

E-Commerce Software: From Analysis to Design

Peter Rittgen

Institute of Business Informatics, University Koblenz-Landau, Koblenz, Germany

ABSTRACT

Early information systems were mainly built around secondary, administrative processes of the value chain (e.g. accounting). But since the internet came into use, more and more primary processes have become accessible to automation: customer acquisition, ordering, billing and, in the case of intangible goods such as software, even delivery. To facilitate this complex task, we suggest that the relevant parts of the enterprise be modeled according to the MEMO method (Multi-perspective Enterprise MOdeling). It allows for the description of an enterprise on three levels - strategy, organization and information system - and from four angles - process, structure, resources and goals. All partial models for the views are integrated via a common object-oriented core. In this framework we suggest a modeling language for the IS layer, the Event-driven Method Chain (EMC), a process-oriented language based on Event-driven Process Chains (EPCs), which we adapt to fit both the MEMO framework and the object-oriented paradigm thus making it suitable for the development of web-based applications in an object-oriented programming language. To illustrate this we use the example of a software trading company.

INTRODUCTION

Early information systems were mainly built around secondary, administrative processes of the value chain (e.g. accounting). But since the internet came into use, more and more primary processes have become accessible to automation: customer acquisition, ordering, billing and, in the case of intangible goods such as software, even delivery. Hence an increasing part of an

enterprise has to be modeled and a substantial part thereof is implemented. To create such an information system and to adapt it constantly to a changing environment requires a much more efficient software development process than the one suggested by the traditional methods, namely the separation into the phases analysis, design and implementation where each phase is usually performed by a different team, each relying on the documents produced in the previous phase (possibly with backtracking). In these approaches, the coupling between the phases is weak: changes to an analysis model typically require a substantial reorganization of the design models, which in turn slows down software development considerably. The ARchitecture of Integrated information Systems ARIS (Scheer, 1992) is one such traditional method with a focus on analysis. It received substantial attention both from researchers and practitioners thanks to its close relation to the SAP suite of business applications.

Now, there are several reasons why such methods are not suitable for the development of web-based applications (leading to corresponding requirements):

1. The increasing number of automated business processes requires the integration of these processes with the relevant data. But the parts (views) of conventional models are only loosely coupled (e.g. the data and process models of ARIS). A suitable method for developing web-based applications should integrate partial models, especially the process and the data/object model (*model integration*).
2. Existing methods do not cater for the needs of object orientation. But the predominant use of object-oriented programming languages in developing web-based software demands the compatibility of the modeling method with *object-oriented* concepts.
3. Traditional software development is usually quite slow. But electronic markets require a quick adaptation of web applications to changing needs.
4. The development of design models typically consists of reinventing the analysis models in a different (more formal) language. A smooth transition from analysis to design, by merely refining the existing analysis models, is preferable (*phase integration*).

To solve these problems, we make use of the MEMO framework, which represents a multi-perspective approach to enterprise modeling where all views are strongly connected via a common object-oriented core. Within this framework, the integration of processes and their relevant data is achieved by an analytical, object-oriented process definition language called EMC. Apart from model integration, EMCs also facilitate the transition from analysis to design by providing a suitable level of abstraction/formalization, hence speeding up the software development process.

Hence, EMCs provide a way of an integrated analysis of the major aspects of an information system: its structure, its processes and the required resources. Because the underlying paradigm is object-oriented, it enables a seamless transition to object-oriented design and implementation necessary for the development of web-based applications. This is further supported by the unambiguous process semantics of EMCs. In addition, we assume that users already familiar with EPCs will experience few problems in handling EMCs because they resemble each other closely. The process-driven identification of objects helps modelers with less expertise in object-oriented design to create a complete object model, a task that is both highly abstract and challenging even for software engineers. Put together, these features facilitate an efficient development of web-based applications.

