Modelling the interaction of distributed systems as protocols

Johannes Reich
johannes.reich@sophoscape.org[*]

Abstract

In descriptions of loosely coupled process-like interactions of computational systems and especially in the literature on electronic business processes, the protocol notion is often used only informally. However, its complete expressive power becomes effective only with a precise formalization. Based on Holzmann's protocol concept, a formal protocol definition is introduced, providing an inductive definition of the protocol transition relation. The protocol transition relation describes all possible interactions between system components which are formally specified as nondeterministic extended finite input output automata.

The formal approach is illustrated by means of a buying selling business interaction. Additionally, it is used to illustrate some semantic shortcomings of the "transaction pattern" approach, which tries to partition process-like interactions into one- and two-way interactions.

In the discussion, the "loose" aspect of a protocol based interaction is treated. Motivated by the non-functional system relation defined by a protocol, it is proposed to classify the exchanged information between systems according to their logical relation in documents for (nondeterministic) protocols and I/O parameters for remote function calls.

1 Introduction

In the last couple of years, the scientific interest in process-like interactions of computer systems has taken an unprecedented upsurge. A wide variety of different approaches and techniques have been developed: Among others are Petri nets [e.g. 21, 6], event-driven process chains [e.g. 14], speech acts/transaction pattern [e.g. 1, 27, 22], algebraic process descriptions [e.g. 18, 17, 10], distributed algorithms [e.g. 15, 16], abstract state machines [e.g. 9, 4], persistent Turing machines/interactive transition systems [29, 8], extended I/O automata [e.g. 15, 11], agents [e.g. 12], dedicated process description languages [e.g. 2, 13], and finally protocols [e.g. 24, 31, 11].

Petri nets had been introduced by C.A. Petri [21] to describe asynchronous information flow in analogy to physical flow, determined by conservation laws. Although his basic assumption turned out to be false as the essence of information is its fugacity, Petri nets have been studied and extended in great detail [e.g. 6, for an overview]. However, in [11, p.185] Holzmann says that "Petri Nets do not give us an advantage in the study of protocol design and validation problems".

The attempt to describe business processes as so called "event-driven process chains" [14] has been proven to be difficult to formalize due to semantic ambiguities [28].

Another attempt to describe process-like interactions on a business level currently favoured by UN/CEFACT [e.g. 27] and adapted by industry consortia like RosettaNet [22] is based on the notion of "transaction patterns". This attempt is conceptually based on Searle's theory of speech acts [1] and partitions each process into one or two way interactions. However, as I will show with the extended automata model, the semantics of these "transaction patterns" cannot be defined exclusively with respect to the referred one- or two-way interactions as intended.

Quite influential have been the algebraic approaches to characterized processes [e.g. 18, 17, 10]. In a process algebra, processes are represented by algebraic terms subject to a set of relations. However, depending on the set of relations, the possible mathematical structures fulfilling the relations are only more or less restricted, but usually not uniquely defined. Thus, an algebraic representation captures "process" semantics only to the extend as it guided the designers of the relations to express his intended semantics formally. As long as no concrete structure in a model theoretic sense is universally accepted as characterizing a process, it could well be, that someone comes up with a process structure, which is not covered by any current process algebra. Also, some concrete structure might fulfill such an

[*]published as J. Reich (2008). Modelling the interaction of distributed systems as protocols. *Proceedings of the MCETECH 2008*, pp. 16-24.

axiomatizing, but nevertheless everybody might agree, that this is definitively not a process structure.

Hoare [10, p.2] for example uses the term 'process' "to stand for the behaviour pattern of an object, insofar as it can be described in terms of the limited set of events selected as its alphabet". However, Hoare use the name of the events as labels for arrows between nodes, relating his events more to actions in the sense of state transitions, instead of the relevant I/O. Milner [17] bases his π-calculus on the concept of named channels or specified addresses, trying to treat data access and communication as the same thing. Since names take in some sense the role of send and receive actions, the notion of names and co-names evolve. Again, Milner takes the view, that a message always consists of an operator and possibly of some value [5], with corresponding consequences for the design of process languages, directly or indirectly influenced by his theory [e.g. 30, 2, 13]. Thus, in contrast to the advocated viewpoint of this article, at least these well known process calculi of Hoare and Milner rest on the assumption that interaction should be described by named operations.

The interesting theory of distributed algorithms [e.g. 16] tries to approach the description of distributed systems with algorithmic means, enhanced by constructs to encompass the parallel nature of distributed computing.

What is the reason to propose yet another formalism to describe the interactions of distributed systems? First of all, the protocol concept as a description of (informational) system interaction is not new. According to Holzmann [e.g. 11], R.A. Scantlebury and K.A. Bartlett [24] have introduced the term "protocol" in 1967 to denote a process like data exchange. Indeed, many of the cited authors use the term "protocol" informally to denote the interaction of systems, but do not give or relate to a formal definition.

