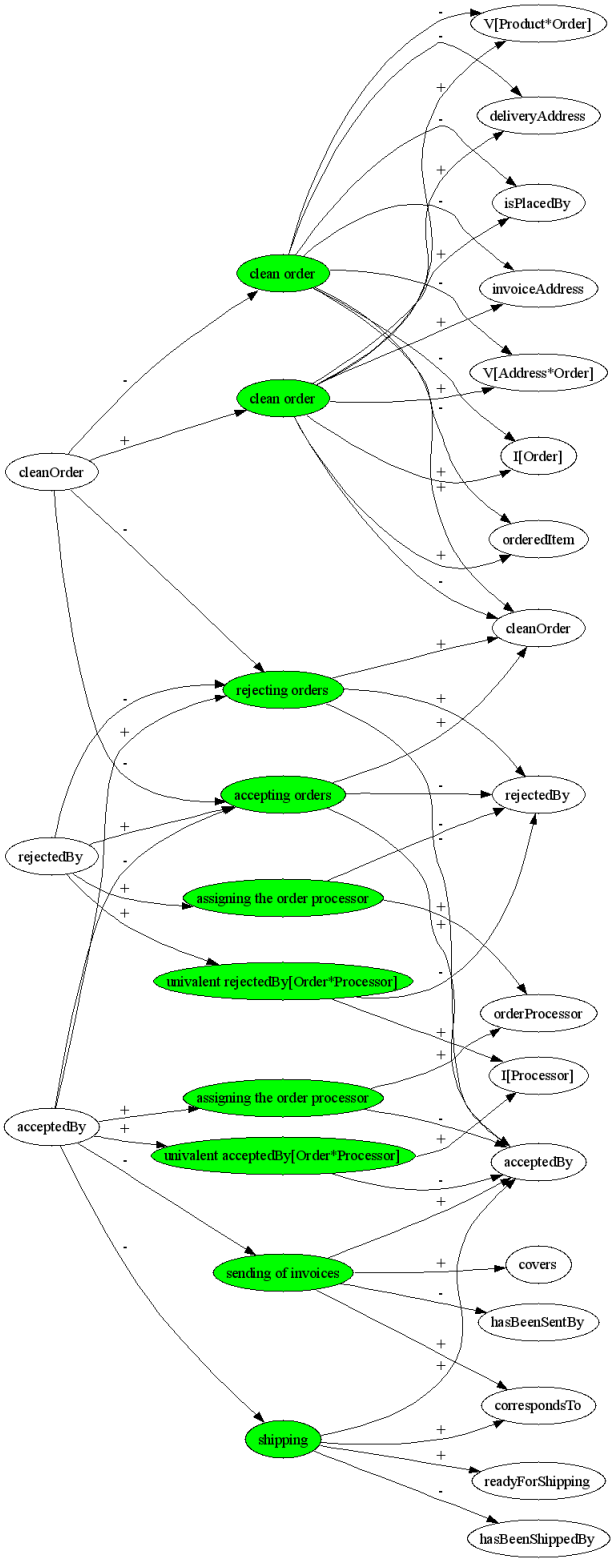


Figure 10.2: Switchboard of accept orders

Chapter 11

pick orders

For every orders that is accepted, a delivery should be assembled consisting of all ordered products. The order processor will be signalled of all accepted orders for which a corresponding delivery is not yet (correctly) assembled.

Activity pick orders must be performed by a user with role Processor. Rules ‘accepting orders’, ‘rejecting orders’, ‘assigning the order processor’, ‘ready for shipping’, ‘shipping’, ‘sending of invoices’, and ‘sending invoices to clients’ will be maintained automatically, without intervention of a user. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	acceptedBy[Order*Processor]	error: rule ‘ship deliveries’ and ‘invoice deliveries’
Del	acceptedBy[Order*Processor]	error: rule ‘pick orders’
Ins	readyForShipping[Delivery]	ECA rule 13
Del	readyForShipping[Delivery]	error: rule ‘ship deliveries’ and ‘invoice deliveries’

Figure 11.1 shows the knowledge graph of this interface.

Figure 11.2 shows the switchboard diagram of this interface.

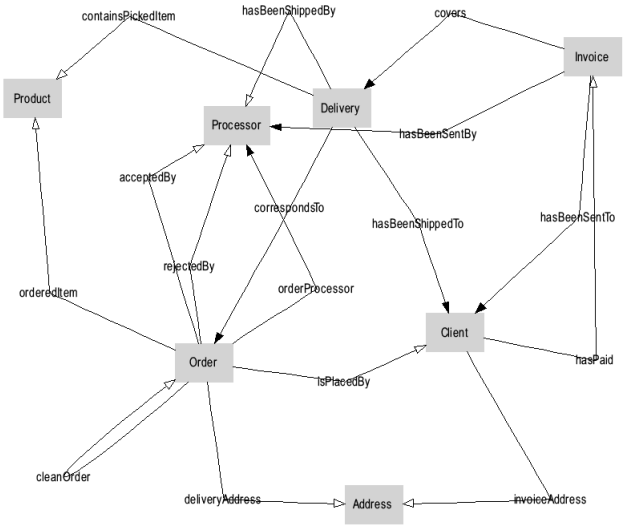


Figure 11.1: Language diagram of pick orders

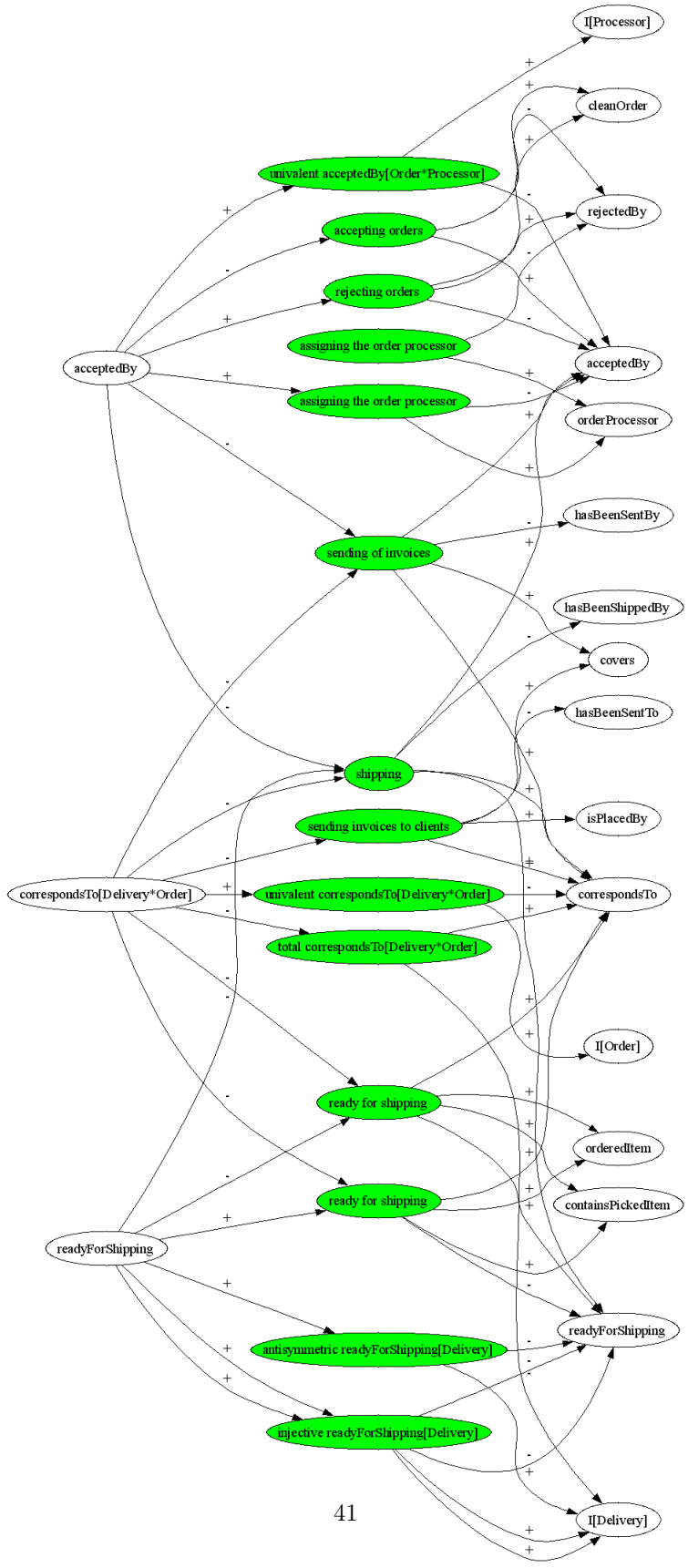


Figure 11.2: Switchboard of pick orders

Chapter 12

ship deliveries

When a delivery is ready for shipping, it must be sent to the client that has ordered it. The order processor will be signalled of deliveries that await shipping.

Activity ship deliveries must be performed by a user with role Processor. Rules 'clean order', 'accepting orders', 'rejecting orders', 'assigning the order processor', 'ready for shipping', 'shipping', 'sending of invoices', and 'sending invoices to clients' will be maintained automatically, without intervention of a user. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	acceptedBy[Order*Processor]	error: rule 'ship deliveries' and 'invoice deliveries'
Del	acceptedBy[Order*Processor]	error: rule 'pick orders'
Ins	readyForShipping[Delivery]	ECA rule 13
Del	readyForShipping[Delivery]	error: rule 'ship deliveries' and 'invoice deliveries'
Ins	isPlacedBy[Order*Client]	error: rule 'ship deliveries' and 'invoice deliveries'
Del	isPlacedBy[Order*Client]	ECA rule 16
Ins	hasBeenShippedTo[Delivery*Client]	error: rule 'ship deliveries'
Del	hasBeenShippedTo[Delivery*Client]	ECA rule 18
Ins	hasBeenShippedBy[Delivery*Processor]	error: rule 'ship deliveries'
Del	hasBeenShippedBy[Delivery*Processor]	ECA rule 20

Figure 12.1 shows the knowledge graph of this interface.