The following sections will explore the details of such an approach: we start with an example scenario of a software trading company planning to go “on-line”. This example motivates the necessity of an integrated modeling of business processes and information across the various phases of software engineering. It also serves as a framework of reference for the sections to follow. We introduce the basis for information modeling in a company: MEMO and EPCs showing how integration is achieved in this setting (in particular that of processes and related documents). Then we show interpretations of EPCs found in literature and we argue why the EPCs are chosen despite their marginal usefulness as a starting point for software development and how they can be improved for this task: they are more easily understood and

more readily accepted by most business people than typical process models from computer science (such as Petri nets, for example), which is an important plus in developing software for business. Trying to make Petri nets understandable is a much more difficult task than making EPCs (in the form of EMCs) suitable for software development. To achieve the latter we introduce the Event-driven Method Chain. The principal differences between EPCs and EMCs are explained. This includes the precise meaning of each construct given in terms of Petri nets and the integration of documents (i.e. objects) into EMC process models. The chapter concludes with an empirical assessment of the usefulness of the EMC (as compared to the EPC) regarding the clarity of the description of the modeled business process (or more precise: the lack of ambiguity).

AN EXAMPLE SETTING

To illustrate the problem of developing a web-based application, we consider the example of a young mail-order company trading software products. Up to now, they were organized in a more or less conventional way: customers ordered via phone, fax or surface mail. Orders were processed manually and then entered into a database. The products were stocked in physical form as CD-ROMs. Stock management was done with the help of a stand-alone system. Delivery was effected by conventional posting, payment by cheque, credit card or money order.

Now, this company plans to operate over the internet. Apart from offering new services (such as ordering via the world wide web and downloading of the ordered product), this also requires substantial reorganization: e.g. the isolated information systems for ordering and stocking have to be integrated to allow the potential customer a combined search for price and availability of a product. The head of IT is therefore asked to draw up a sketch of the principal architecture of the new system (see Figure 1). It consists of a central database containing

information about customers, orders, items and so on. All applications, internal and external, operate on this database. Internal applications are the ones used only by the staff of the company, such as order and customer management, delivery etc. The external applications can also be accessed by the (potential) customers. They are made accessible to the world by an applet server feeding the user's browser.

The next sections will show how such an information system can be analyzed and designed rapidly with the help of the EMC language, which integrates the different views on a system as well as their migration from analysis to design models because it adheres to MEMO as outlined in the following section. We also give a short background on the business process language EPC which forms the basis for EMC.

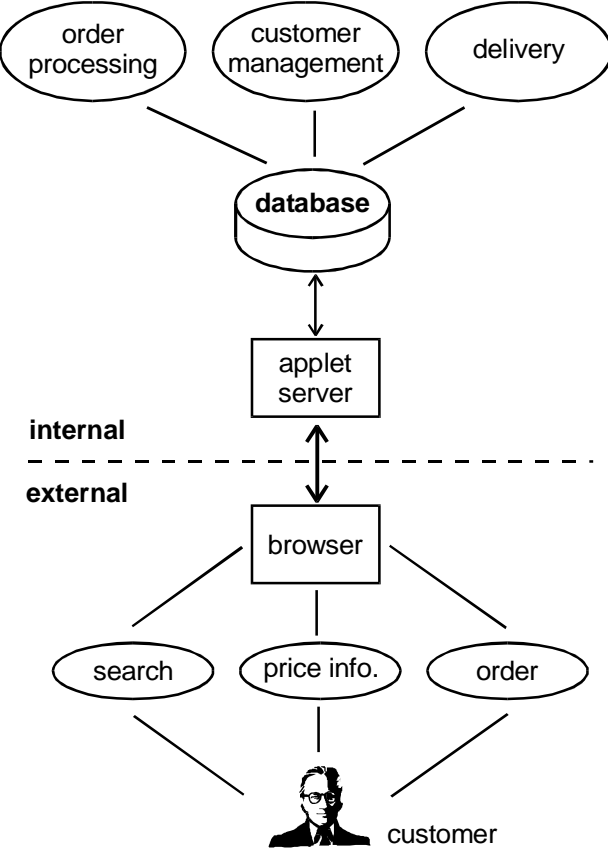


Figure 1: Example architecture of a web application for a software trading company