However, being rule-based interaction specifications, protocols may be much more than just "another formalism". In the well known OSI Reference Model [31], functional interfaces describe a hierarchical, directional cooperation between systems and their subsystems, creating formally identifiable software layers within a program (if applied only one-directionally), while protocols describe the interaction of systems on a single abstraction level and can therefore be attributed to a single software layer. Thus, the OSI model basic paradigm is that protocols are well suited to denote a different class of system relation than the conventional functional relations, usually described by interfaces with operations. Actually, starting with the OSI paradigm, graphics protocols like X [20] show another OSI assumption to be false, namely that GUI and application should cooperate in a functional way. Quite in contrast, due to the many imponderables GUI and application are exposed to, protocol like peer-to-peer interactions between both have been proven to be by far superior.

With protocols we are able to provide a complete set of formal rules that govern the behavior of all participants in a process-like interaction. Such a formal description is a prerequisite for any formal verification technique, comprising validation as the consistency check of the protocol specification itself as well as conformance as the equivalence of the behavior of a given protocol implementation to its formal specification.

Thus, all fields of computer science where peer-to-peer interactions play a dominate role profit enormously from formal protocol specifications. To name just two: first, the emerging field of agent based software engineering [e.g. 12], which views computer systems as autonomous systems, interacting in a multitude of ways with other systems on a single abstraction level; and second, enterprise software where the execution of electronic business interactions on a peer-to-peer level becomes more and more prominent such that large software vendors already see it as strategically important to position their solutions as a operating system for business processes or business process platform [e.g. 23].

Since protocols explicitly describe the interaction of systems, they have to rely on system descriptions from an - external - interaction point of view. Holzmann uses extended I/O automata as means to describe the participating systems for this purpose. These automata are similar to Gurevich's [9] concept of abstract state machines or the interactive transition systems of Goldin, Smolka and Wegener [8]. The differences will be discussed later on.

In the following, based on a system description of extended I/O automata, a formal protocol definition will be given, where the definition of the protocol transition relation is inductively constructed. Subsequently, the formal approach is illustrated by means of a buying selling business collaboration. Additionally, as an example of the explanatory power of the protocol notion, the "transaction pattern" approach to describe process-like interactions with isolated one- and two-way interactions is shown not to adequately capture the interaction semantics as is described by protocols. In the discussion, the nondeterministic aspects of a protocol based system relation and the fact, that a protocol specification usually only relates to a (small) subset of a system's states are subsumed under the headline of "loose coupling". Also, it is argued that the semantics of the entity "document", which seems to play a principle part in conventional business processes, stem from its usage within process-like (business) interactions. Thus, it is not to be expected that electronic based process-like business interactions following the same model can do without such entities.

2 Protocols

Our starting point is that of agent based software engineering [12] were information systems interact with many other information systems on a single layer of abstraction or peer-to-peer. To describe a particular interaction, we therefore have to focus our system view to those parts, which are necessary for the description of this particular interaction, in other words a system projection. Within the protocol concept, such an interaction view of a system is specified by a nondeterministic extended finite I/O automaton (NEFIOA) [e.g. 11, 3, 15].

Definition 2.1: A *nondeterministic, extended finite I/O automaton* is defined by $\mathcal{A} = (Q, I, O, q_0, \Delta, F, C)$, with Q is the set of states, I and O are the input and output alphabets, q_0 is the initial state, $\Delta \subseteq Q \times Q \times I \cup \{\epsilon\} \times O \cup \{\epsilon\} \times C$ is the transition relation (ϵ is the empty word), F is the set of final states, and C is the set of conditions. Its components may also be denoted by the system symbol as subscript.

For practical purposes, the states as well as the characters of the I/O-alphabets are allowed to have an interior structure and thus represent classes of structured states and characters (which are nothing else than strings structured according to a formal grammar). As a consequence of the possible internal structure, the transition relation of a NEFIOA has to be extended by conditions c, which relate to the interior structure of the states and characters.

The semantics of the transitions are quite interesting. A transition of a NEFIOA specifies that beginning with a start state and either triggered by an input or spontaneously (which formally is indicated by the empty word ϵ), a target state will be reached and some output may be done if a certain condition is fulfilled. This does not specify any particular function in an abstract sense, but only a relation or rule which could by fulfilled by an arbitrary number of functions. In the realm of algebraic specifications, this is called loose semantics [e.g. 19, chapter 3]. This is the main difference to approaches like abstract state machines [e.g. 9, 4] or interactive transition systems [e.g. 29, 8], where each transition is supposed to represent a unique computable function.