Figure 12.2 shows the switchboard diagram of this interface.

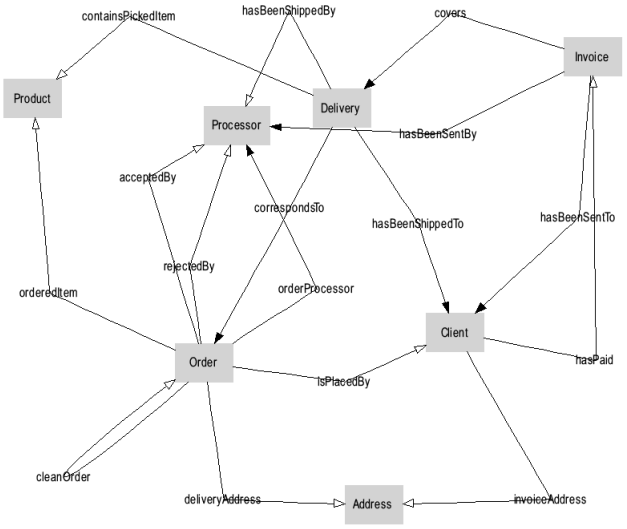
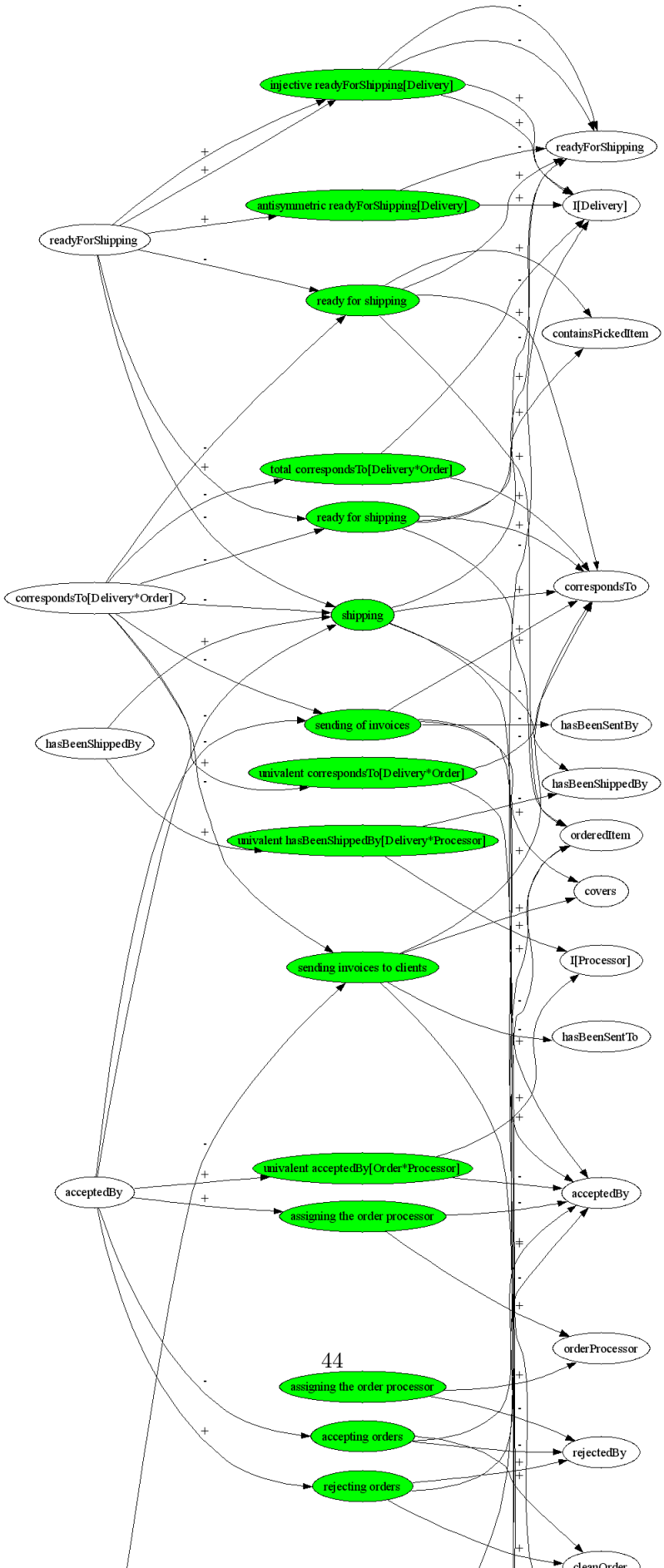


Figure 12.1: Language diagram of ship deliveries


injective readyForShipping[Delivery]
readyForShipping
antisymmetric readyForShipping[Delivery]
I[Delivery]
readyForShipping
ready for shipping
containsPickedItem
total correspondsTo[Delivery*Order]
ready for shipping
correspondsTo
correspondsTo[Delivery*Order]
shipping
sending of invoices
hasBeenSentBy
hasBeenShippedBy
univalent correspondsTo[Delivery*Order]
hasBeenShippedBy
univalent hasBeenShippedBy[Delivery*Processor]
orderedItem
covers
sending invoices to clients
I[Processor]
hasBeenSentTo
univalent acceptedBy[Order*Processor]
acceptedBy
acceptedBy
assigning the order processor
orderProcessor
assigning the order processor
accepting orders
rejectedBy
rejecting orders

Chapter 13

invoice deliveries

When a delivery is ready for shipping, an invoice for it should be sent to the client that has ordered it. The order processor will be signalled of deliveries for which an invoice needs to be sent.

Activity invoice deliveries must be performed by a user with role Processor. Rules ‘clean order’, ‘accepting orders’, ‘rejecting orders’, ‘assigning the order processor’, ‘ready for shipping’, ‘shipping’, ‘invoice address is available’, ‘sending of invoices’, ‘sending invoices to clients’, and ‘correct payments’ will be maintained automatically, without intervention of a user. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	acceptedBy[Order*Processor]	error: rule ‘ship deliveries’ and ‘invoice deliveries’
Del	acceptedBy[Order*Processor]	error: rule ‘pick orders’
Ins	readyForShipping[Delivery]	ECA rule 13
Del	readyForShipping[Delivery]	error: rule ‘ship deliveries’ and ‘invoice deliveries’
Ins	isPlacedBy[Order*Client]	error: rule ‘ship deliveries’ and ‘invoice deliveries’
Del	isPlacedBy[Order*Client]	ECA rule 16
Ins	hasBeenSentTo[Invoice*Client]	error: rule ‘invoice deliveries’
Del	hasBeenSentTo[Invoice*Client]	ECA rule 22
Ins	hasBeenSentBy[Invoice*Processor]	error: rule ‘invoice deliveries’
Del	hasBeenSentBy[Invoice*Processor]	ECA rule 24

Figure 13.1 shows the knowledge graph of this interface.

Figure 13.2 shows the switchboard diagram of this interface.

