From Business Model to Process Pattern in e-commerce

Prasad Jayaweera, Paul Johannesson, and Petia Wohed

Department of Computer and Systems Sciences
Stockholm University and Royal Institute of Technology
Electrum 230, SE-164 40 Kista, Sweden
{prasad, pajo, petia} @dsv.su.se

Abstract

The basic notions in e-commerce are communicative, institutional, and deontic notions such as obligation, responsibility, and trust. The Language Action approach, therefore, seems to be a most promising framework for designing e-commerce systems. However, the penetration of the Language Action approach in industrial practice is still limited. We discuss some reasons for this state of affairs by identifying a number of problems that hinder an effective application of the Language Action approach. We propose modeling techniques and methodological guidelines that can contribute to the solution of these problems. These techniques and guidelines are based on three building blocks: a business model describing the values exchanged in an e-commerce process, a formal and executable language based on communicating state machines, and an automated designer's assistant that guides a user from a business model to an executable process model.

1 Introduction

In e-commerce, business transactions are carried out using IT as a medium. The use of IT enables transactions to be carried out rapidly and to a low cost. As a consequence, new ways of working, new forms of organization, and new business models are emerging, such as virtual enterprises, integrated supply chains, and value networks. A common theme is that of inter-organizational co-operation and communication. Business processes are not carried out within a single organization but across organizational boundaries. As noted in [Weigand98], inter-organizational processes have two distinguishing features. Firstly, the resources needed for a process cannot be assigned centrally as they reside in different organizations. Secondly, the organizations involved in a process have a certain degree of autonomy meaning that no central authority has control over all the co-operating organizations. These features of processes in an e-commerce setting imply that in order to build effective IT-systems, it is necessary to explicitly model and manage communicative, institutional, and deontic notions such as request, acknowledgement, commitment, obligation, responsibility, and trust. Thus, the Language Action approach to communication and information modeling seems to be promising framework for designing e-commerce systems. However, the penetration of the Language Action approach in industrial practice is still low although there exists a

comprehensive body of theoretical as well as applied research in the area, [Winograd86], [Weigand98], [Goldkuhl96], [Dietz00], and. [Johannesson01].

The limits of the applicability of the Language Action approach have been widely discussed in academia, e.g. the Suchman/Winograd debate, [Taylor00]. We acknowledge the importance of the arguments put forward in these discussions, but believe that they are less relevant in e-commerce settings, as e-commerce processes are more formalized and structured than many intra-organizational work processes. Instead, we believe that other factors are more important for the low penetration of the Language Action approach. Our own experience of applying the Language Action approach for e-commerce in industrial case studies as well as in undergraduate teaching has identified three factors that hinder an effective use of the approach:

1. Using the Language Action approach for process modeling easily encourages a low-level perspective where the modeling quickly focuses on communicative acts like requests, replies, acknowledgements, cancellations, etc. Managers often experience this level as too detailed and inadequate starting point for understanding the business objectives motivating the process design.
2. The underlying notions and terminology of the Language Action approach are unfamiliar to most users and designers. They find it difficult to reason and communicate using the specialized terminology of the Language Action approach.
3. There is a considerable distance between Language Action models and executable systems. After having designed a process model using the Language Action approach, there is still much design and implementation work to be done before an executable system is completed.

The purpose of this paper is to suggest methodological guidelines and modeling techniques that can overcome these problems and thereby facilitate the application of the Language Action approach. The proposed guidelines and techniques are based on three building blocks:

1. A business model describing the values exchanged in an e-commerce process.
2. A formal and executable language based on communicating state machines used for modeling processes.
3. An automated designer's assistant that guides a user from a business model to an executable process model.

The paper is structured as follows. Section 2 briefly reviews related research. Section 3 introduces a type of business model built on value exchanges. Section 4 describes BML, a formal language for process modeling. Section 5 describes an automated designer's assistant that supports the task of creating process models. Section 6 concludes the paper and gives directions for further research.

2 Related Research

One of the most well known Language Action approaches is the Action Workflow approach, [Winograd86] and [Mora93]. In this approach, business processes are modeled as loops, see Fig. 1. A loop starts by a request from a customer, followed by a negotiation phase, which results in the provider accepting the request and promising to carry it out. The third step consists of the provider carrying out the request, and the last step is the acknowledgement of the customer that the request has been satisfied.

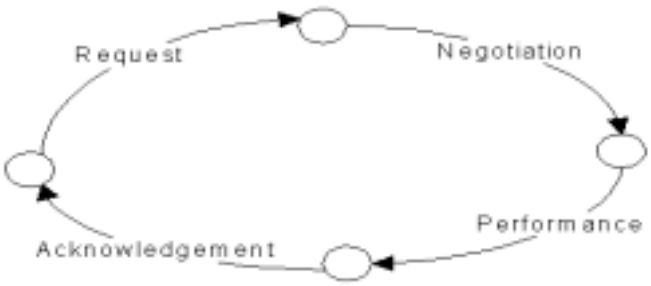

Fig. 1. Action Workflow Loop with four phases

The Action Workflow approach only considers the communicative action. In contrast, DEMO (Dynamic Essential Modeling of Organizations), [Reijswoud99] and [Dietz00], also takes the material action into account. In DEMO, a transaction consists of three phases: the order phase, the execution phase, and the result phase. In the order phase, an actagenic conversation takes place: one actor, the initiator, requests something from another actor, the executor, who can reject or accept. If the executor accepts, the result phase starts and a factagenic conversation takes place. In this conversation, the executor declares that she has completed her task and finally the initiator accepts or rejects this claim.

A third Language Action approach is BAT (Business Action Theory), [Goldkuhl96]. BAT has a more limited scope than the Action Workflow approach as it only addresses business transactions and not works processes in general. For business processes, BAT provides a more elaborated framework for business transactions than Action Workflow by also incorporating preliminary phases, such as contact search. An important novel feature of BAT is the symmetry it introduces by stating that both actors in a business transaction have mutual obligations to each other. This idea exists also in other approaches (see Section 3), and is one basic element for the modeling guidelines proposed in this paper. In contrast to the approaches above and most Language Action modeling, the approach we are proposing starts with a Business Model of reciprocal value transactions. Then based on the business model and identified process patterns based on communicative acts, final executable process model is deduced with the help of designer's assistant.

3 Business Models

When developing an e-commerce system, an important first activity is to design a business model. The purpose of a business model is to describe the fundamental business aspects of the e-commerce system to be built. Thus, a business model describes which actors are involved, what the actors offer each other, and what activities they perform, [Gordijn00]. The central concept in a business model is that of *value*, and the model describes how value is exchanged between actors [Porter98]. This can be contrasted to a process model, which aims at describing the procedural aspects of a process and specifies the control flow of the activities carried out in a process.