For further simplification, we focus our following considerations on pair protocols, where two NEFIOAs $\mathcal{A}$ and $\mathcal{B}$ are connected to each other by two reciprocal transfer functions, each provided by a channel. An extension to protocols with arbitrary many participants or more than two transfer functions should be obvious.

Definition 2.2: Let $\mathcal{A}$ and $\mathcal{B}$ be two NEFIOAs, whose I/O-alphabets obey the condition that there are two mappings $f_{\mathcal{AB}} : O_{\mathcal{A}} \rightarrow I_{\mathcal{B}}$ and and $f_{\mathcal{BA}} : O_{\mathcal{B}} \rightarrow I_{\mathcal{A}}$, allowing the coupling. A *(pair) protocol* between $\mathcal{A}$ and $\mathcal{B}$ is given by $\mathcal{P} = (S_{\mathcal{P}} Q_{\mathcal{P}}, q_{0\mathcal{P}}, T_{\mathcal{P}}, \Sigma_{\mathcal{P}}, \Delta_{\mathcal{P}}, F_{\mathcal{P}})$, with $S_{\mathcal{P}} = \{\mathcal{A}, \mathcal{B}\}$ is the set of systems, also called participants, $Q_{\mathcal{P}} = Q_{\mathcal{A}} \times Q_{\mathcal{B}}$ is the set of (structured) protocol states, $q_{0\mathcal{P}} = (q_{0\mathcal{A}}, q_{0\mathcal{B}})$ is the initial protocol state , $T_{\mathcal{P}} = \{f_{\mathcal{AB}}, f_{\mathcal{BA}}\}$ is the set of transfer functions, mapping the output of one onto the input of the other NEFIOA, $\Sigma_{\mathcal{P}} = I_{\mathcal{A}} \cup I_{\mathcal{B}} \cup O_{\mathcal{A}} \cup O_{\mathcal{B}}$ is the set of (structured) characters, used within the protocol, $\Delta_{\mathcal{P}} \subseteq Q_{\mathcal{P}} \times Q_{\mathcal{P}} \times \Sigma_{\mathcal{P}} \cup \{\epsilon\} \times \Sigma_{\mathcal{P}} \cup \{\epsilon\} \times C \times S$ is the protocol transition relation, and $F_{\mathcal{P}} = F_{\mathcal{A}} \times F_{\mathcal{B}}$ is the set of common final states.

The definition of the protocol transition relation describes how the coupled system transits from a state to a next state, which is always achieved by either automaton $\mathcal{A}$ or $\mathcal{B}$. The transitions t of the coupled system are therefore described by the starting state $(p_{\mathcal{A}}, p_{\mathcal{B}})$ and target state $(q_{\mathcal{A}}, q_{\mathcal{B}})$ of both automata, the input and output characters i, o and the condition c of the particular automaton transition and the automaton aut itself to which transition this protocol transition relates. Thus, we have $t = ((p_{\mathcal{A}}, p_{\mathcal{B}}), (q_{\mathcal{A}}, q_{\mathcal{B}}), i, o, c, aut)$.

The elements of the transition relation $t \in \Delta_P$ are determined inductively. First, all spontaneous transitions, which start from the automata initial states are part of the relation. Then, assuming that a given transition t belongs to the transition relation, other elements t' of the transition relation can be constructed, depending on the automaton to which the transition is related to and the emptiness or nonemptiness of the output.

i. (a) Assuming that $q \in Q_{\mathcal{A}}$ exists, such that $(q_{0\mathcal{A}}, q, \epsilon, o, c) \in \Delta_{\mathcal{A}}$,
then $t = ((q_{0\mathcal{A}}, q_{0\mathcal{B}}), (q, q_{0\mathcal{B}}), \epsilon, o, c, \mathcal{A}) \in \Delta_{\mathcal{P}}$
   (b) Assuming that $q \in Q_{\mathcal{B}}$ exists, such that $(q_{0\mathcal{B}}, q, \epsilon, o, c) \in \Delta_{\mathcal{B}}$,
then $t = ((q_{0\mathcal{A}}, q_{0\mathcal{B}}), (q_{0\mathcal{A}}, q), \epsilon, o, c, \mathcal{B}) \in \Delta_{\mathcal{P}}$