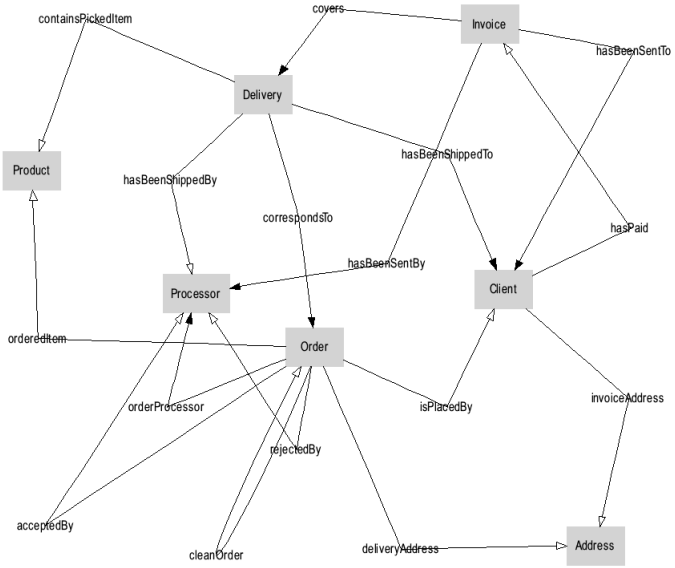
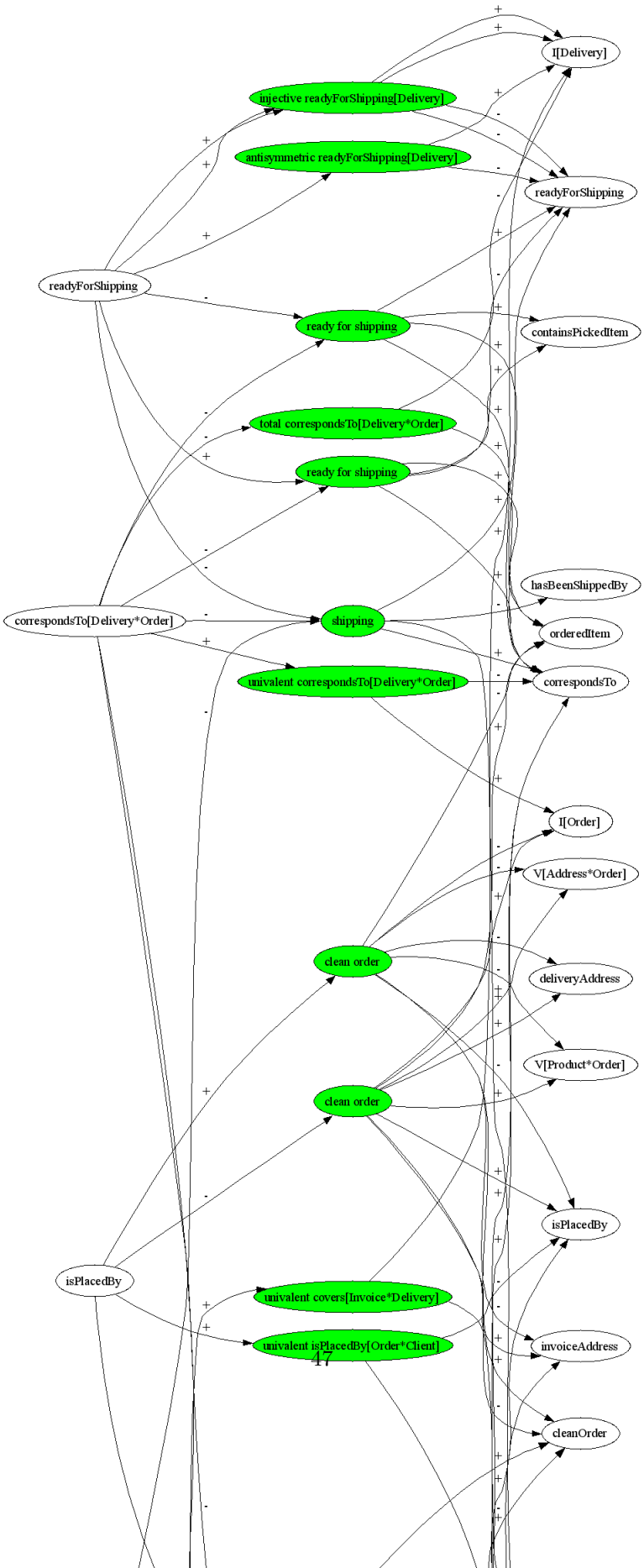


Figure 13.1: Language diagram of invoice deliveries


I[Delivery]
injective readyForShipping[Delivery]
antisymmetric readyForShipping[Delivery]
readyForShipping
readyForShipping
ready for shipping
containsPickedItem
total correspondsTo[Delivery*Order]
ready for shipping
hasBeenShippedBy
correspondsTo[Delivery*Order]
shipping
orderedItem
univalent correspondsTo[Delivery*Order]
correspondsTo
I[Order]
V[Address*Order]
clean order
deliveryAddress
V[Product*Order]
clean order
isPlacedBy
isPlacedBy
univalent covers[Invoice*Delivery]
univalent isPlacedBy[Order*Client]
invoiceAddress
cleanOrder

Chapter 14

pay invoices

A client to which an invoice has been sent, should pay this. The order processor will be signalled all invoices for which payment has not yet been received.

Activity pay invoices must be performed by a user with role Processor or Client. Rules 'invoice address is available', 'sending invoices to clients', and 'correct payments' will be maintained automatically, without intervention of a user. The following table shows which edit actions invoke which function.

action	relation	rule
Ins	hasBeenSentTo[Invoice*Client]	error: rule 'invoice deliveries'
Del	hasBeenSentTo[Invoice*Client]	ECA rule 22

Figure 14.1 shows the knowledge graph of this interface.

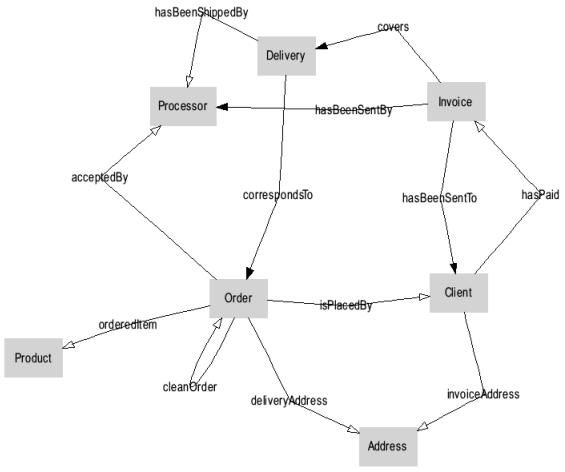


Figure 14.1: Language diagram of pay invoices

Figure 14.2 shows the switchboard diagram of this interface.