Following Gordijn et.al., [Gordijn00b], we identify the following basic notions of a business model:

Actor. An actor is an independent economic and/or legal entity.

Value object. A value object is a service or a thing that is of value to one or more actors.

Value transfer. A value transfer is the transfer of a value object from one actor to another actor.

Value exchange. A value exchange consists of two value transfers, T1 and T2, that satisfy the following condition: if T1 is a value transfer from actor A1 to actor A2, then T2 is a value transfer from A2 to A1. The intuition is that a value exchange consists of two reciprocal acts -

one actor providing another actor with something of value and receiving something of value in return.

(Note that we have omitted several notions in the approach of Gordijn et.al. and simplified others, as this will be sufficient for our present purposes.)

Business model. A business model consists of three parts:

- A - a set of actor types
- VO - a set of value object types
- VE - a set of value exchange types

A business model can be expressed in a graph, see Fig. 2 for an example. In this business model, A = {Customer, Retailer, Importer}, VO = {Car, Money}, VE = {<<Retailer, Customer, Car>, <Customer, Retailer, Money>>, <<Importer, Retailer, Car>, <Retailer, Importer, Money>>} There are two value exchanges: purchase and import. In a purchase, a retailer provides a customer with a car in return for money. In an import, an importer provides a retailer with a car in return for money.

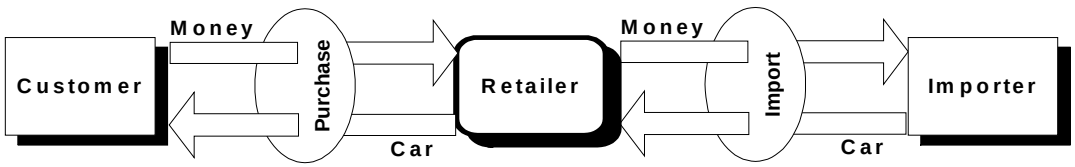

Fig. 2. Business Model for Car Sale example.

4 Business Modeling Language

This section briefly introduces a language based on communicating state machines, BML (Business Modeling Language), which is developed by Viewlocity, [Johannesson00] & [Viewlocity]. The language has similarities to SDL (Specification and Description Language), [Belina91] & [SDL]. BML is a communication oriented process language, which means that it focuses on describing interaction between actors through the sending and receiving of messages. An important advantage of BML is that it can be used for the specification and design as well as maintenance of systems. This means that the same language can be used in different phases of a system's life cycle: in feasibility analysis, in requirement specification, in the design and implementation phases, and even in the operation phase. This enables different categories of stakeholders to use the same language for different purposes.
The dynamic behaviour of a system is described by using process models, which visualize the order in which the messages shall be sent and received, see Fig. 3.

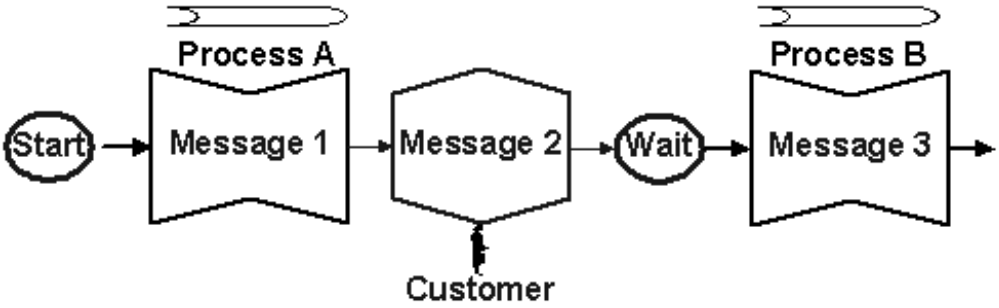

Fig. 3. The BML diagram visualizes the order in which the messages shall be sent and received in a process. Note that the figure only shows the beginning of a Process.

The process segment shown in Fig. 3. describes the situation when Message 1 is received from Process A, Message 2 will be sent out to the Customer. Then it waits for Message 3 from Process B. In the original semantics of BML, a transition from a wait state can only be made upon immediate receives massage. But for our work we have extended BML semantics such that wait states can be abstracted by delaying immediately following exact inter process communications as described in the next section.

The main BML symbols are the following, see also Fig. 4:

Wait for Event and **Start.** The process instance is waiting in the Wait for Event state until a message is received or a timer has expired. A Wait for Event symbol with a name "Start" is the starting state.

Stop. Describes the end of the flow of the process instance.

Receive Message. Describes the consumption of a message from the input queue.

Send Message. Describes the sending of a message.

Automated Business Decision. The control flow is dynamically changed depending on different business rules.

Human actor. Symbols of external actors.

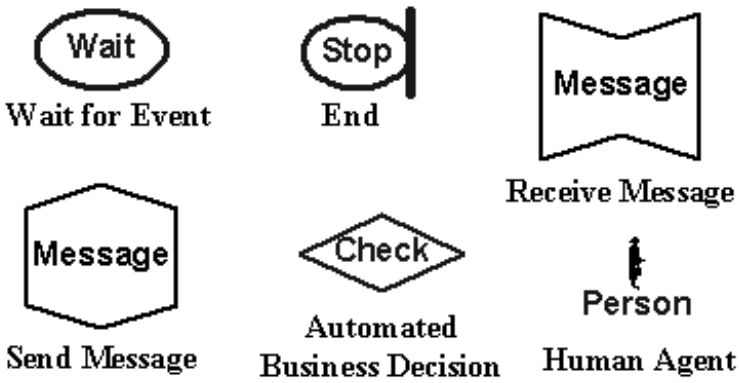

Fig. 4. Symbols used in BML

A basic characteristic of a BML diagram is that it is designed from one actor's perspective; we will call this actor the *base actor*. The base actor sends messages to, and receives messages from other actors. Typically, the base actor is the organization for whom an e-commerce system is to be built.

We now introduce two process patterns in the form of BML diagrams that correspond to the action-workflow loop. We need two distinct process patterns due to the fact that a BML diagram is designed from one actor's perspective. We need one process pattern for the case where the base actor is the requesting actor for value object in a value transfer and another process pattern when the base actor is the supplying actor for value object in a value transfer. The first pattern is called an incoming diagram and the second an outgoing diagram.

Incoming diagram (basic form)

An incoming diagram models a situation where the base actor receives a value object from another actor, see Fig. 5.