ii.1 Assuming that $t = ((p_{\mathcal{A}}, p_{\mathcal{B}}), (p_{\mathcal{A}}, q_{\mathcal{B}}), i, o, c, \mathcal{B}) \in \Delta_{\mathcal{P}}$, $i \in I_{\mathcal{B}} \cup \{\epsilon\}$, $o \in O_{\mathcal{B}}$ and
there exits $q' \in Q_{\mathcal{A}}$ such that $(p_{\mathcal{A}}, q', f_{\mathcal{BA}}(o), o', c') \in \Delta_{\mathcal{A}}$,
then $t' = ((p_{\mathcal{A}}, q_{\mathcal{B}}), (q', q_{\mathcal{B}}), f_{\mathcal{BA}}(o), o', c', \mathcal{A}) \in \Delta_{\mathcal{P}}$

ii.2 Assuming that $t = ((p_{\mathcal{A}}, p_{\mathcal{B}}), (q_{\mathcal{A}}, p_{\mathcal{B}}), i, o, c, \mathcal{A}) \in \Delta_{\mathcal{P}}$, $i \in I_{\mathcal{A}} \cup \{\epsilon\}$, $o \in O_{\mathcal{A}}$ and
there exists $q' \in Q_{\mathcal{B}}$ such that $(q_{\mathcal{B}}, q', f_{\mathcal{AB}}(o), o', c') \in \Delta_{\mathcal{B}}$,
then $t' = ((p_{\mathcal{A}}, q_{\mathcal{B}}), (q_{\mathcal{A}}, q'), f_{\mathcal{AB}}(o), o', c', \mathcal{B}) \in \Delta_{\mathcal{P}}$

ii.3 Assuming that $t = ((p_{\mathcal{A}}, p_{\mathcal{B}}), (q_{\mathcal{A}}, q_{\mathcal{B}}), i, o, c, aut) \in \Delta_{\mathcal{P}}$ with $i \in I_{aut} \cup \{\epsilon\}$, $o \in O_{aut} \cup \{\epsilon\}$, then
   (a) if $q' \in Q_{\mathcal{A}}$ exists, such that $(q_{\mathcal{A}}, q', \epsilon, o', c') \in \Delta_{\mathcal{A}}$,
then $t' = ((q_{\mathcal{A}}, q_{\mathcal{B}}), (q', q_{\mathcal{B}}), \epsilon, o', c', \mathcal{A}) \in \Delta_{\mathcal{P}}$,

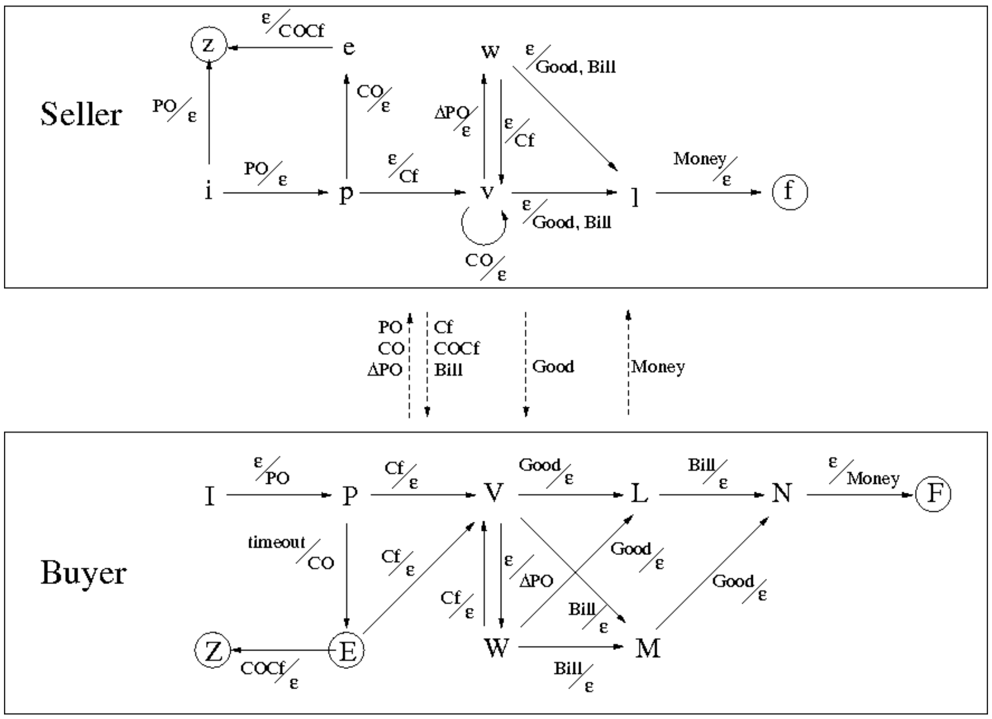

Figure 1. State diagram of a simple buying selling interaction. Each partner is described as a NEFIOA. Each transition displays the possible input and output. ϵ represents the empty input or output. Seller may send a purchase order (PO), a purchase order update (ΔPO), a cancel order (CO) and the money. Buyer may send a confirmation (CF), a cancel order confirmation (COCf), the bill (Bill) and the Good. For a detailed process description, and the additional relevant conditions, see text.