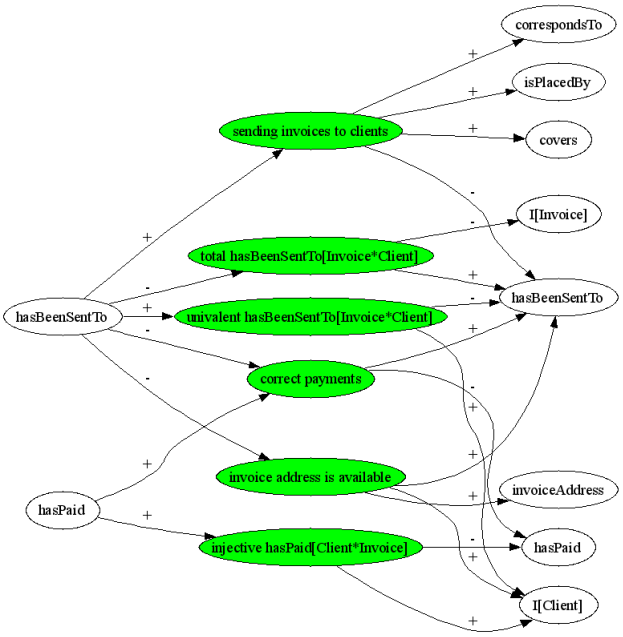


Figure 14.2: Switchboard of pay invoices

Chapter 15

ECA rules

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

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

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

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

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

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

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

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

```
ON INSERT Delta IN I[Order] EXECUTE    -- (ECA rule 7)
INSERT INTO cleanOrder SELECTFROM
  (I[Order] \/ Delta)/\(I[Order] \/ -cleanOrder)/\(-cleanOrder \/ Delta)/\(-cleanOrde
(TO MAINTAIN -I[Order] \/ cleanOrder FROM R2)

ON DELETE Delta FROM I[Order] EXECUTE    -- (ECA rule 8)
BLOCK
(CANNOT CHANGE -I[Order] \/ cleanOrder FROM R2)

ON INSERT Delta IN cleanOrder EXECUTE    -- (ECA rule 9)
ONE of CREATE x:Processor;
         ALL of INSERT INTO V[Processor*Order] SELECTFROM
                  V[Processor*Order];((cleanOrder \/ Delta)/\-(acceptedBy;V[Processor
                INSERT INTO acceptedBy SELECTFROM
                  ((cleanOrder \/ Delta)/\-(acceptedBy;V[Processor*Order])/\-(rejecte
         (MAINTAINING -cleanOrder \/ acceptedBy;V[Processor*Order] \/ rejectedBy;V[Pr
       (MAINTAINING -cleanOrder \/ acceptedBy;V[Processor*Order] \/ rejectedBy;V[Proc
       SELECT x:Processor FROM codomain(acceptedBy);
         INSERT INTO V[Processor*Order] SELECTFROM
           V[Processor*Order];((cleanOrder \/ Delta)/\-(acceptedBy;V[Processor*Order]
       (MAINTAINING -cleanOrder \/ acceptedBy;V[Processor*Order] \/ rejectedBy;V[Proc
       SELECT x:Processor FROM codomain(V[Order*Processor]);
         INSERT INTO acceptedBy SELECTFROM
           ((cleanOrder \/ Delta)/\-(acceptedBy;V[Processor*Order])/\-(rejectedBy;V[P
       (MAINTAINING -cleanOrder \/ acceptedBy;V[Processor*Order] \/ rejectedBy;V[Proc
       SELECT x:Processor FROM codomain(rejectedBy);
         INSERT INTO V[Processor*Order] SELECTFROM
           V[Processor*Order];((cleanOrder \/ Delta)/\-(acceptedBy;V[Processor*Order]
       (MAINTAINING -cleanOrder \/ acceptedBy;V[Processor*Order] \/ rejectedBy;V[Proc
       SELECT x:Processor FROM codomain(V[Order*Processor]);
         INSERT INTO rejectedBy SELECTFROM
           ((cleanOrder \/ Delta)/\-(acceptedBy;V[Processor*Order])/\-(rejectedBy;V[P
       (MAINTAINING -cleanOrder \/ acceptedBy;V[Processor*Order] \/ rejectedBy;V[Proc
(MAINTAINING -cleanOrder \/ acceptedBy;V[Processor*Order] \/ rejectedBy;V[Processor*O

ON DELETE Delta FROM cleanOrder EXECUTE    -- (ECA rule 10)
BLOCK
(CANNOT CHANGE -cleanOrder \/ acceptedBy;V[Processor*Order] \/ rejectedBy;V[Processor

ON INSERT Delta IN acceptedBy EXECUTE    -- (ECA rule 11)
BLOCK
(CANNOT CHANGE -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippedTo;i
(CANNOT CHANGE -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenSentTo

ON DELETE Delta FROM acceptedBy EXECUTE    -- (ECA rule 12)
BLOCK
(CANNOT CHANGE -acceptedBy \/ correspondsTo~;readyForShipping;V[Delivery*Processor] F
```

```
ON INSERT Delta IN readyForShipping EXECUTE    -- (ECA rule 13)
ALL of ONE of CREATE x:Order;
                ALL of INSERT INTO correspondsTo~ SELECTFROM
                         V[Order*Delivery];((readyForShipping \/ Delta)/\-((hasBeenSh
                       ONE of CREATE x:Processor;
                                ALL of INSERT INTO acceptedBy~ SELECTFROM
                                         V[Processor*Delivery];((readyForShipping \/ I
                                       INSERT INTO hasBeenShippedBy SELECTFROM
                                         ((readyForShipping \/ Delta)/\-((hasBeenShipp
                                  SELECT x:Processor FROM codomain(hasBeenShippedBy);
                                    INSERT INTO acceptedBy~ SELECTFROM
                                      V[Processor*Delivery];((readyForShipping \/ Delta)/
                                  SELECT x:Processor FROM codomain(acceptedBy);
                                    INSERT INTO hasBeenShippedBy SELECTFROM
                                      ((readyForShipping \/ Delta)/\-((hasBeenShippedBy;a
                       ONE of CREATE x:Client;
                                ALL of INSERT INTO isPlacedBy~ SELECTFROM
                                         V[Client*Delivery];((readyForShipping \/ Delt
                                       INSERT INTO hasBeenShippedTo SELECTFROM
                                         ((readyForShipping \/ Delta)/\-((hasBeenShipp
                                  SELECT x:Client FROM codomain(hasBeenShippedTo);
                                    INSERT INTO isPlacedBy~ SELECTFROM
                                      V[Client*Delivery];((readyForShipping \/ Delta)/\-(
                                  SELECT x:Client FROM codomain(isPlacedBy);
                                    INSERT INTO hasBeenShippedTo SELECTFROM
                                      ((readyForShipping \/ Delta)/\-((hasBeenShippedBy;a
                  (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasB
                (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeen
                SELECT x:Order FROM codomain((hasBeenShippedBy;acceptedBy~/\hasBeenShipp
                  INSERT INTO correspondsTo~ SELECTFROM
                    V[Order*Delivery];((readyForShipping \/ Delta)/\-((hasBeenShippedBy
                (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeen
                SELECT x:Order FROM codomain(correspondsTo);
                  ALL of ONE of CREATE x:Processor;
                                ALL of INSERT INTO acceptedBy~ SELECTFROM
                                         V[Processor*Delivery];((readyForShipping \/ I
                                       INSERT INTO hasBeenShippedBy SELECTFROM
                                         ((readyForShipping \/ Delta)/\-((hasBeenShipp
                                  SELECT x:Processor FROM codomain(hasBeenShippedBy);
                                    INSERT INTO acceptedBy~ SELECTFROM
                                      V[Processor*Delivery];((readyForShipping \/ Delta)/
                                  SELECT x:Processor FROM codomain(acceptedBy);
                                    INSERT INTO hasBeenShippedBy SELECTFROM
                                      ((readyForShipping \/ Delta)/\-((hasBeenShippedBy;a
                       ONE of CREATE x:Client;
                                ALL of INSERT INTO isPlacedBy~ SELECTFROM
                                         V[Client*Delivery];((readyForShipping \/ Delt
                                       INSERT INTO hasBeenShippedTo SELECTFROM
                                         ((readyForShipping \/ Delta)/\-((hasBeenShipp
                                  SELECT x:Client FROM codomain(hasBeenShippedTo);
```

```
                          INSERT INTO isPlacedBy~ SELECTFROM
                            V[Client*Delivery];((readyForShipping \/ Delta)/\-(
                        SELECT x:Client FROM codomain(isPlacedBy);
                          INSERT INTO hasBeenShippedTo SELECTFROM
                            ((readyForShipping \/ Delta)/\-((hasBeenShippedBy;a
        (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBee
        ALL of ONE of CREATE x:Processor;
                          ALL of INSERT INTO acceptedBy~ SELECTFROM
                                   'x'[Processor];V[Processor*Delivery];((readyFo
                                 INSERT INTO hasBeenShippedBy SELECTFROM
                                   ((readyForShipping;correspondsTo \/ Delta;corr
                        SELECT x:Processor FROM codomain(hasBeenShippedBy);
                          INSERT INTO acceptedBy~ SELECTFROM
                            'x'[Processor];V[Processor*Delivery];((readyForShippi
                        SELECT x:Processor FROM codomain(acceptedBy);
                          INSERT INTO hasBeenShippedBy SELECTFROM
                            ((readyForShipping;correspondsTo \/ Delta;corresponds
              ONE of CREATE x:Client;
                          ALL of INSERT INTO isPlacedBy~ SELECTFROM
                                   'x'[Client];V[Client*Delivery];((readyForShipp
                                 INSERT INTO hasBeenShippedTo SELECTFROM
                                   ((readyForShipping;correspondsTo \/ Delta;corr
                        SELECT x:Client FROM codomain(hasBeenShippedTo);
                          INSERT INTO isPlacedBy~ SELECTFROM
                            'x'[Client];V[Client*Delivery];((readyForShipping;cor
                        SELECT x:Client FROM codomain(isPlacedBy);
                          INSERT INTO hasBeenShippedTo SELECTFROM
                            ((readyForShipping;correspondsTo \/ Delta;corresponds
        (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBee
(MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippe
ONE of CREATE x:Invoice;
          ALL of ONE of CREATE x:Order;
                          ALL of INSERT INTO correspondsTo~ SELECTFROM
                                   V[Order*Invoice];V[Invoice*Delivery];((ready
                                 ONE of CREATE x:Processor;
                                          ALL of INSERT INTO acceptedBy~ SELECT
                                                   V[Processor*Invoice];V[Invoi
                                                 INSERT INTO hasBeenSentBy SELE
                                                   V[Invoice*Delivery];((readyF
                                        SELECT x:Processor FROM codomain(hasBee
                                          INSERT INTO acceptedBy~ SELECTFROM
                                            V[Processor*Invoice];V[Invoice*Deli
                                        SELECT x:Processor FROM codomain(accept
                                          INSERT INTO hasBeenSentBy SELECTFROM
                                            V[Invoice*Delivery];((readyForShipp
                                 ONE of CREATE x:Client;
                                          ALL of INSERT INTO isPlacedBy~ SELECT
                                                   V[Client*Invoice];V[Invoice*
                                                 INSERT INTO hasBeenSentTo SELE
                                                   V[Invoice*Delivery];((readyF
```

```
                                    SELECT x:Client FROM codomain(hasBeenSe
                                      INSERT INTO isPlacedBy~ SELECTFROM
                                        V[Client*Invoice];V[Invoice*Deliver
                                    SELECT x:Client FROM codomain(isPlacedB
                                      INSERT INTO hasBeenSentTo SELECTFROM
                                        V[Invoice*Delivery];((readyForShipp
                          SELECT x:Order FROM codomain((hasBeenSentBy;acceptedBy~
                            INSERT INTO correspondsTo~ SELECTFROM
                              V[Order*Invoice];V[Invoice*Delivery];((readyForShip
                          SELECT x:Order FROM codomain(correspondsTo);
                            ALL of ONE of CREATE x:Processor;
                                        ALL of INSERT INTO acceptedBy~ SELECT
                                                 V[Processor*Invoice];V[Invoi
                                               INSERT INTO hasBeenSentBy SELE
                                                 V[Invoice*Delivery];((readyF
                                    SELECT x:Processor FROM codomain(hasBee
                                      INSERT INTO acceptedBy~ SELECTFROM