MULTI-PERSPECTIVE ENTERPRISE MODELING AND EPC

In the early phase of analysis, the modeler of any application is typically confronted with a yet largely unstructured problem domain. This applies to web-based applications in particular, which involve a complete reorganization of a substantial part of the company: processes to be performed over the web have to be designed newly, related internal processes have to be adapted accordingly. These changes affect not only the information system itself but also the organization as a whole resulting in a significant strategical impact of the corresponding decisions. Hence a modeling methodology should distinguish three levels of an enterprise: strategy, organization and information system (see Figure 2). But even if we restrict attention to one of the levels, too much complexity remains to be handled by one model only. Therefore each level is further subdivided into the following four foci:

- process: dynamic representation of the system
- structure: static description of the domain objects and their relations
- resources: means required to execute a process
- goals: results to be achieved and their relations

The resulting 12 partial models span the 4×3 matrix of the views of MEMO (Frank, 1997). Figure 2 gives iconized examples for the 12 views. The three-letter acronyms for each view consist of the first initial of the column header, the first initial of the row title and an M for Model. For example, the strategy process model (SPM) consists of a value chain á la (Porter, 1985) where the primary processes (procurement, production, distribution, marketing) form the basic chain (arrow-shaped boxes) with the subsidiary (supporting) activities attached to them (the flags). On the organizational and IS levels, an EPC-like language for modeling process is used. The OSM is represented in the form of an organizational chart and the ISM is an object class model. The remaining views in the matrix have not been covered yet.

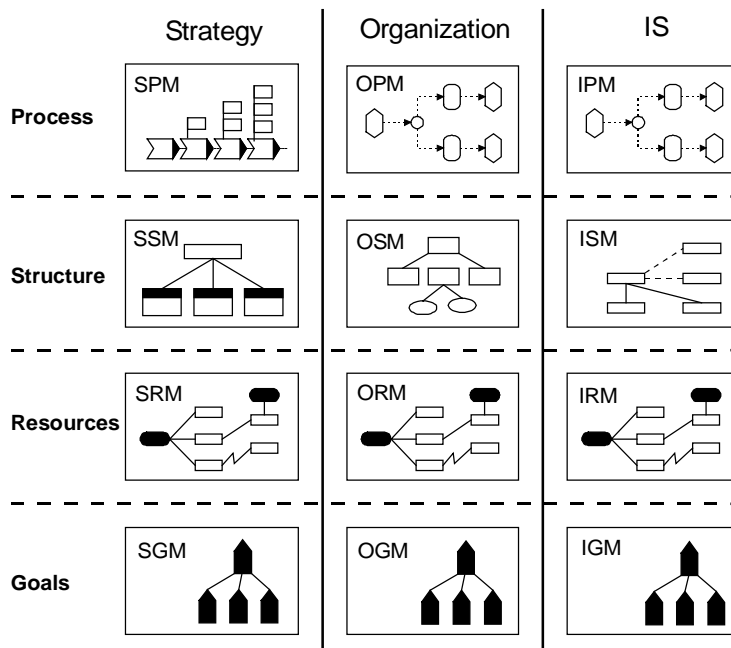


Figure 2: Perspectives on an enterprise (MEMO)

Here we focus on developing an integrated modeling language for the IS level (abstracting from the goal view for the time being). We call this language EMC. It covers IPM (IS Process Model), IRM (IS Resource Model) and the integrative part of ISM (IS Structure Model). The details of the structure are thereupon specified with the help of MEMO-OML (MEMO Object Modeling Language) described in (Frank, 1998). Here MEMO-OML is explained only insofar as it is necessary to understand the integration of all three views.

We start from the assumption that the processes of a problem domain are rather easier to identify than the objects because the latter represent a more abstract notion. Hence we put the process focus in the center of the EMC language and suggest that an initial EMC is drawn before the corresponding object model. In fact, the EMC even helps in the construction of the object model. The basic dynamic element of an EMC is the method (or service) which is linked to the objects involved in providing this service. The objects themselves and their relations are defined in MEMO-OML. We establish the resource focus by attaching the resources to the method which requires them.