(b) if $q' \in Q_{\mathcal{B}}$ exists, such that $(q_{\mathcal{B}}, q', \epsilon, o', c') \in \Delta_{\mathcal{B}}$,
then $t' = ((q_{\mathcal{A}}, q_{\mathcal{B}}), (q_{\mathcal{A}}, q'), \epsilon, o', c', \mathcal{B}) \in \Delta_{\mathcal{P}}$

To exclude deadlocks and guarantee the possibility to escape from live locks, it is additionally demanded that the transition relation satisfies the *cooperation condition*, that for each reachable state pair $(p_{\mathcal{A}}, p_{\mathcal{B}})$ there exists a finite path, leading to a final state $(q_{\mathcal{A}}, q_{\mathcal{B}}) \in F$.

The instruction to construct the protocol transition relation does not guarantee that the relation is non-empty. Especially, it follows from the definition, that it is empty if none of the systems offers a spontaneous transition from an initial state: someone has to start the interaction spontaneously.

A pair protocol describes the interaction between two systems self-contained in the sense that per definition all characters which are sent within the described interaction by one system have to be received by the other system. Thus, a self-contained protocol lacks any input or output and therefore is in this sense by itself no automaton anymore.

While protocols according to definition 2.2 assume a set of dedicated final states and therefore could be termed *goal oriented*, endless cooperation are characterized by a set of allowed common state pairs representing the achievement of a given protocol loop. Thus, in the endless case, there is some justification for saying that "the journey is the reward".

2.1 Example: Business interaction of buying and selling

A buyer named *Buyer* and a seller named *Seller* are involved in a mutual business interaction as illustrated in Fig. 1. This example broadens our protocol definition in two aspects. First, the participants communicate over 4 channels instead of two and second, seller is allowed to send out two items at once, namely the bill and the good.

Both participants start from their initial state (i, I). *Buyer* initiates the interaction by sending a purchase order (PO) and waits for the confirmation (Cf). If the confirmation does not arrive in time, a timeout triggers another transition of *Buyer* such that it cancels its purchase and issues a cancellation of its purchase order (CO).

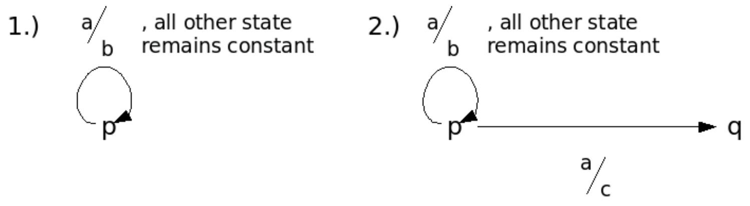
1.)
a/b
, all other state remains constant
p
2.)
a/b
, all other state remains constant
p
q
a/c

Figure 2. Part 1. shows a single transition which accepts a character ’a’ and sends another character ’b’ without changing the state. Part 2. shows an additional transition, accepting the same character ’a’ and changes the state to q while sending ’c’.

Seller receives the purchase order and decides either to sell *Buyer* as requested or not. In the latter case, the interaction is already over for it. In the former case, *Seller* attempts to organize the ordered good and sends a confirmation in case it is sure enough to be able to deliver. If this time takes too long, *Buyer* is as said before allowed to cancel the order.

As long as *Seller* has confirmed but not yet sent the good away, *Buyer* is allowed to update its purchase order (ΔPO). However, *Seller* may neglect an update if it arrives after the good has already been packed and sent away together with the bill.

Since bill and good are transported by different channels, their receiving order is undetermined. So, after having received both, order and bill, *Buyer* transfers the money.

An essential part of the interaction, but not displayed in Fig.1, are the additional conditions. The first condition is that each document has to relate to this interaction. This can be assured by a “my sign/your sign” mechanism, providing a name for a particular interaction instance. With the first PO, *Buyer* issues its sign and with the first response *Seller*, which can be a Cf or a COCf, issues its sign. All other transitions demand correct signs. Furthermore, the delivered good has to match the ordered one. For example, the delivered book “The plaque” has to match the ordered book with the ISBN 978-3499225000. If applicable, the delivered quantities may be allowed to deviate from the ordered quantities. However, the payed money has to match the price of the delivered goods.