                                        V[Processor*Invoice];V[Invoice*Deli
                                    SELECT x:Processor FROM codomain(accept
                                      INSERT INTO hasBeenSentBy SELECTFROM
                                        V[Invoice*Delivery];((readyForShipp
                                ONE of CREATE x:Client;
                                        ALL of INSERT INTO isPlacedBy~ SELECT
                                                 V[Client*Invoice];V[Invoice*
                                               INSERT INTO hasBeenSentTo SELE
                                                 V[Invoice*Delivery];((readyF
                                    SELECT x:Client FROM codomain(hasBeenSe
                                      INSERT INTO isPlacedBy~ SELECTFROM
                                        V[Client*Invoice];V[Invoice*Deliver
                                    SELECT x:Client FROM codomain(isPlacedB
                                      INSERT INTO hasBeenSentTo SELECTFROM
                                        V[Invoice*Delivery];((readyForShipp
                  INSERT INTO covers~ SELECTFROM
                    ((readyForShipping \/ Delta)/\-(covers~;(hasBeenSentBy;accep
            (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/
          (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\h
          SELECT x:Invoice FROM codomain(covers~);
            ONE of CREATE x:Order;
                    ALL of INSERT INTO correspondsTo~ SELECTFROM
                             V[Order*Invoice];V[Invoice*Delivery];((readyForShip
                           ONE of CREATE x:Processor;
                                   ALL of INSERT INTO acceptedBy~ SELECTFROM
                                             V[Processor*Invoice];V[Invoice*Deli
                                           INSERT INTO hasBeenSentBy SELECTFROM
                                             V[Invoice*Delivery];((readyForShipp
                                  SELECT x:Processor FROM codomain(hasBeenSentBy
                                    INSERT INTO acceptedBy~ SELECTFROM
                                      V[Processor*Invoice];V[Invoice*Delivery];(
                                  SELECT x:Processor FROM codomain(acceptedBy);
                                    INSERT INTO hasBeenSentBy SELECTFROM
```

```
                                V[Invoice*Delivery];((readyForShipping \/ I
                        ONE of CREATE x:Client;
                                  ALL of INSERT INTO isPlacedBy~ SELECTFROM
                                           V[Client*Invoice];V[Invoice*Deliver
                                         INSERT INTO hasBeenSentTo SELECTFROM
                                           V[Invoice*Delivery];((readyForShipp
                                      SELECT x:Client FROM codomain(hasBeenSentTo);
                                        INSERT INTO isPlacedBy~ SELECTFROM
                                          V[Client*Invoice];V[Invoice*Delivery];((re
                                      SELECT x:Client FROM codomain(isPlacedBy);
                                        INSERT INTO hasBeenSentTo SELECTFROM
                                          V[Invoice*Delivery];((readyForShipping \/ I
                                SELECT x:Order FROM codomain((hasBeenSentBy;acceptedBy~/\hasBe
                                  INSERT INTO correspondsTo~ SELECTFROM
                                    V[Order*Invoice];V[Invoice*Delivery];((readyForShipping \/
                                SELECT x:Order FROM codomain(correspondsTo);
                                  ALL of ONE of CREATE x:Processor;
                                                  ALL of INSERT INTO acceptedBy~ SELECTFROM
                                                           V[Processor*Invoice];V[Invoice*Deli
                                                         INSERT INTO hasBeenSentBy SELECTFROM
                                                           V[Invoice*Delivery];((readyForShipp
                                                      SELECT x:Processor FROM codomain(hasBeenSentBy
                                                        INSERT INTO acceptedBy~ SELECTFROM
                                                          V[Processor*Invoice];V[Invoice*Delivery];(
                                                      SELECT x:Processor FROM codomain(acceptedBy);
                                                        INSERT INTO hasBeenSentBy SELECTFROM
                                                          V[Invoice*Delivery];((readyForShipping \/ I
                                                ONE of CREATE x:Client;
                                                          ALL of INSERT INTO isPlacedBy~ SELECTFROM
                                                                   V[Client*Invoice];V[Invoice*Deliver
                                                                 INSERT INTO hasBeenSentTo SELECTFROM
                                                                   V[Invoice*Delivery];((readyForShipp
                                                              SELECT x:Client FROM codomain(hasBeenSentTo);
                                                                INSERT INTO isPlacedBy~ SELECTFROM
                                                                  V[Client*Invoice];V[Invoice*Delivery];((re
                                                              SELECT x:Client FROM codomain(isPlacedBy);
                                                                INSERT INTO hasBeenSentTo SELECTFROM
                                                                  V[Invoice*Delivery];((readyForShipping \/ I
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\ha
SELECT x:Invoice FROM codomain(correspondsTo;(acceptedBy;hasBeenSentBy~
  INSERT INTO covers~ SELECTFROM
    ((readyForShipping \/ Delta)/\-(covers~;(hasBeenSentBy;acceptedBy~/
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\ha
CREATE x:Order;
  ALL of INSERT INTO correspondsTo~ SELECTFROM
           V[Order*Delivery];((readyForShipping \/ Delta)/\-(covers~;(ha
         ONE of CREATE x:Invoice;
                  ALL of ONE of CREATE x:Processor;
                                  ALL of INSERT INTO acceptedBy~ SELECT
                                           V[Processor*Invoice];V[Invoi
```

```
                                                    INSERT INTO hasBeenSentBy SELE
                                                      V[Invoice*Delivery];((readyF
                                                SELECT x:Processor FROM codomain(hasBee
                                                  INSERT INTO acceptedBy~ SELECTFROM
                                                    V[Processor*Invoice];V[Invoice*Deli
                                                SELECT x:Processor FROM codomain(accept
                                                  INSERT INTO hasBeenSentBy SELECTFROM
                                                    V[Invoice*Delivery];((readyForShipp
                                              ONE of CREATE x:Client;
                                                      ALL of INSERT INTO isPlacedBy~ SELECT
                                                               V[Client*Invoice];V[Invoice*
                                                             INSERT INTO hasBeenSentTo SELE
                                                               V[Invoice*Delivery];((readyF
                                                SELECT x:Client FROM codomain(hasBeenSe
                                                  INSERT INTO isPlacedBy~ SELECTFROM
                                                    V[Client*Invoice];V[Invoice*Deliver
                                                SELECT x:Client FROM codomain(isPlacedB
                                                  INSERT INTO hasBeenSentTo SELECTFROM
                                                    V[Invoice*Delivery];((readyForShipp
                                              INSERT INTO covers~ SELECTFROM
                                                ((readyForShipping \/ Delta)/\-(covers~;(has
                                          SELECT x:Invoice FROM codomain(covers~);
                                            ALL of ONE of CREATE x:Processor;
                                                      ALL of INSERT INTO acceptedBy~ SELECT
                                                               V[Processor*Invoice];V[Invoi
                                                             INSERT INTO hasBeenSentBy SELE
                                                               V[Invoice*Delivery];((readyF
                                                SELECT x:Processor FROM codomain(hasBee
                                                  INSERT INTO acceptedBy~ SELECTFROM
                                                    V[Processor*Invoice];V[Invoice*Deli
                                                SELECT x:Processor FROM codomain(accept
                                                  INSERT INTO hasBeenSentBy SELECTFROM
                                                    V[Invoice*Delivery];((readyForShipp
                                              ONE of CREATE x:Client;
                                                      ALL of INSERT INTO isPlacedBy~ SELECT
                                                               V[Client*Invoice];V[Invoice*
                                                             INSERT INTO hasBeenSentTo SELE
                                                               V[Invoice*Delivery];((readyF
                                                SELECT x:Client FROM codomain(hasBeenSe
                                                  INSERT INTO isPlacedBy~ SELECTFROM
                                                    V[Client*Invoice];V[Invoice*Deliver
                                                SELECT x:Client FROM codomain(isPlacedB
                                                  INSERT INTO hasBeenSentTo SELECTFROM
                                                    V[Invoice*Delivery];((readyForShipp
                                          SELECT x:Invoice FROM codomain((acceptedBy;hasBeenSentB
                                            INSERT INTO covers~ SELECTFROM
                                              ((readyForShipping \/ Delta)/\-(covers~;(hasBeenSen
                                    (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/
                                  (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\ha
                                  SELECT x:Order FROM codomain(covers~;(hasBeenSentBy;acceptedBy~/\hasBeen
```

```
  INSERT INTO correspondsTo~ SELECTFROM
    V[Order*Delivery];((readyForShipping \/ Delta)/\-(covers~;(hasBeenS
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\h
SELECT x:Order FROM codomain(correspondsTo);
  ONE of CREATE x:Invoice;
           ALL of ONE of CREATE x:Processor;
                        ALL of INSERT INTO acceptedBy~ SELECTFROM
                                 V[Processor*Invoice];V[Invoice*Deli
                               INSERT INTO hasBeenSentBy SELECTFROM
                                 V[Invoice*Delivery];((readyForShipp
                      SELECT x:Processor FROM codomain(hasBeenSentBy
                        INSERT INTO acceptedBy~ SELECTFROM
                          V[Processor*Invoice];V[Invoice*Delivery];(
                      SELECT x:Processor FROM codomain(acceptedBy);
                        INSERT INTO hasBeenSentBy SELECTFROM
                          V[Invoice*Delivery];((readyForShipping \/
                  ONE of CREATE x:Client;