- The first step is a directive from the base actor to another actor, called the originator, asking for some value object. (Send Message "dir(VT)", where "VT" stands for value transfer, and "dir" for directive speech act.)
- The second step is a reply from the originator. The reply has to be interpreted and can be either a rejection or a commitment to fulfil the request. (Receive Message "Reply" followed by either Send Message "Rejection" or Send Message "com(VT)", where "com" stands for commissive speech act.)
- In the third step, the directive is fulfilled and the originator declares that this is the case. (Receive Message "decl(VT)", where "decl" stands for declarative speech act.)
- Finally, the base actor acknowledges that her original directive has been fulfilled. (Send Message "Ack(VT)", where "Ack" stands for an acknowledgement.)

In these basic BML templates (basic forms) that we are proposing, there are some explicit positions where inter diagram communications are possible. Sending out positions are named as OUT ports while receiving in positions are named as IN ports. Depending on the number of interactions and required control among them, these ports are completed by BML message symbols visualizing the communication between different diagrams building up a model.

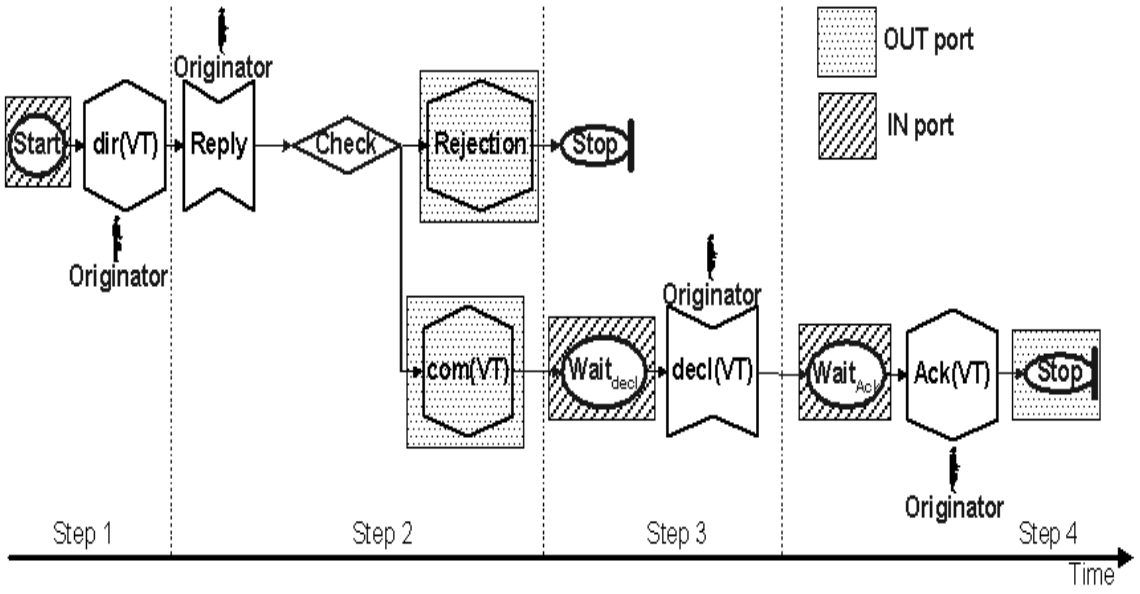

Fig. 5. Incoming Diagram with marked IN and OUT ports

Note, that here we model only the simplest possible version of the action-workflow loop and omit, for example, negotiations, counter offers, and breakdowns. Extensions to cover these and other cases are left for further work.

Outgoing diagram (basic form)

An outgoing diagram models a situation where the base actor supplies another actor with a value object, see Fig. 6. In this case, we follow the suggestion by James Taylor in [Taylor98] and introduce an additional qualification step, where the base actor acquires the means required for carrying out the requested action. e.g. Some material needed to produce and deliver a product.

- The first step is a directive from an actor, called the recipient, to the base actor asking for some value object. (Receive Message "dir(VT)".)
- The second step is the additional qualification step. It consists of one or more requests to other actors to supply the base actor with the means it needs. The step also includes the responses from these actors. (Send Message "Request" followed by Receive Message "Reply".)
- In the third step, the responses are evaluated and the base actor either rejects the directive or commits to fulfil it. (either Send Message "Rejection" or Send Message "com(VT)".)
- In the fourth step, the directive is fulfilled and the base actor declares that this is the case. (Send Message "decl(VT)".)
- Finally, the recipient acknowledges that her original directive has been fulfilled. (Receive Message "Ack(VT)".)

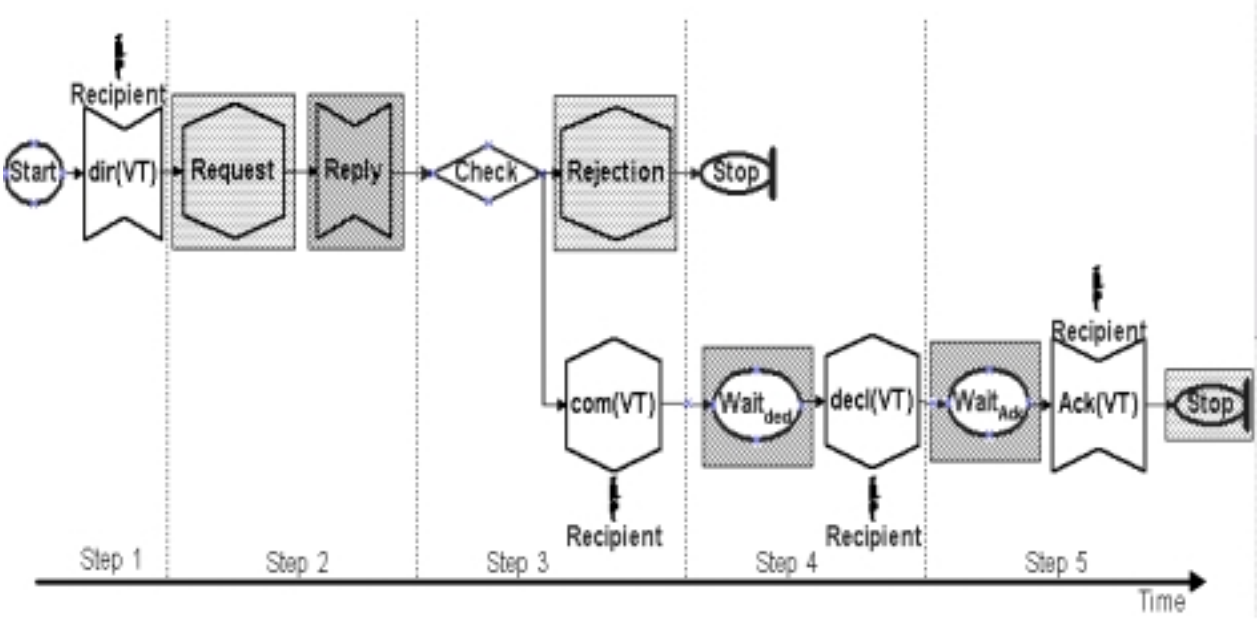

Fig. 6. Outgoing Diagram with marked IN and OUT ports

Note that there is an asymmetry between the incoming and the outgoing diagram. The reason being that the qualification step is relevant only when the base actor has to supply a value object. Also note that we have only introduced the basic forms of incoming and outgoing diagrams; they may be extended with additional symbols in order to handle their communication with each other.