The process modeling language EMC is based on the EPCs of ARIS. Although this language exhibits major shortcomings as demonstrated in the next section, we do not reject it outright

because it has proved its ease of use in practice: it is the preferred choice of consultants and large IT departments, and it is a must for companies introducing SAP. Hence a process language based on EPCs has a better chance of being accepted by the practitioner than some completely new artefact. Still, the shortcomings have to be overcome to make EPCs fit into the MEMO framework (thus achieving model integration), to align them to the object-oriented paradigm and to ensure an unambiguous interpretation of the analysis models by the designers thus achieving phase integration (see points 1-4 in the introduction). The result is the EMC language described later.

Since the EPCs were introduced in (Scheer, 1992) there have been many opinions on how a correct EPC should look like. Proposals ranged from syntactical issues (which nodes can be linked to each other) to semantics (what is the exact meaning of a connector?). On the syntactical level some rules have been established that are now generally accepted, for example (Keller, & Teufel, 1997): An EPC consists of strictly alternating sequences of events and functions (i.e. processes) that are linked by logical connectors (AND, OR, XOR). There are opening connectors (splits) and closing connectors (joins). Among the syntactical restrictions are:

K1: There are no isolated nodes.

K3/4: Functions/events have exactly one incoming/outgoing edge (except start/end events).

K6: Connectors are either splits (1 input, many outputs) or joins (many inputs, 1 output).

K8/9: An event is always followed by a function and vice versa (modulo connectors).

Sometimes it is also requested that an event should not be followed by an XOR split because events cannot make decisions. Figure 3 gives an example of how an EPC looks like.

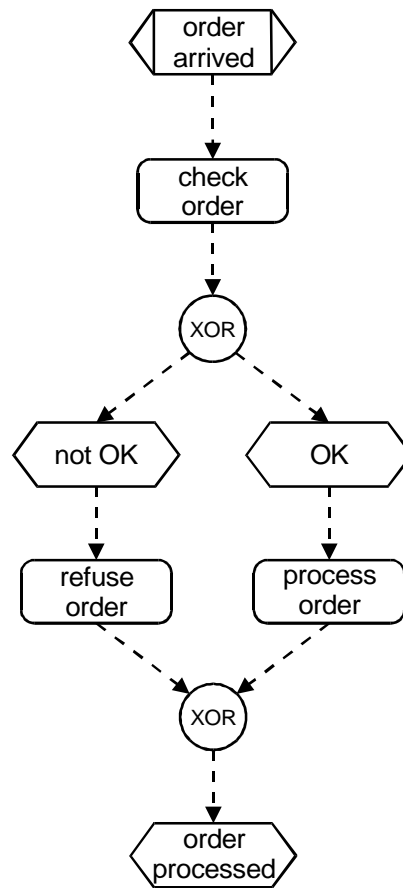


Figure 3: Example EPC

SEMANTIC MODELS OF EPCS

There is considerably less unanimity on the subject of semantics. Here we sketch only two of the existing approaches: The first was suggested by (Chen, & Scheer, 1994). That is why we call it the original semantics although it covers only a subset of all EPCs. A more elaborate model was given later by (Langner, Schneider, & Wehler, 1997). But it still requires the transformation of an arbitrary EPC into a well-formed one. Therefore we introduce a new semantics, the so-called *modEPC* semantics, which is applicable to any EPC. To facilitate the design of correct EPCs we also slightly modify the syntax concerning the problematic OR join.

The Original Semantics

The first formal approach to a semantics of EPCs was suggested by (Chen, & Scheer, 1994). The semantics is based on Petri nets, more precisely place/transition nets, which obviously

closely resemble EPCs: the functions correspond to transitions, events can be represented by places. The XOR split and join are described by the modules in Figure 4.

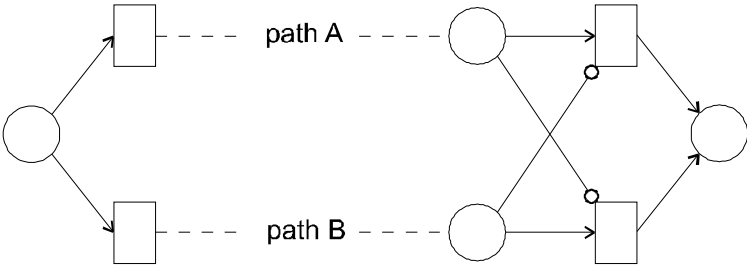


Figure 4: Petri nets for XOR split and join

The left module is the XOR split where on arrival of a token only one transition can fire removing the token necessary for the other transition to fire. Hence only one path can be activated at a time. The right module represents the XOR join which only fires if not more than one place is marked. Should both places hold tokens, the connector blocks (deadlock), thus indicating a possibly wrongful design of the EPC. This is achieved by the inhibitor edges (the ones with the small circles at the end) which inhibit firing in the presence of a token.