                        ALL of INSERT INTO isPlacedBy~ SELECTFROM
                                 V[Client*Invoice];V[Invoice*Deliver
                               INSERT INTO hasBeenSentTo SELECTFROM
                                 V[Invoice*Delivery];((readyForShipp
                      SELECT x:Client FROM codomain(hasBeenSentTo);
                        INSERT INTO isPlacedBy~ SELECTFROM
                          V[Client*Invoice];V[Invoice*Delivery];((re
                      SELECT x:Client FROM codomain(isPlacedBy);
                        INSERT INTO hasBeenSentTo SELECTFROM
                          V[Invoice*Delivery];((readyForShipping \/
                  INSERT INTO covers~ SELECTFROM
                    ((readyForShipping \/ Delta)/\-(covers~;(hasBeenSen
         SELECT x:Invoice FROM codomain(covers~);
           ALL of ONE of CREATE x:Processor;
                        ALL of INSERT INTO acceptedBy~ SELECTFROM
                                 V[Processor*Invoice];V[Invoice*Deli
                               INSERT INTO hasBeenSentBy SELECTFROM
                                 V[Invoice*Delivery];((readyForShipp
                      SELECT x:Processor FROM codomain(hasBeenSentBy
                        INSERT INTO acceptedBy~ SELECTFROM
                          V[Processor*Invoice];V[Invoice*Delivery];(
                      SELECT x:Processor FROM codomain(acceptedBy);
                        INSERT INTO hasBeenSentBy SELECTFROM
                          V[Invoice*Delivery];((readyForShipping \/
                  ONE of CREATE x:Client;
                        ALL of INSERT INTO isPlacedBy~ SELECTFROM
                                 V[Client*Invoice];V[Invoice*Deliver
                               INSERT INTO hasBeenSentTo SELECTFROM
                                 V[Invoice*Delivery];((readyForShipp
                      SELECT x:Client FROM codomain(hasBeenSentTo);
                        INSERT INTO isPlacedBy~ SELECTFROM
                          V[Client*Invoice];V[Invoice*Delivery];((re
                      SELECT x:Client FROM codomain(isPlacedBy);
```

```
                              INSERT INTO hasBeenSentTo SELECTFROM
                                V[Invoice*Delivery];((readyForShipping \/ 
          SELECT x:Invoice FROM codomain((acceptedBy;hasBeenSentBy~/\isP
            INSERT INTO covers~ SELECTFROM
              ((readyForShipping \/ Delta)/\-(covers~;(hasBeenSentBy;acc
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\h
SELECT x:Order FROM codomain((hasBeenSentBy;acceptedBy~/\hasBeenSentTo;
  INSERT INTO correspondsTo~ SELECTFROM
    V[Order*Invoice];((covers;readyForShipping \/ covers;Delta)/\-((has
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\h
SELECT x:Order FROM codomain(correspondsTo);
  ALL of ONE of CREATE x:Processor;
                  ALL of INSERT INTO acceptedBy~ SELECTFROM
                           V[Processor*Invoice];((covers;readyForShippi
                         INSERT INTO hasBeenSentBy SELECTFROM
                           ((covers;readyForShipping \/ covers;Delta)/\
                SELECT x:Processor FROM codomain(hasBeenSentBy);
                  INSERT INTO acceptedBy~ SELECTFROM
                    V[Processor*Invoice];((covers;readyForShipping \/ c
                SELECT x:Processor FROM codomain(acceptedBy);
                  INSERT INTO hasBeenSentBy SELECTFROM
                    ((covers;readyForShipping \/ covers;Delta)/\-((hasB
          ONE of CREATE x:Client;
                  ALL of INSERT INTO isPlacedBy~ SELECTFROM
                           V[Client*Invoice];((covers;readyForShipping 
                         INSERT INTO hasBeenSentTo SELECTFROM
                           ((covers;readyForShipping \/ covers;Delta)/\
                SELECT x:Client FROM codomain(hasBeenSentTo);
                  INSERT INTO isPlacedBy~ SELECTFROM
                    V[Client*Invoice];((covers;readyForShipping \/ cove
                SELECT x:Client FROM codomain(isPlacedBy);
                  INSERT INTO hasBeenSentTo SELECTFROM
                    ((covers;readyForShipping \/ covers;Delta)/\-((hasB
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\h
ALL of ONE of CREATE x:Processor;
                ALL of INSERT INTO acceptedBy~ SELECTFROM
                         'x'[Processor];V[Processor*Invoice];((covers;r
                       INSERT INTO hasBeenSentBy SELECTFROM
                         ((covers;readyForShipping;correspondsTo \/ cov
              SELECT x:Processor FROM codomain(hasBeenSentBy);
                INSERT INTO acceptedBy~ SELECTFROM
                  'x'[Processor];V[Processor*Invoice];((covers;readyFor
              SELECT x:Processor FROM codomain(acceptedBy);
                INSERT INTO hasBeenSentBy SELECTFROM
                  ((covers;readyForShipping;correspondsTo \/ covers;Del
        ONE of CREATE x:Client;
                ALL of INSERT INTO isPlacedBy~ SELECTFROM
                         'x'[Client];V[Client*Invoice];((covers;readyFo
                       INSERT INTO hasBeenSentTo SELECTFROM
                         ((covers;readyForShipping;correspondsTo \/ cov
```

```
                                SELECT x:Client FROM codomain(hasBeenSentTo);
                                  INSERT INTO isPlacedBy~ SELECTFROM
                                    'x'[Client];V[Client*Invoice];((covers;readyForShippi
                                SELECT x:Client FROM codomain(isPlacedBy);
                                  INSERT INTO hasBeenSentTo SELECTFROM
                                    ((covers;readyForShipping;correspondsTo \/ covers;Del
                (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\ha
                SELECT x:Invoice FROM codomain(covers~);
                  ALL of ONE of CREATE x:Processor;
                                    ALL of INSERT INTO acceptedBy~ SELECTFROM
                                             V[Processor*Invoice];V[Invoice*Delivery];((r
                                           INSERT INTO hasBeenSentBy SELECTFROM
                                             V[Invoice*Delivery];((readyForShipping;corre
                                  SELECT x:Processor FROM codomain(hasBeenSentBy);
                                    INSERT INTO acceptedBy~ SELECTFROM
                                      V[Processor*Invoice];V[Invoice*Delivery];((readyFor
                                  SELECT x:Processor FROM codomain(acceptedBy);
                                    INSERT INTO hasBeenSentBy SELECTFROM
                                      V[Invoice*Delivery];((readyForShipping;correspondsT
                           ONE of CREATE x:Client;
                                    ALL of INSERT INTO isPlacedBy~ SELECTFROM
                                             V[Client*Invoice];V[Invoice*Delivery];((read
                                           INSERT INTO hasBeenSentTo SELECTFROM
                                             V[Invoice*Delivery];((readyForShipping;corre
                                  SELECT x:Client FROM codomain(hasBeenSentTo);
                                    INSERT INTO isPlacedBy~ SELECTFROM
                                      V[Client*Invoice];V[Invoice*Delivery];((readyForShi
                                  SELECT x:Client FROM codomain(isPlacedBy);
                                    INSERT INTO hasBeenSentTo SELECTFROM
                                      V[Invoice*Delivery];((readyForShipping;correspondsT
                (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\ha
                SELECT x:Invoice FROM codomain((acceptedBy;hasBeenSentBy~/\isPlacedBy;ha
                  INSERT INTO covers~ SELECTFROM
                    ((readyForShipping;correspondsTo \/ Delta;correspondsTo)/\-(covers~
                (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\ha
        (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
(MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippedTo;isP
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenSentTo;is

ON DELETE Delta FROM readyForShipping EXECUTE    -- (ECA rule 14)
BLOCK
(CANNOT CHANGE -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippedTo;is
(CANNOT CHANGE -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenSentTo

ON INSERT Delta IN isPlacedBy EXECUTE    -- (ECA rule 15)
BLOCK
(CANNOT CHANGE -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippedTo;is
(CANNOT CHANGE -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenSentTo

ON DELETE Delta FROM isPlacedBy EXECUTE    -- (ECA rule 16)
```

```
ALL of ONE of DELETE FROM readyForShipping SELECTFROM
                readyForShipping/\-((hasBeenShippedBy;acceptedBy~/\(hasBeenShippedTo;
              (TO MAINTAIN -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeen
              SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
                REMOVE x:Delivery;
                  ALL of DELETE FROM correspondsTo SELECTFROM
                           'x'[Delivery];V[Delivery];(readyForShipping;correspondsTo/
                         DELETE FROM readyForShipping SELECTFROM
                           (readyForShipping;correspondsTo/\(-(hasBeenShippedBy;accep
                  (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\has
                (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBe
              (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeen
              SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
                DELETE FROM correspondsTo SELECTFROM
                  'x'[Delivery];V[Delivery];(readyForShipping;correspondsTo/\(-(hasBe
              (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeen
              SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
                DELETE FROM readyForShipping SELECTFROM
                  (readyForShipping;correspondsTo/\(-(hasBeenShippedBy;acceptedBy~) \
              (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeen
        (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippe
        ONE of DELETE FROM readyForShipping SELECTFROM
                readyForShipping/\-(covers~;(hasBeenSentBy;acceptedBy~/\(hasBeenSentT
              (TO MAINTAIN -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\ha
              SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
                REMOVE x:Delivery;
                  ALL of DELETE FROM readyForShipping SELECTFROM
                           'x'[Delivery];V[Delivery*Invoice];(covers;readyForShipping
                         DELETE FROM covers SELECTFROM
                           (covers;readyForShipping/\-((hasBeenSentBy;acceptedBy~/\(ha
                  (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy
                (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/
              (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\ha
              SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
                DELETE FROM readyForShipping SELECTFROM
                  'x'[Delivery];V[Delivery*Invoice];(covers;readyForShipping/\-((hasBe
              (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\ha
              SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
                DELETE FROM covers SELECTFROM
                  (covers;readyForShipping/\-((hasBeenSentBy;acceptedBy~/\(hasBeenSen
              (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\ha
              SELECT x:Delivery FROM codomain(covers/\correspondsTo~;readyForShipping
                REMOVE x:Delivery;
                  ALL of ONE of SELECT x:Delivery FROM codomain(readyForShipping/\cor
                                  REMOVE x:Delivery;
                                    ALL of DELETE FROM correspondsTo SELECTFROM