A process can be modeled by a set of incoming diagrams and outgoing diagrams – such a set is called a *process model*. The basic structure of the diagrams in a process model can be derived simply from a business model. However, the communication among the

diagrams is not uniquely determined by the business model, but may vary depending on the requirements for the process. How to move from a business model to a process model is the main topic of the next section.

The diagrams can communicate with each other by sending and receiving messages. To specify where this can occur, we introduce the notions of IN-ports and OUT-ports.

5 A designer's assistant

In this section, we will show how a business model can be transformed and extended into a process model in a systematic way. A business model, as defined in Section 3, states what value objects are exchanged between what actors, while a process model, as defined in Section 4, shows the order of the actors' activities in the form of communicative acts. Moving from a business model to a process model is not a trivial task but requires a large number of design decisions. In order to support a designer in this task, we propose an automated designer's assistant that guides the designer through the task by means of a sequence of questions. This sequence can be divided into four phases, see Fig. 7.

Phase 1. The designer builds the business model, identifies the base actor, i.e. the organization from whose perspective the system is to be built, and the customer of the process to be designed.

Phase 2. The designer establishes a (partial) order between the value transfers of the business model.

Phase 3. The designer introduces the communicative acts needed to handle the value transfers and establishes a (partial) order between them.

Phase 4. From the output of phase 3, a process model is automatically derived.

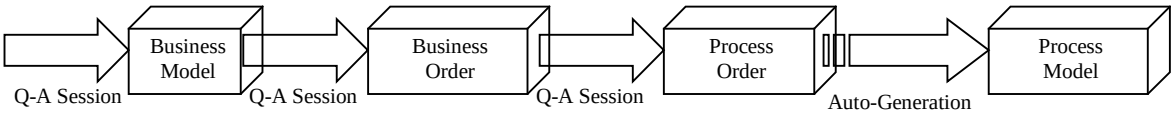

Fig. 7. Phases and Output of each phase

5.1 Phase 1 - Business Model

In the first phase, the designer builds a business model and specifies the organization for which the e-commerce system is to be developed, and the customer of the process to be designed. During this phase the answers to the following questions are obtained.

1. *Who are the actors?*
2. *Who is the base actor?*
3. *Who is the customer?*
4. *What are the value objects?*
5. *What are the value exchanges and who are the actors involved in each one?*
6. *What are the value transfers for each value exchange?*

The questions guide the designer through the task and prompt her to provide names for actors, value objects, value transfers, and value exchanges. An example of a set of answers to these questions is given in Fig. 8 and the resulting business model is shown in Fig. 2.

---

1. Who are the actors?

Customer
Retailer
Importer

2. Who is the base actor?

	Customer
X	Retailer
	Importer

3. Who is the customer?

X	Customer
	Importer

4. What are the value-objects (VO)?

Car
Money

5. What value exchanges (VE) are there, and who are the actors involved in each value exchange?

Value Exchange	Actor	Actor
Purchase	Customer	Retailer
Import	Retailer	Importer

6. What are the value transfers (VT) of each value exchange?

Value Exchange	Value Transfer	Originator	Recipient
Purchase	CarToCustomer	Retailer	Customer
	PaymentFromCustomer	Customer	Retailer
Import	CarFromImporter	Importer	Retailer
	PaymentToImporter	Retailer	Importer

Fig. 8. Questions & Answers for the Example in Phase 1

5.2 Phase 2 - Business Order

In this phase, the designer starts to construct an order between the activities of the process. First, the designer takes into account only value transfers while disregarding the communicative acts that co-ordinate the process. By considering only the order of the value transfers in this phase, the designer can concentrate on the main business logic and postpone until later more detailed design decisions about the coordination of communicative acts.
The designer first has to decide whether a value transfer must or can be divided into parts; such a part is called a *value transfer part*. A typical example is a payment (one value transfer), which may be divided into one down payment and one final payment (two value transfer parts). Another example is the delivery of goods that may be divided into several parts.
The first question in this phase is:

7. What are the value transfer parts of each value transfer?

After having identified and named the value transfer parts, the designer is prompted to order them by determining the dependencies that exist between them. In an e-commerce context, we identify two main types of dependencies: trust dependencies and flow dependencies.

A *trust dependency* occurs between two value transfer parts within the same value exchange, e.g. that a product must be paid before it can be delivered. A trust dependency expresses the level of trust between the actors involved in a value exchange, e.g. requesting a down payment expresses low trust.

A *flow dependency*, [Malone98], occurs between two value transfer parts in different value exchanges and expresses that the value object obtained by one of the value transfers is needed for the other value transfer. A simple example is that a retailer has to obtain a product from an importer before delivering it to a customer.

The second question in Phase 2 is:

8. How do you order the value transfer parts?

An example of answers to the questions is given in Fig. 9. The resulting partial order is the following (< means precedes):

{DownPaymentFromCustomer < CarToCustomer,
CarToCustomer < FinalPaymentFromCustomer,
DownPaymentFromCustomer < FinalPaymentFromCustomer,
CarFromImporter < PaymentToImporter,
CarFromImporter < CarToCustomer}

Such a partial order between value transfer parts is called a *business order*. It expresses the order between the most important activities in the process and abstracts from communicative activities.

6. What are the value-transfers parts (VTP) of each value transfer?

Car to customer	
Payment from customer	`Down payment from customer`
	`Final payment from customer`
Car from importer	
Payment to importer	

7. How do you order the value transfers and the value transfers parts?
(Add only < symbols in the matrix)

	CarToCustomer	DownPayment FromCustomer	FinalPayment FromCustomer	CarFrom Importer	PaymentTo Importer
CarToCustomer			<		
DownPayment FromCustomer	<		<		
FinalPayment FromCustomer					
CarFrom Importer	<				<
PaymentTo Importer					

The table shall be read in the following way: row header, cell, column header, e.g. the second cell on the first row gives CarToCustomer PROCEEDS DownPaymentFromCustomer.