There are several interesting things to comment on. First, the timeout is described as an ordinary event and not as an exception. Second, to decide, which of the participants has decided correctly with respect to potentially criss-crossing CO and Cf documents, both participants have to agree upon the same time. And third, in full accordance with Holzmann, who wrote in his foreword that “even the most carefully developed protocol specifications contain subtle errors unless they are subjected to a thorough error analysis”, this protocol is indeed seriously flawed. A confirmation could reach *Buyer*, which was issued too late, such that *Buyer* had correctly decided not to buy anything. But still, according to the above protocol description, this reception would result in a transition to state V and a corresponding waiting for goods and bill. Thus, another conditioned CF-receiving transition of *Buyer* from E to E is necessary. Additionally, and even worse, since the confirmation and the good are transported with different channels, possibly not guaranteeing in order transport of both, it could well be that the good arrives before the first confirmation and cannot be accepted. Such protocol errors might become quite expensive in case perishable good is delivered.

2.2 The semantics of UMM transaction patterns

As a simple application of the formal protocol model of system interactions, I would like to analyse the semantics of the UN/CEFACT’s modelling methodology (UMM) transaction pattern approach [e.g. 27, 22] to partition each process-like interaction into one- or two-way interactions. As Holzmann [11, section 1.3] explicitly refers to protocols as “languages” , realizing that “a full protocol definition, in fact looks much like a language definition” this small investigation should be quite interesting, as the transaction pattern approach was inspired by Searle’s speech act theory [1].

According to UMM, there are 6 different transaction patterns. One-way are information and notification. Two-way are request-response, request-confirm, query-response and commercial transaction. The ’query’ and ’response’ or ’request’ and ’confirm’ characteristic is supposed to denote a property of the sent data and their processing. However, in a protocol, the semantics of the transmitted information is not determined by a single, but by all accepting transitions of a given state. That is, it depends on the complete protocol transition relation and not just one single transition.

Fig. 2 shows two simple snippets of a system’s NEFIOA. The first part displays a single transition as a case for a typical query function, which accepts a character ’a’ as a question and sends another character 'b' as a response, while leaving the complete state invariant. In the second part of Fig. 2 an additional transition is introduced, leaving the semantics of the first transition invariant, but obviously not the semantics of the received character 'a'.

With spontaneous interaction, we have a simple case, which one might spontaneously qualify as a two way asynchronous interaction pattern like question-response or request-confirm as is illustrated with another NEFIOA snippet in Fig. 3. Starting from state q, a character 'a' is received, triggering a silent state changing transition. While spontaneously transiting back, a character 'b' is sent away.

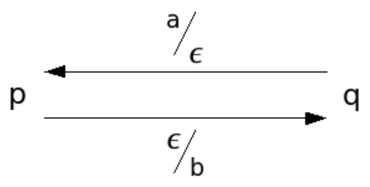

Figure 3. While holding state p, a character 'a' is triggers a silent transition to q. Starting from q, a spontaneous transition sends a character 'b'

But, just looking at Fig. 3, we do not know, whether q is really occupied in the beginning. It could well be, that state p is occupied first. As a result, the system would be in the opposite role and 'b' would become its question and 'a' would be the others' response. Additionally, the whole picture changes again, if other transitions referring to the same characters come into play.

Summarizing, we can say that UMM transaction pattern do not describe adequately process-like interaction semantics with the necessary formal decoupling between the exchanged data and their processing as protocols do.

3 Discussion

Protocols describe the rule based external coupling of information processing systems with respect to their behavior and exchanged information. Their nondeterminism is a simple consequence of the fact that the interaction view (or projection) of the systems intentionally does not provide enough information about the internal state so that the input as well the provided internal state does not suffice to determine the state transitions. However, it does not imply that the systems themselves are nondeterministic. It just means that they behave nondeterministically with respect to the interaction.

I.e. nondeterminism is not introduced to abstract from details of implementation [10, p.84] or only for convenience [16, p.257], but nondeterminism seems to be the precondition for an efficient description of process-like interactions between many different systems, where all interactions happen on the same level of abstraction. It seems highly intuitive that in such a situation, the projection of a system on one of its possibly many interactions cannot be deterministic.

3.1 The meaning of loose coupling

In general, process like interacting systems seem not to expose their functionality, except to some receive functions in the sense of a sensor like an ear or a post box. Especially, these receive functions do not make any commitment with respect to the semantics of the received information. I.e. the correctness of such a receive function does not depend on the correctness of further data processing.

In the sense of Wegener [29], protocol based interaction descriptions therefore go beyond the description of functional system relations, as they are based not on computational functions, but on abstract mathematical relations. If we agree that a protocol provides a base for describing peer-to-peer interactions or partnership, then we can say that partnership is not a functional relation.

Protocols provide a somehow loose coupling in at least two aspects: first, a protocol describes an interaction in a way that both systems can cooperate sensibly without knowing the complete state of the other system or even knowing parts of it definitively all the time. To be able to act sensibly in a protocol based interaction, it suffices to know only parts of the others' state for only certain points in the past. Nondeterminism in this sense leads to "loose coupling" in the sense of Glassman [7], who introduced this term to describe sensible interactions between systems, which relate only to some of their many states.