                                             'x'[Delivery];V[Delivery];('x'[Delivery]
                                           DELETE FROM readyForShipping SELECTFROM
                                             V[Delivery*Invoice];(covers;readyForShip
                                SELECT x:Delivery FROM codomain(readyForShipping/\cor
```

```
                    DELETE FROM correspondsTo SELECTFROM
                      'x'[Delivery];V[Delivery];('x'[Delivery];V[Delive
                  SELECT x:Delivery FROM codomain(readyForShipping/\cor
                    DELETE FROM readyForShipping SELECTFROM
                      V[Delivery*Invoice];(covers;readyForShipping;corr
            DELETE FROM covers SELECTFROM
              (covers;readyForShipping;correspondsTo/\(-(hasBeenSentBy;a
    (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy
  (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\h
SELECT x:Delivery FROM codomain(covers/\correspondsTo~;readyForShipping
  ONE of SELECT x:Delivery FROM codomain(readyForShipping/\correspondsT
           REMOVE x:Delivery;
             ALL of DELETE FROM correspondsTo SELECTFROM
                      'x'[Delivery];V[Delivery];('x'[Delivery];V[Delive
                    DELETE FROM readyForShipping SELECTFROM
                      V[Delivery*Invoice];(covers;readyForShipping;corr
         SELECT x:Delivery FROM codomain(readyForShipping/\correspondsT
           DELETE FROM correspondsTo SELECTFROM
             'x'[Delivery];V[Delivery];('x'[Delivery];V[Delivery*Invoic
         SELECT x:Delivery FROM codomain(readyForShipping/\correspondsT
           DELETE FROM readyForShipping SELECTFROM
             V[Delivery*Invoice];(covers;readyForShipping;correspondsTo
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\h
SELECT x:Delivery FROM codomain(covers/\correspondsTo~;readyForShipping
  DELETE FROM covers SELECTFROM
    (covers;readyForShipping;correspondsTo/\(-(hasBeenSentBy;acceptedBy
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\h
SELECT x:Delivery FROM codomain(covers;readyForShipping/\correspondsTo~
  REMOVE x:Delivery;
    ALL of DELETE FROM correspondsTo SELECTFROM
             'x'[Delivery];V[Delivery*Invoice];(covers;readyForShipping
           ONE of SELECT x:Delivery FROM codomain(covers/\readyForShipp
                    REMOVE x:Delivery;
                      ALL of DELETE FROM readyForShipping SELECTFROM
                               V[Delivery*Invoice];(covers;readyForShip
                             DELETE FROM covers SELECTFROM
                               ((covers;readyForShipping;correspondsTo/
                  SELECT x:Delivery FROM codomain(covers/\readyForShipp
                    DELETE FROM readyForShipping SELECTFROM
                      V[Delivery*Invoice];(covers;readyForShipping;corr
                  SELECT x:Delivery FROM codomain(covers/\readyForShipp
                    DELETE FROM covers SELECTFROM
                      ((covers;readyForShipping;correspondsTo/\(-(hasBe
    (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy
  (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\h
SELECT x:Delivery FROM codomain(covers;readyForShipping/\correspondsTo~
  DELETE FROM correspondsTo SELECTFROM
    'x'[Delivery];V[Delivery*Invoice];(covers;readyForShipping;correspo
```

```
                (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\ha
                SELECT x:Delivery FROM codomain(covers;readyForShipping/\correspondsTo~
                  ONE of SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
                           REMOVE x:Delivery;
                             ALL of DELETE FROM readyForShipping SELECTFROM
                                      V[Delivery*Invoice];(covers;readyForShipping;corre
                                    DELETE FROM covers SELECTFROM
                                      ((covers;readyForShipping;correspondsTo/\(-(hasBe
                           SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
                             DELETE FROM readyForShipping SELECTFROM
                               V[Delivery*Invoice];(covers;readyForShipping;correspondsTo
                           SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
                             DELETE FROM covers SELECTFROM
                               ((covers;readyForShipping;correspondsTo/\(-(hasBeenSentBy;
                (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\ha
                SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
                  REMOVE x:Delivery;
                    ALL of DELETE FROM correspondsTo SELECTFROM
                             'x'[Delivery];V[Delivery];(readyForShipping;correspondsTo/
                           DELETE FROM readyForShipping SELECTFROM
                             (readyForShipping;correspondsTo/\-(covers~;(hasBeenSentBy;
                    (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy
                  (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/
                (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\ha
                SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
                  DELETE FROM correspondsTo SELECTFROM
                    'x'[Delivery];V[Delivery];(readyForShipping;correspondsTo/\-(covers
                (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\ha
                SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
                  DELETE FROM readyForShipping SELECTFROM
                    (readyForShipping;correspondsTo/\-(covers~;(hasBeenSentBy;acceptedB
                (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\ha
        (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
(MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippedTo;isP
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenSentTo;i

ON INSERT Delta IN hasBeenShippedTo EXECUTE    -- (ECA rule 17)
BLOCK
(CANNOT CHANGE -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippedTo;i

ON DELETE Delta FROM hasBeenShippedTo EXECUTE    -- (ECA rule 18)
ONE of DELETE FROM readyForShipping SELECTFROM
          readyForShipping/\-((hasBeenShippedBy;acceptedBy~/\(hasBeenShippedTo;isPlace
        (TO MAINTAIN -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippe
        SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
          REMOVE x:Delivery;
            ALL of DELETE FROM correspondsTo SELECTFROM
                     'x'[Delivery];V[Delivery];(readyForShipping;correspondsTo/\(-(has
                   DELETE FROM readyForShipping SELECTFROM
```

```
                  (readyForShipping;correspondsTo/\(-(hasBeenShippedBy;acceptedBy~)
          (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenSh
        (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShipp
      (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippe
      SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
        DELETE FROM correspondsTo SELECTFROM
          'x'[Delivery];V[Delivery];(readyForShipping;correspondsTo/\(-(hasBeenShipp
      (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippe
      SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
        DELETE FROM readyForShipping SELECTFROM
          (readyForShipping;correspondsTo/\(-(hasBeenShippedBy;acceptedBy~) \/ -(has
      (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippe
(MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippedTo;isP

ON INSERT Delta IN hasBeenShippedBy EXECUTE    -- (ECA rule 19)
BLOCK
(CANNOT CHANGE -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippedTo;i

ON DELETE Delta FROM hasBeenShippedBy EXECUTE    -- (ECA rule 20)
ONE of DELETE FROM readyForShipping SELECTFROM
        readyForShipping/\-(((hasBeenShippedBy;acceptedBy~ \/ Delta;acceptedBy~)/\ha
      (TO MAINTAIN -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippe
      SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
        REMOVE x:Delivery;
          ALL of DELETE FROM correspondsTo SELECTFROM
                  'x'[Delivery];V[Delivery];(readyForShipping;correspondsTo/\(-(has
                DELETE FROM readyForShipping SELECTFROM
                  (readyForShipping;correspondsTo/\(-(hasBeenShippedBy;acceptedBy~)
          (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenSh
        (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShipp
      (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippe
      SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
        DELETE FROM correspondsTo SELECTFROM
          'x'[Delivery];V[Delivery];(readyForShipping;correspondsTo/\(-(hasBeenShipp
      (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippe
      SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
        DELETE FROM readyForShipping SELECTFROM
          (readyForShipping;correspondsTo/\(-(hasBeenShippedBy;acceptedBy~)/\-(Delta
      (MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippe
(MAINTAINING -readyForShipping \/ (hasBeenShippedBy;acceptedBy~/\hasBeenShippedTo;isP

ON INSERT Delta IN hasBeenSentTo EXECUTE    -- (ECA rule 21)
BLOCK
(CANNOT CHANGE -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenSentTo

ON DELETE Delta FROM hasBeenSentTo EXECUTE    -- (ECA rule 22)
ONE of DELETE FROM readyForShipping SELECTFROM
        readyForShipping/\-(covers~;(hasBeenSentBy;acceptedBy~/\(hasBeenSentTo;isPla
      (TO MAINTAIN -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
```

```
SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
  REMOVE x:Delivery;
    ALL of DELETE FROM readyForShipping SELECTFROM
             'x'[Delivery];V[Delivery*Invoice];(covers;readyForShipping/\-((ha
           DELETE FROM covers SELECTFROM
             (covers;readyForShipping/\-((hasBeenSentBy;acceptedBy~/\(hasBeenS
    (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasB
  (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBee
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
  DELETE FROM readyForShipping SELECTFROM
    'x'[Delivery];V[Delivery*Invoice];(covers;readyForShipping/\-((hasBeenSentB
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
  DELETE FROM covers SELECTFROM
    (covers;readyForShipping/\-((hasBeenSentBy;acceptedBy~/\(hasBeenSentTo;isP
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
SELECT x:Delivery FROM codomain(covers/\correspondsTo~;readyForShipping~);
  REMOVE x:Delivery;
    ALL of ONE of SELECT x:Delivery FROM codomain(readyForShipping/\correspond
                    REMOVE x:Delivery;
                      ALL of DELETE FROM correspondsTo SELECTFROM
                               'x'[Delivery];V[Delivery];('x'[Delivery];V[Deli
                             DELETE FROM readyForShipping SELECTFROM
                               V[Delivery*Invoice];(covers;readyForShipping;co
                  SELECT x:Delivery FROM codomain(readyForShipping/\correspond
                    DELETE FROM correspondsTo SELECTFROM
                      'x'[Delivery];V[Delivery];('x'[Delivery];V[Delivery*Invo
                  SELECT x:Delivery FROM codomain(readyForShipping/\correspond
                    DELETE FROM readyForShipping SELECTFROM
                      V[Delivery*Invoice];(covers;readyForShipping;corresponds
           DELETE FROM covers SELECTFROM
             (covers;readyForShipping;correspondsTo/\(-(hasBeenSentBy;accepted
    (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasB
  (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBee
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
SELECT x:Delivery FROM codomain(covers/\correspondsTo~;readyForShipping~);
  ONE of SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
           REMOVE x:Delivery;
             ALL of DELETE FROM correspondsTo SELECTFROM