Fig. 9. Questions & Answers for the Example in Phase 2

5.3 Phase 3 - Process Order

In phase 3, the designer will extend the business order from phase 2 by specifying dependencies between communicative acts. A starting point for this task is that for each value transfer part, there will be one action-workflow loop (modeled by an incoming or outgoing diagram in phase 4). The designer has to determine the interactions between the loops given by all the value transfer parts. The designer's assistant will support this task through a number of questions. The intuition behind several of these questions is, roughly expressed, the following: Before an actor does something of value to another actor, it will check whether that actor has deserved it. By doing "something of value to another actor" is meant to carry out a value transfer, to commit to carry out a value transfer, or to initiate the acquisition of means needed to carry out a value transfer. The expression "check whether that actor has deserved it" has to do with the fact that a value transfer from an actor A to an actor B always is accompanied by another value transfer from B to A; recall that these two value transfers together constitute one value exchange. The expression states that before actor A is prepared to carry out its value transfer (or some preparation to it) to B, it will check that B has done its corresponding value transfer (or some preparation). Note that this check will be done only if the business order so prescribes. Furthermore, there are questions for ensuring that all required means for carrying out a value transfer have been obtained.

In order to formulate the questions, we need to distinguish between an incoming value transfer part (VTP), where the base actor receives a value object, and an outgoing VTP, where the base actor supplies a value object.

If In is an incoming VTP and Out an outgoing VTP within the same value exchange, and In < Out in the business order, ask:

9a. Do you require that In be performed before you commit to perform Out?

9b. Do you require a commitment for In before you commit to perform Out?

If In is an incoming VTP and Out an outgoing VTP within the same value exchange, and Means is an incoming VTP in another value exchange, and In < Out, and Means < Out in the business order, ask:

10a. Do you require that Means be performed before you commit to perform Out?

10b. Do you require a commitment for Means before you commit to perform Out?

10c. Do you require that In be performed before you request Means?

An example of answers to these questions is given in Fig. 10. These answers will result in an extension to the business order from phase 2, which also includes ordering between communicative acts. Such an order is called a *process order*. In this case, we arrive at a process order PO:

PO = BO ∪ {Down payment from customer < com(Car to customer),
com(Final payment from customer < com(Car to customer),
Car from importer < com(Payment to importer),
com(Car from importer) < com(Car to customer)
Down payment from customer < dir(Car from importer)}

BO is the business order derived in phase 2 for the example.

9a. Do you require DownPaymentFromCustomer be performed before you commit to perform CarToCustomer?

X	Yes
	No

It is unnecessary to ask 9b as the answer to 9a already implies it.

10a. Do you require that CarFromImporter be performed before you commit to perform CarToCustomer?

	Yes
X	No

10b. Do you require a commitment for CarFromImporter before you commit to perform CarToCustomer?

X	Yes
	No

10c. Do you require DownPaymentFromCustomer be performed before you request CarFromImporter?

	Yes
X	No

Fig. 10. Questions & Answers for the Example in Phase 3

5.4 Phase 4 - Mapping Process Order to BML Process Model

After completing phase 3, the designer ended up with a set of ordered communication acts. For each inequality in this partial order, we have defined a rule that completes inter-diagram communication by connecting IN-port of one diagram with OUT-ports of another diagram.

Two operation templates have been associated in expressing the mapping from process order to complete process model as shown below;

- Add(<Source/Destination DIAGRAM>,{<Port>,<DIAGRAM that port belongs>})
- Remove(<Source/Destination DIAGRAM>,{<Port>,<DIAGRAM that port belongs>})

"add (A, {B, C})" means there should be a state B in diagram C that has interaction (or communication) with diagram A and "remove(A, {B, C})" means the diagram A at the state B of the diagram C is removed. Basically "remove" operation is used to reduce the redundancy of inter diagram communications. Also we have used suffixes "i" and "j" such that "$DIAGRAM_i$" and "$DIGRAM_j$" correspond to value transfer parts "VTPi" and "VTPj" respectively.

Rules to map Process Order to BML Process Model

i. $\forall$ $VTP_i < VTP_j$ in the Business Order;
   a) Add($DIAGRAM_j$, {Stop, $DIAGRAM_i$})
   b) Add($DIAGRAM_i$, {$Wait_{decl}$, $DIAGRAM_j$})

This is the rule that delays the delivery of a value transfer part until the completion of another, i.e. successful completion of VTP_i has to communicate at wait state prior to decl(VTP_j).

ii. $\forall$ com(VTP_i) < com(VTP_j) in the Process Order:
   a) Add($DIAGRAM_j$, {com(VTP_i), $DIAGRAM_i$})
   b) Add($DIAGRAM_i$, {Reply, $DIAGRAM_j$})

The rule ii. is to get the promise for a value transfer part before promising another, i.e. com(VTP_i) has to communicate with "Reply" which receives messages prior to promise VTP_j (com(VTP_j)) out.

iii. $\forall$ decl(VTP_i) < com(VTP_j) in the Process Order
   a) Add($DIAGRAM_j$, {Stop, $DIAGRAM_i$})
   b) Add($DIAGRAM_i$, {Reply, $DIAGRAM_j$})

The rule iii. is stronger than rule ii and requires delivery of a value transfer part prior to promise our another, i.e. successful completion of VTP_i has to communicate at "Reply" which receives messages prior to promise VTP_j.

iv. $\forall$ decl(VTP_i) < dir(VTP_j) in the Process Order
   a) Add($DIAGRAM_j$, {Stop, $DIAGRAM_i$})
   b) Add($DIAGRAM_i$, {Start, $DIAGRAM_j$})

The rule iv. is the strongest compared to rules ii. and iii. that requires delivery of a value transfer part prior to requesting another, i.e. successful completion of VTP_i has to communicate at "Start" which enacts the $DIGRAM_j$.

v. $\forall$ $DIAGRAM_j$
   if ($DIAGRAM_i$, {Reply, $DIAGRAM_j$})$\wedge$ [$\neg$($DIAGRAM_i$, {Request, $DIAGRAM_j$})] then;
   Add($DIAGRAM_i$, {Request, $DIAGRAM_j$})