The second aspect seems to be that protocols don't specify the functions with which each participant might realizes the transitions completely. The actual transition functions only have to comply with the given rules and otherwise just have to be "good enough".

The flexibility of protocol based interactions, which even allows to easily describe criss-crossing information exchange, seems to come at the costs of the inductive transition relation definition. In contrast to automata, where the transition relation is usually given as a static entity, the transition relation of a protocol is a result of the interaction itself. It depends for example on the initial state of the participants. Perhaps this is one of the main reasons, why protocol design is so inherently difficult.

3.2 Logical system relations and data semantics classification: I/O parameters versus documents

The non-functional system relation described by protocols sheds new light on the question how we should sensibly talk about exchanged information. I propose that the semantic classification of the exchanged information should relate to the logical system relation of the interacting systems. That is, without knowing this relation, the exchanged information cannot be classified and it therefore probably does not make much sense from a semantic perspective to talk about "message exchange patterns" as is done in [25]. In contrast, knowing the system relation as being well defined by obeying a protocol specification, it should be possible to define compatibility relations similar to upward and downward compatibility of systems obeying functional relations.

Exchanged information are commonly termed "messages". This can lead to an important ambiguity, since the technical data structure, which is used by the channel's transport function for its technical realization, i.e. the entirety of transport envelope and exchanged information, is usually also termed "message". As a result, there is the danger to name two different entities, belonging to two different software layers, namely the process and the transport layer, with the same term.

Information which are exchanged in a functional relation imply as input of the functional mapping what will happen in the sense of a uniquely defined output. This is independent of any need of data transport like in remote function calls (or remote method invocations). Actually, the catenation of transfer $(f_{\mathcal{AB}}, f_{\mathcal{BA}})$ and processing functionality g of a remote procedure r as $r = f_{\mathcal{BA}} \circ g \circ f_{\mathcal{AB}}$ leads to disappearance of the transfer functions - except for the case, where information transfer does not work as specified, the reason for the additional exceptions in remote function calls.

Information, which are exchanged within a (nondeterministic) protocol do not - from the perspective of the sender - have this functional input character and relate to future actions only subject to some uncertainty left to the receiver. It therefore does not make sense to denote any operation with an imperative semantics in such a document as is proposed by others [e.g. 5, 2]. Within a protocol, the exchanged information first and foremost signal a state transition of the sender formulated in a way that the receiver can make its own state transition. It suggests itself to name these exchanged data "documents", since their primary purpose is to document something, namely state transitions of a sender, in a for a receiver comprehensible way.

I think that protocols do provide the semantics for the UML 2.1 collaboration diagram [26], as a UML collaboration describes a view (or projection) of a set of cooperating "classifiers". However the existence of at least two different system relation classes, namely collaborative and functional also suggests that the semantics of the UML 2.1 connector interpreted as a system relation, which is suggested by its usage in composition diagrams, is not well defined.

An interesting consequence results from the fact that the information needs of a receiver could also stem from other protocol interactions it is involved in. For example a patient, admitted to a hospital, has to provide its insurance card. For the treatment interaction between the hospital and him or her, this is unnecessary. However, because the hospital will be involved in an additional billing interaction with the patient's insurance, the information has to be provided at the admission by the patient. In an emergency case, the hospital will set its claims aside, showing the flexibility of process like interactions. It seems that one of the price tags we have to pay for the loose coupling of systems is the lacking generic, ever valid way to document a state transition of a process within a given protocol. The requirements to document a patient admission within the treatment interaction are influenced by the requirements of the billing interactions. There is not the one and only true way to say "a patient xyz is admitted" as there is no such one and only way to say to ones children that they should make their homework or to say to your partner: "I love you!" . This fact has far reaching consequences in the field of business processes. It means that first, standard documents can guaranteely support only standard processes. Second, to achieve the necessary flexibility to change business interactions, extension mechanisms for the involved documents have to be necessarily provided with respect to defined compatibility rules. Contrariwise, certain process-like interactions can be circumvented by withholding the appropriate information - the essence of data protection.

3.3 Concluding remark

Nondeterminism is the price we have to pay for achieving a certain decoupling between information systems from their partner's functionality and state. Nondeterministic interactions also imply a decoupling between transport and processing. It might even be desirable to introduce the entity "protocol" to programming languages as "interfaces" had been introduced to object oriented languages.

In summary we can say that process-like cooperation makes a virtue of the necessity not to know the state of a partner in detail and achieves a remarkable flexibility, simplicity and robustness in achieving common goals.