                      'x'[Delivery];V[Delivery];('x'[Delivery];V[Delivery*Invo
                    DELETE FROM readyForShipping SELECTFROM
                      V[Delivery*Invoice];(covers;readyForShipping;corresponds
         SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
           DELETE FROM correspondsTo SELECTFROM
             'x'[Delivery];V[Delivery];('x'[Delivery];V[Delivery*Invoice];(cov
         SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
           DELETE FROM readyForShipping SELECTFROM
             V[Delivery*Invoice];(covers;readyForShipping;correspondsTo/\(-(ha
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
```

```
SELECT x:Delivery FROM codomain(covers/\correspondsTo~;readyForShipping~);
  DELETE FROM covers SELECTFROM
    (covers;readyForShipping;correspondsTo/\(-(hasBeenSentBy;acceptedBy~) \/ -
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
SELECT x:Delivery FROM codomain(covers;readyForShipping/\correspondsTo~);
  REMOVE x:Delivery;
    ALL of DELETE FROM correspondsTo SELECTFROM
             'x'[Delivery];V[Delivery*Invoice];(covers;readyForShipping;corres
           ONE of SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
                    REMOVE x:Delivery;
                      ALL of DELETE FROM readyForShipping SELECTFROM
                               V[Delivery*Invoice];(covers;readyForShipping;co
                             DELETE FROM covers SELECTFROM
                               ((covers;readyForShipping;correspondsTo/\(-(has
                  SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
                    DELETE FROM readyForShipping SELECTFROM
                      V[Delivery*Invoice];(covers;readyForShipping;corresponds
                  SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
                    DELETE FROM covers SELECTFROM
                      ((covers;readyForShipping;correspondsTo/\(-(hasBeenSentB
    (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasB
  (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBee
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
SELECT x:Delivery FROM codomain(covers;readyForShipping/\correspondsTo~);
  DELETE FROM correspondsTo SELECTFROM
    'x'[Delivery];V[Delivery*Invoice];(covers;readyForShipping;correspondsTo/\
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
SELECT x:Delivery FROM codomain(covers;readyForShipping/\correspondsTo~);
  ONE of SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
           REMOVE x:Delivery;
             ALL of DELETE FROM readyForShipping SELECTFROM
                      V[Delivery*Invoice];(covers;readyForShipping;corresponds
                    DELETE FROM covers SELECTFROM
                      ((covers;readyForShipping;correspondsTo/\(-(hasBeenSentB
         SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
           DELETE FROM readyForShipping SELECTFROM
             V[Delivery*Invoice];(covers;readyForShipping;correspondsTo/\(-(ha
         SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
           DELETE FROM covers SELECTFROM
             ((covers;readyForShipping;correspondsTo/\(-(hasBeenSentBy;accepte
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
  REMOVE x:Delivery;
    ALL of DELETE FROM correspondsTo SELECTFROM
             'x'[Delivery];V[Delivery];(readyForShipping;correspondsTo/\-(cove
           DELETE FROM readyForShipping SELECTFROM
             (readyForShipping;correspondsTo/\-(covers~;(hasBeenSentBy;accepte
    (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasB
  (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBee
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
```

```
       SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
         DELETE FROM correspondsTo SELECTFROM
           'x'[Delivery];V[Delivery];(readyForShipping;correspondsTo/\-(covers~;(hasB
       (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
       SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
         DELETE FROM readyForShipping SELECTFROM
           (readyForShipping;correspondsTo/\-(covers~;(hasBeenSentBy;acceptedBy~/\(ha
       (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenSentTo;i
(MAINTAINING -hasBeenSentTo \/ hasPaid~ FROM R7)

ON INSERT Delta IN hasBeenSentBy EXECUTE    -- (ECA rule 23)
BLOCK
(CANNOT CHANGE -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenSentTo

ON DELETE Delta FROM hasBeenSentBy EXECUTE    -- (ECA rule 24)
ONE of DELETE FROM readyForShipping SELECTFROM
         readyForShipping/\-(covers~;((hasBeenSentBy;acceptedBy~ \/ Delta;acceptedBy~
       (TO MAINTAIN -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
       SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
         REMOVE x:Delivery;
           ALL of DELETE FROM readyForShipping SELECTFROM
                    'x'[Delivery];V[Delivery*Invoice];(covers;readyForShipping/\-(((h
                  DELETE FROM covers SELECTFROM
                    (covers;readyForShipping/\-(((hasBeenSentBy;acceptedBy~ \/ Delta;
           (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasB
         (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeen
       (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
       SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
         DELETE FROM readyForShipping SELECTFROM
           'x'[Delivery];V[Delivery*Invoice];(covers;readyForShipping/\-(((hasBeenSent
       (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
       SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
         DELETE FROM covers SELECTFROM
           (covers;readyForShipping/\-(((hasBeenSentBy;acceptedBy~ \/ Delta;acceptedBy
       (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
       SELECT x:Delivery FROM codomain(covers/\correspondsTo~;readyForShipping~);
         REMOVE x:Delivery;
           ALL of ONE of SELECT x:Delivery FROM codomain(readyForShipping/\correspond
                           REMOVE x:Delivery;
                             ALL of DELETE FROM correspondsTo SELECTFROM
                                      'x'[Delivery];V[Delivery];('x'[Delivery];V[Deli
                                    DELETE FROM readyForShipping SELECTFROM
                                      V[Delivery*Invoice];(covers;readyForShipping;co
                         SELECT x:Delivery FROM codomain(readyForShipping/\correspond
                           DELETE FROM correspondsTo SELECTFROM
                             'x'[Delivery];V[Delivery];('x'[Delivery];V[Delivery*Invo
                         SELECT x:Delivery FROM codomain(readyForShipping/\correspond
                           DELETE FROM readyForShipping SELECTFROM
```

```
                    V[Delivery*Invoice];(covers;readyForShipping;corresponds
          DELETE FROM covers SELECTFROM
            (covers;readyForShipping;correspondsTo/\(-(hasBeenSentBy;accepted
    (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasB
  (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBee
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
SELECT x:Delivery FROM codomain(covers/\correspondsTo~;readyForShipping~);
  ONE of SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
          REMOVE x:Delivery;
            ALL of DELETE FROM correspondsTo SELECTFROM
                     'x'[Delivery];V[Delivery];('x'[Delivery];V[Delivery*Invo
                   DELETE FROM readyForShipping SELECTFROM
                     V[Delivery*Invoice];(covers;readyForShipping;corresponds
        SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
          DELETE FROM correspondsTo SELECTFROM
            'x'[Delivery];V[Delivery];('x'[Delivery];V[Delivery*Invoice];(cov
        SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
          DELETE FROM readyForShipping SELECTFROM
            V[Delivery*Invoice];(covers;readyForShipping;correspondsTo/\(-(ha
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
SELECT x:Delivery FROM codomain(covers/\correspondsTo~;readyForShipping~);
  DELETE FROM covers SELECTFROM
    (covers;readyForShipping;correspondsTo/\(-(hasBeenSentBy;acceptedBy~)/\-(D
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
SELECT x:Delivery FROM codomain(covers;readyForShipping/\correspondsTo~);
  REMOVE x:Delivery;
    ALL of DELETE FROM correspondsTo SELECTFROM
            'x'[Delivery];V[Delivery*Invoice];(covers;readyForShipping;corres
          ONE of SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
                   REMOVE x:Delivery;
                     ALL of DELETE FROM readyForShipping SELECTFROM
                              V[Delivery*Invoice];(covers;readyForShipping;co
                            DELETE FROM covers SELECTFROM
                              ((covers;readyForShipping;correspondsTo/\(-(has
                 SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
                   DELETE FROM readyForShipping SELECTFROM
                     V[Delivery*Invoice];(covers;readyForShipping;corresponds
                 SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
                   DELETE FROM covers SELECTFROM
                     ((covers;readyForShipping;correspondsTo/\(-(hasBeenSentB
    (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasB
  (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBee
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
SELECT x:Delivery FROM codomain(covers;readyForShipping/\correspondsTo~);
  DELETE FROM correspondsTo SELECTFROM
    'x'[Delivery];V[Delivery*Invoice];(covers;readyForShipping;correspondsTo/\
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
SELECT x:Delivery FROM codomain(covers;readyForShipping/\correspondsTo~);
  ONE of SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
          REMOVE x:Delivery;
```

```
                    ALL of DELETE FROM readyForShipping SELECTFROM
                             V[Delivery*Invoice];(covers;readyForShipping;corresponds
                           DELETE FROM covers SELECTFROM
                             ((covers;readyForShipping;correspondsTo/\(-(hasBeenSentB
                 SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
                   DELETE FROM readyForShipping SELECTFROM
                     V[Delivery*Invoice];(covers;readyForShipping;correspondsTo/\(-(ha
                 SELECT x:Delivery FROM codomain(covers/\readyForShipping~);
                   DELETE FROM covers SELECTFROM
                     ((covers;readyForShipping;correspondsTo/\(-(hasBeenSentBy;accepte
          (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
          SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
            REMOVE x:Delivery;
              ALL of DELETE FROM correspondsTo SELECTFROM
                       'x'[Delivery];V[Delivery];(readyForShipping;correspondsTo/\-(cove
                     DELETE FROM readyForShipping SELECTFROM
                       (readyForShipping;correspondsTo/\-(covers~;((hasBeenSentBy;accept
              (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasB
            (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBee
          (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
          SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
            DELETE FROM correspondsTo SELECTFROM
              'x'[Delivery];V[Delivery];(readyForShipping;correspondsTo/\-(covers~;((has
          (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
          SELECT x:Delivery FROM codomain(readyForShipping/\correspondsTo~);
            DELETE FROM readyForShipping SELECTFROM
              (readyForShipping;correspondsTo/\-(covers~;((hasBeenSentBy;acceptedBy~ \/
          (MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenS
(MAINTAINING -readyForShipping \/ covers~;(hasBeenSentBy;acceptedBy~/\hasBeenSentTo;is
```