Though the reverse of the rule v.) can also be defined but according to the our current questions in designer's assistant and cases studied can not find any applicability. The reverse rule can be defined as;

vi. $\forall$ $DIAGRAM_j$
if ($DIAGRAM_i$, {Request, $DIAGRAM_j$}) $\wedge$ [$\neg$($DIAGRAM_i$, {Reply, $DIAGRAM_j$})] then;
Add($DIAGRAM_i$, {Reply, $DIAGRAM_j$})

The rule v. and vi. are to ensure the completeness of the final process model. It basically completes missing inter diagram communication cases such as requesting without replying and replying without requesting.

vii. $\forall$ $DIAGRAM_j$
if ($DIAGRAM_j$, {<OutPort>, $DIAGRAM_i$}) $\wedge$ ($DIAGRAM_j$, {<InPort>, $DIAGRAM_i$}) $\wedge$ [$\neg$($DIAGRAM_i$, {<InPort>, $DIAGRAM_j$})] then;
if [$\neg$(<Any>, {Start, $DIAGRAM_j$})] then
Add($DIAGRAM_i$, {<Start>, $DIAGRAM_j$})
Else
Remove($DIAGRAM_j$, {Request, $DIAGRAM_i$})

The rule vii. is to remove any redundant inter diagram communications and to invoke inter diagram communication with not enacted processes.

The initial BML model can be generated by adding one customized basic diagram for each VTP identified at the phase 2. Addition of one diagram for each VTP ends with the initial BML model without inter-diagram communication as in Fig 11.

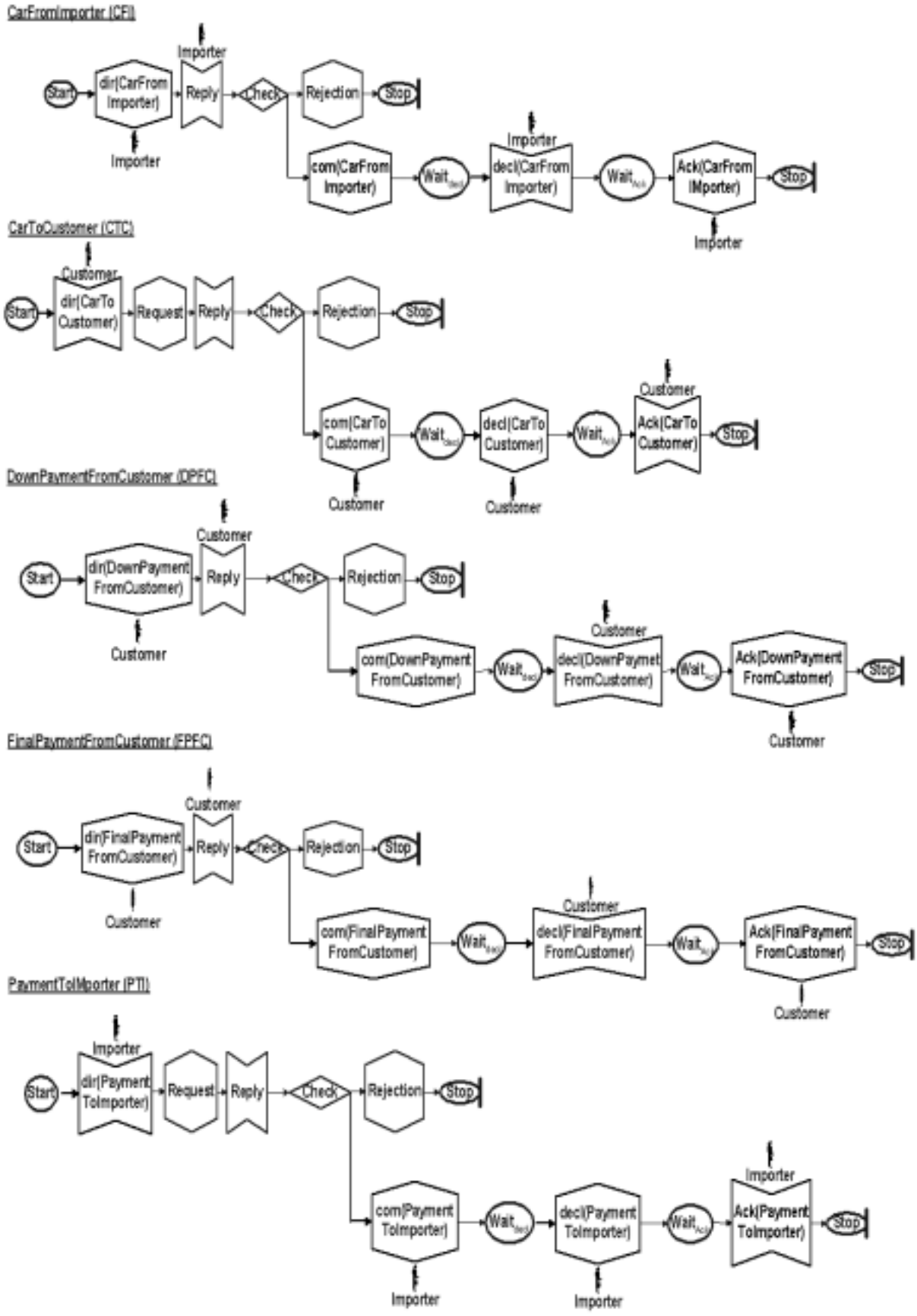

Fig. 11. Initial BML model without inter-diagram communication

Now, by following the mapping rules for the all inequalities that can be found in the process order (shown below) we can generate the inter-diagram communication to complete the BML process model. The important thing to notice here is we are placing only symbols to the initial BML process model without attempting any structural modifications to any basic diagrams composing the final process model.

Process Order = {DownPaymentFromCustomer < CarToCustomer,
CarToCustomer < FinalPaymentFromCustomer,
DownPaymentFromCustomer < FinalPaymentFromCustomer,
CarFromImporter < PaymentToImporter,
CarFromImporter < CarToCustomer}
$\cup$
{Down payment from customer < com(Car to customer),
com(Final payment from customer < com(Car to customer),
Car from importer < com(Payment to importer),
com(Car from importer) < com(Car to customer)
Down payment from customer < dir(Car from importer)}

For an example, application of the rule i.) in phase-4, for the first inequality, DownPaymentFromCustomer < CarToCustomer of PO, requires to perform the following two operations on the model;

Add(CarToCustomer, {Stop, DownPaymentFromCustomer})
Add(DownPaymentFromCustomer, {Wait$_{decl}$, CarToCustomer})

Completion of these operations on the initial BML model results in a modified version with an inter-diagram communication as shown in the Fig. 12.