Analogously Petri-net modules for the AND connectors can be specified (see Figure 5).

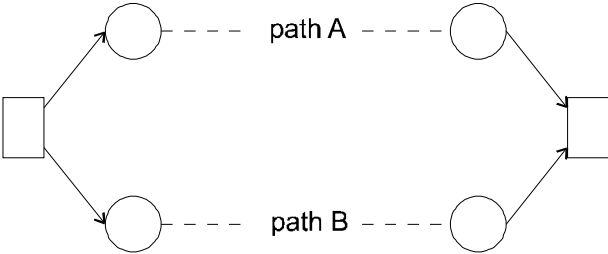


Figure 5: Petri nets for AND split and join

If we try to do the same for the OR connectors we discover that here the semantics of the join cannot be determined on itself. The EPC on the left side of Figure 6, for example, has a unique interpretation because the join brings together again exactly the paths separated by the split. So the join simply waits for the completion of all paths activated by the split.

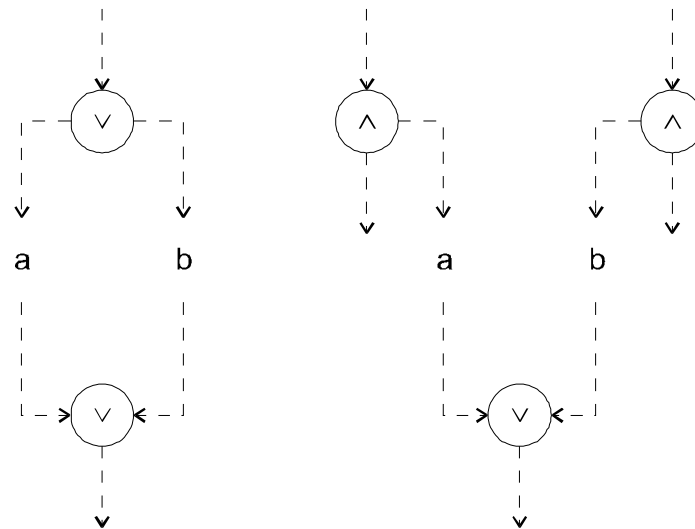


Figure 6: OR join with and without corresponding split

But what is the meaning of the EPC on the right? According to the semantics of (Chen, & Scheer, 1994) yet to be presented it has no meaning at all because the OR join has no corresponding split. Due to (Langner, Schneider, & Wehler, 1997), explained below, the OR join is interpreted as an AND, i.e. it waits for both paths. But perhaps the modeler intended the join to be triggered by the first completed path. So there are at least three possible interpretations, a situation most probably provoking mistakes in later stages of software development. For this reason we suggest an unambiguous semantics and modify the syntax accordingly. But before that we sketch the OR semantics of (Chen, & Scheer, 1994) and (Langner, Schneider, & Wehler, 1997).

In (Chen, & Scheer, 1994) there are different tokens for the branches of an OR e.g. token “a” for path A and token “b” for path B. The split informs the join of the tokens to be expected. In Figure 7 the split is to activate both paths and hence the first transition puts both a and b tokens on both successor places. The first two travel along their respective path, the other two tell the join to wait for the travelling token from both path A and path B.