Acknowledgement I herewith like to wholeheartedly thank Prof. Hans-Jörg Kreowski who took quite some of his time to help me formalizing my ideas and for his invaluable comments on several prior versions of this article.

References

[1] Maria Bergholtz, Prasad Jayaweera, Paul Johannesson, and Petia Wohed. Bringing speech acts into UMM. In 1st *Int. REA Technology Workshop Copenhagen, Denmark*, April 2004. http://www.itu.dk/people/kasper/REA2004/pospapers/PrasadJayaweera.pdf, called 2007-06-05.

[2] Web services business process execution language version 2.0, 2007. http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.pdf, called 2007-04-18.

[3] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

[4] Egon Börger and Robert Stärk. *Abstract State Machines*. Springer, Berlin, Heidelberg, New York, 2003.

[5] Marco Carbone, Kohei Honda, Nobuko Yoshida, Robin Milner, Gary Brown, and Steve Ross-Talbot. A theortical basis of communication-centred concurrent programming, 2002. http://www.w3.org/2002/ws/chor/edcopies/theory/note .pdf, called 2007-07-28.

[6] H. Ehrig, W. Reisig, and G. Rozenberg. *Petri Net Technology for Communication-Based Systems. Advances in Petri Nets.: Advances in Petri Nets*. Springer, Berlin, Heidelberg, New York, 2004.

[7] R. B. Glassman. Persistence and loose coupling in living systems. *Behavioral Science*, 18:83–98, 1973.

[8] Dina Q. Goldin, Scott A. Smolka, Paul C. Attie, and Elaine L. Sonderegger. Turing machines, transition systems, and interaction. *Inf. Comput.*, 194(2):101–128, 2004. http://www.cs.umb.edu/ dqg/papers/its-8-01.ps, preliminary version, called 2007-07-28.

[9] Y. Gurevich. A New Thesis. Abstracts, American Mathematical Society, abstract 85T-68-203, August 1985.

[10] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985/2004.

[11] Gerard J. Holzmann. *Design and validation of computer protocols*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[12] N. R. Jennings. On agent-based software engineering. *Artificial Intelligence Journal*, 117(2):277–296, 2000.

[13] Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon, and Charlton Barreto. Web services choreography description language version 1.0. http://www.w3.org/TR/ws-cdl-10/, called 2007-07-28, 2005.

[14] G. Keller, M. Nüttgnes, and A.-W. Scheer. Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). *Veröffentlichung des Instituts für Wirtschaftsinformatik (IWi) der Universität des Saarlandes*, 89, 1992.

[15] N. A. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. Technical Report MIT/LCS/TR-387, 1987.

[16] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc. San Francisco, California, USA, 1996.

[17] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (parts I and II). *Information and Computation*, 100(1):1–77, 1992.

[18] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[19] John C. Mitchell. *Foundations for Programming Languages*. MIT Press, Cambridge, Massachusetts, 3 edition, 2000.

[20] Adrian Nye. *X Protocol Reference Manual for X11, Release 6*. O'Reilly Media, 4 edition, 1995.

[21] A. Petri, Carl. Fundamentals of a theory of asynchronous information flow. In *Information Processing 62, Proceedings of the 1962 IFIP Congress*, pages 386–390. North-Holland, Amsterdam, The Netherlands, 1962.

[22] *RosettaNet Implementation Framework: Core Specification, V02.00.00*, 2001. http://xml.coverpages.org/RNIF-Spec020000.pdf, called 2007-3-11.

[23] SAP Annual Report 2005. http://www.sap.com/company/investor/reports/annual report/2005/pdf/2005_SAP_Annual_Report.pdf, called 2007-09-06.

[24] R. A. Scantlebury and K. A. Bartlett. A protocol for use in the NPL data communications network. Technical Memorandum, 1967.

[25] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), 2007. http://www.w3.org/TR/soap12-part1, called 2007-09-08.

[26] Unified Modeling Language: Superstructure, version 2.1.1, 2007. http://www.omg.org/docs/formal/07-02-05.pdf, called 2007-09-06.

[27] *UN/CEFACT Modelling Methodology (UMM) User Guide, V20030922*, 2003.

[28] W.M.P. van der Aalst, J. Desel, and E. Kindler. On the semantics of EPCs. a vicious circle. In M. Nüttgens and F.J. Rump, editors, *EPK2002, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, pages 71–79. 2002.

[29] P. Wegner. Why interaction is more powerful than algorithms. *Comm. ACM*, 40(5):80–91, 1997.

[30] Web services description language (wsdl) 1.1, 2001. http://www.w3.org/TR/wsdl, called 2006-07-20.

[31] Hubert Zimmermann. OSI reference model - The ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, COM-28(4):425–432, 1980.