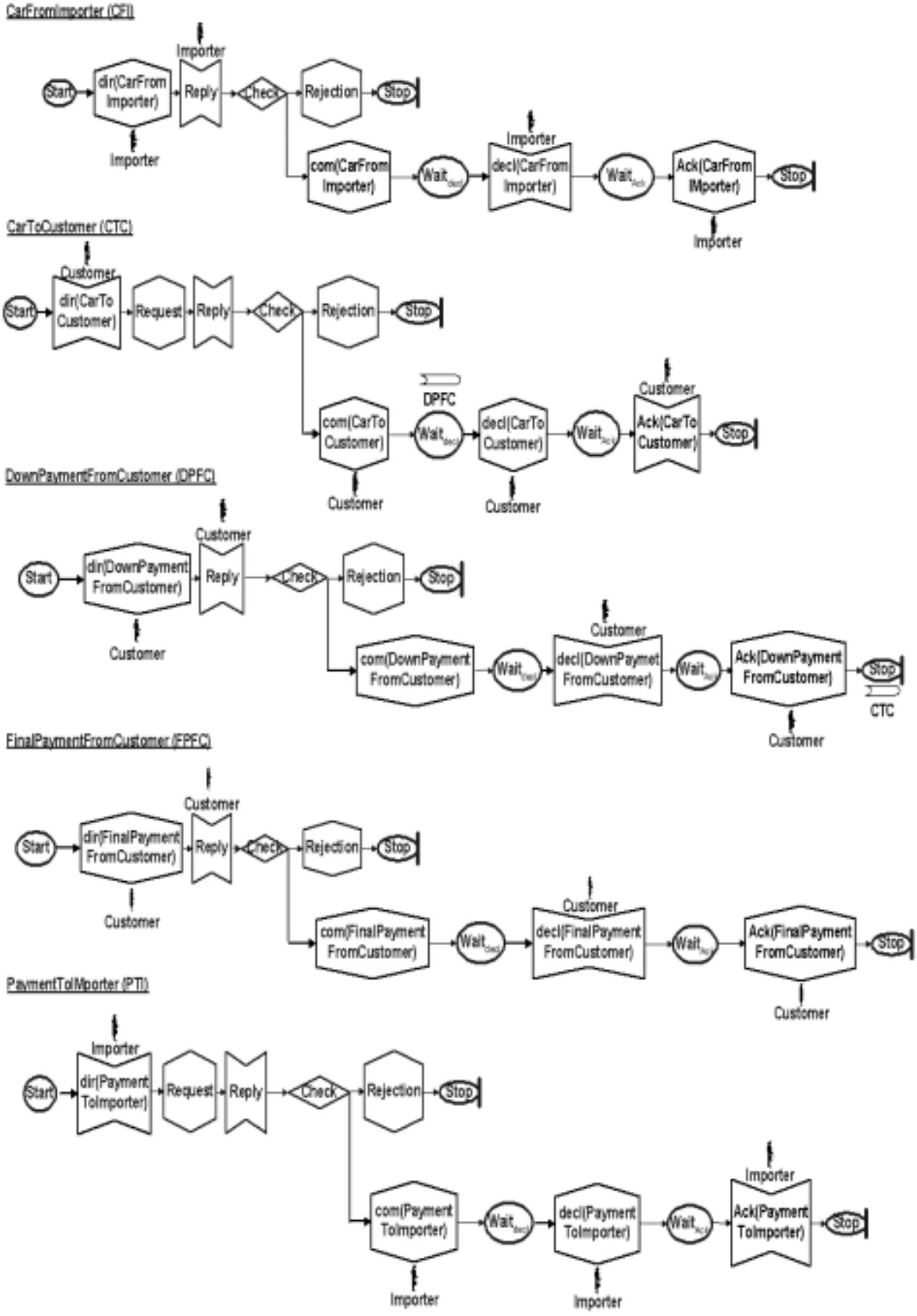

Fig. 12. First Intermediate BML model with a inter-diagram communication

Similarly, following the relevant rules for all the inequalities that can be found in the PO on the successive versions of BML models, the final BML model (see Fig 13.) can be reached automatically.