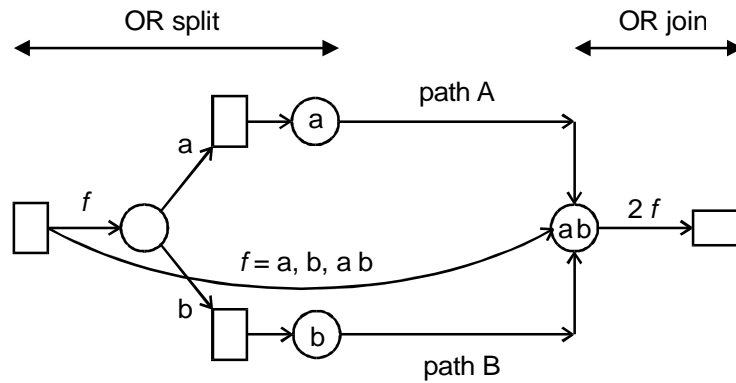


Figure 7: Petri net for the OR connectors according to (Chen, & Scheer, 1994)

But unfortunately this approach limits the amount of interpretable EPCs severely. It forces the modeler to specify splits and joins correspondingly. This is clearly undesirable for the early phase of analysis where the ideas of the modeler are not yet well structured.

A Semantics for Well-formed EPCs

A less restrictive semantics is given in (Langner, Schneider, & Wehler, 1997). It makes use of boolean Petri nets with tokens 0 (false or inactive) and 1 (true or active). The OR problem is solved by the simple trick of sending tokens along all paths: a 1 to activate it and a 0 to deactivate it. Now the OR join can wait for the arrival of tokens from all incoming paths and if at least one 1 token is present it activates its successor. The boolean transition is called branch/fork for the OR split and merge/join for the OR join. The opening and closing XOR transitions are branch and merge respectively. In the case of the AND they are referred to as fork and join. The firing rules are given by the standard truth tables of propositional calculus with the following exceptions: the entries “0 1” and “1 0” of the AND are not applicable, and neither is the combination “1 1” of XOR. The corresponding joins block on this input instead of passing on a 0 to the successor.

Strictly speaking this semantics only applies to well-formed EPCs. An EPC is well-formed if all generated tokens are extinguished eventually, no dead paths exist and no connector blocks. This is the case if all branches of a split come together in one corresponding join without jumps into or from the branches. Well-formedness is checked by a static and a dynamic

analysis only after the transformation of the EPC into a boolean net. This process involves the restructuring of not-well-formed nets to meet the criteria. The result is always a well-formed net but one that in general has not the same meaning as the EPC from which we started. Whether these fundamental changes are admissible can only be judged by the people from the responsible department. But they are usually not in a position to handle the complex transformations into well-formed nets. Hence problems of this kind can only be solved by a team of IT specialists and users but such a process is rather costly. From an economic point of view we should therefore avoid making EPCs well-formed.

IMPROVING EPCS

When modeling business processes in ARIS, we identify core processes of the company and represent them as EPCs. An EPC consists of a strictly alternating sequence of events (“invoice arrived”) and functions (or processes) such as “enter invoice” (hence its name). In addition, alternative or concurrent processes can be specified with the help of connectors (XOR, OR, AND). The model either captures existing processes or specifies planned ones, and it can be employed to reengineer inefficient business processes.

Concerning semantics, there is little unanimity. It is given only roughly (in a verbal form) in the original publication (Scheer, 1992). Later there have also been attempts to give EPCs a formal semantics, e.g. (Chen, & Scheer, 1994), (Rump, 1997) and (Langner, Schneider, & Wehler, 1997). But all approaches differ considerably, i.e. they attribute different meanings to the same EPC in many cases. Many of these discrepancies stem from diverging interpretations of the logical connectors, in particular of the (X)OR join. Additional problems are caused by an unclear concept of a start event, by the (non-)existence of a corresponding split and by the strict alternation of events and functions. The following sections will treat these issues in the

order: start events, corresponding splits, OR join, XOR join and alternation of events and functions.

Start Events

(Keller, & Teufel, 1997) defines a start event as any event without an incoming edge. But there are such events which do not trigger the whole EPC. These so-called external events only require the EPC to wait for something happening outside the process. A start event, however, invokes a new execution of the EPC template. To identify an event as a start event in this sense it is drawn with two additional vertical lines as suggested in Figure 3.