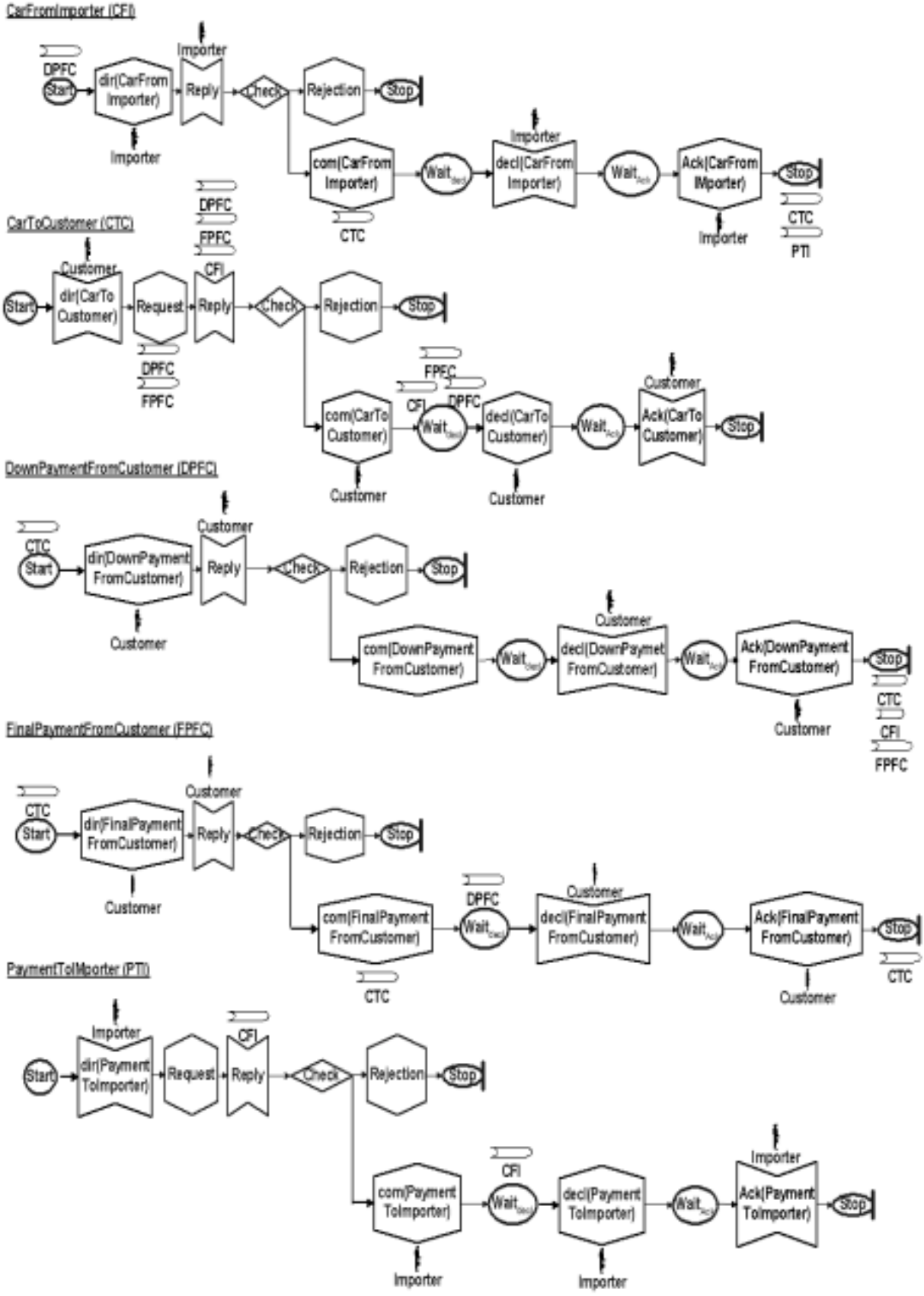

Fig. 13. Final BML model with inter-diagram communications.

6 Concluding Remarks

In the introduction, we identified three problems that hinder an effective use of the Language Action approach. In this section, we will show how the modeling techniques and guidelines introduced in the paper can address these problems, and we will suggest directions for further research.

1. Using the Language Action approach encourages a low-level perspective.
We suggest that the design of an e-commerce process be preceded by the design of a business model that focuses on actors and their exchange of value objects. A business model is a natural starting point for discussions with users and managers. When the business model has been designed, it is successively transformed and extended into a process model based on Language Action notions. In this way, the designer's assistant helps the designer to investigate a large number of possible design alternatives before committing to one of them. Furthermore, it is also possible to move backwards and from a process model track the business objectives that motivated its design.

2. The notions and terminology of the Language Action approach are unfamiliar.
We propose an automated designer's assistant that guides the designer through the task by means of a sequence of questions that use only terminology familiar to the ordinary user or manager. We have outlined the appearance of these questions but much work remains in order to make the questions easily understandable. Furthermore, a graphical interface showing partial models would improve the interaction with the designer. Another topic for future work is to identify high-level concepts in which the questions can be formulated. Examples of such concepts are the trust and flow dependencies introduced in Section 5.3.

3. There is a large distance between Language Action models and executable systems.
We suggest the use of communicating state machines, in the form of the executable language BML, for modeling processes. Thus, the specified process models are executable. Another advantage of using communicating state machines is that each state machine corresponds to an Action Workflow loop, which makes it easy to understand. Further work is needed here to specify the form of the contents of the messages sent between the state machines.

In this paper, we have only covered the simplest form of a process. Further work is, therefore, needed to handle extensions such as negotiations, breakdowns, cancellations, etc. Furthermore, the scope of the processes could also be extended to handle additional phases in e-commerce, like contact search as in BAT.

References

[Belina91] Belina F., Hogrefe D. and Amarddeo S.: "*SDL with Applications from Protocol Specification*", Carl Hanser Verlag and Printice Hall International UK 1991.

[Dietz00] Dietz, J.L.G.; Barjis, J.: "Petri Net expressions of DEMO Process Models as a rigid foundation for Requirements Engineering", *The 2nd International Conference on Enterprise Information Systems (ICEIS'00)*, 2000.

[Goldkuhl96] Goldkuhl, G.: "Generic Business Frameworks and Action Modeling", *First International Workshop on Communications Modeling - The Language/Action Perspective*, Springer Verlag 1996.

[Gordijn00a] Gordijn J., Akkermans J. M. & Vliet J. C.: "Business Modeling is not Process Modeling", *eCOM2000 workshop, 19th International Conference on Conceptual Modeling 2000*.

[Gordijn00b] Gordijn J., Akkermans J. M. & Vliet J. C.: "What's in an Electronic Business Model? ", *Knowledge Engineering and Knowledge Management - Methods, Models, and Tools, 12th International Conference*, Springer-Verlag 2000.

[Johannesson01] Johannesson P.: "A Language/action based Approach to Information Modeling", in *Information Modeling in the New Millennium*, eds. M. Rossi and K. Siau, IDEA Publishing, 2001.

[Johannesson00] Johannesson P. and Perjons E.: "Design Principle for Application Integration", *12th Conference on Advanced Information Systems Engineering*, eds. B. Wangler and L. Bergman, Springer LNCS, 2000.

[Mora92] Raul Medina et al.: "The Action Workflow Approach to Workflow Management Technology", Proceedings of 4th Conference on Computer Supported Cooperative Work, ACM Press, 1992.

[Malone98] Malone et al.: "Towards a handbook of organizational processes", MIT eBusiness Process Handbook http://ccs.mit.edu/21c/mgtsci/index.htm

[Porter98] Porter M. E.: *"Competitive Advantages. Creating and Sustaining Superior Performance"* The Free Press 1998.

[Reijswoud99] Reijswoud V. E. & Dietz J. L. G.: "Business Process Re-design with DEMO", *Third International Workshop, The Language Action Perspective on Communication Modeling*, 1999.

[SDL] SDL Standards, http://www.sdl-forum.org/Publications/Standards.htm

[Searle69] Searle J.: *"Speech Acts - An Essay in the Philosophy of Language"*, Cambridge University Press 1969.

[Taylor00] Taylor, J. R., Groleau, C., Heaton, L. and VAN EVERY E. J.: *"The Computerization of Work : A Communication Perspective"*, Thousand Oaks CA: Sage.

[Taylor98] Taylor J.: "The Limits of Rationality in Communication Modeling – a Semiotic Reinterpretation of the Concept of "Speech Act"", *Third International Workshop, The Language Action Perspective on Communication Modeling*, eds. G. Goldkuhl et.al. 1998

[Viewlocity] Viewlocity, http://www.viewlocity.com/solutions/frame_index.html

[Weigand98] Weigand H., van den Heuvel W. and Dignum F.: "Modeling Electronic Commerce Transactions – A Layered Approach", *Third International Workshop, The Language Action Perspective on Communication Modeling*, eds. G. Goldkuhl et.al. 1998.

[Winograd86] Winograd, T. & Flores, F.: *"Understanding Computers and Cognition: A New Foundation for Design"*, Ablex, Norwood, N.J 1986.