Corresponding Splits

The semantics of a join often depends on whether or not it has a corresponding split but his split cannot be derived automatically from the structure of the EPC. We have to rely on the modeler to identify it. The modeling tool can only provide him with a list of alternatives (all splits on backwards paths from the join). A pair of split and join are labelled with corresponding flags (see Figure 8). If the corresponding split is of the same type as the join we call it a matching split.

OR join

Assuming the OR join has input paths a and b (like in Figure 6, on the right) the following ambiguities may arise: If it has a matching split the semantics is usually taken to be “wait for the completion of all paths activated by the matching split”. If there is no matching split, there are three symmetrical interpretations (i.e. interpretations not distinguishing between a and b) of an OR join (see Figure 6, on the right):

- Wait for the completion of all **activated** paths (called wait-for-all). This is the default semantics because it coincides with that of a matched OR.
- Wait only for the path that is completed first and ignore the second (called first-come).
- Trigger the outgoing path c on each completion of a or b (called every-time).

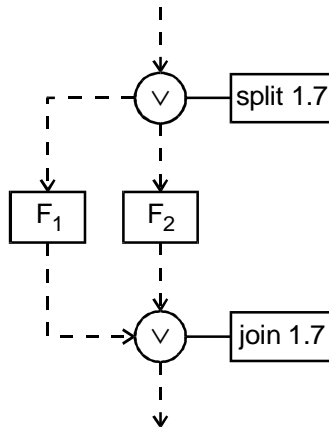


Figure 8: Matching split and join

The semantical shortcomings mentioned above can be remedied by extending the syntax of the connectors. We suggest allowing the modeler to add a comment flag to a connector. This flag can uniquely identify corresponding connectors (see Figure 8), and it may serve to clarify the intended meaning of an unmatched join (see Figure 9). Alternatively, the meaning can also be encoded in the connector symbol itself: a standard OR symbol to denote wait-for-all, a funnel-like trapezoid for first-come and an inverse trapezoid for every-time.

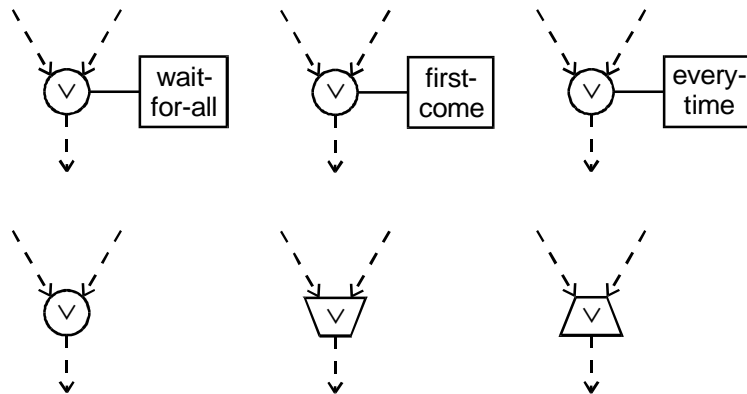


Figure 9: Making the OR join unambiguous

XOR join

Similar considerations hold for the XOR join. If it is matched by a split its semantics is straightforward: it blocks if both paths are activated and it is triggered by the completion of a single activated path. But what happens in the unmatched case? Imagine the OR connector of Figure 6, right, were an XOR join. All feasible interpretations that do not involve blocking (first-come, every-time, wait-for-all) are already covered by the OR and contradict the

exclusivity of the XOR: a token from one path may only be accepted if it is sure that no second token will arrive via the other path. But we cannot decide this if the tokens do not come from the same source (at least not statically). Hence we forbid the use of XOR in the unmatched case.

Alternation of Events and Functions

Beyond their semantical shortcomings, EPCs also exhibit some deficiencies originating in the syntactical domain. Empirical studies like (Speck, 1998) have shown that particularly middle and upper management people consider the strict alternation between events and functions as too restrictive. They find it hard to identify the necessary events on an abstract level of process description.

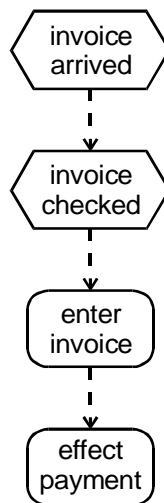


Figure 10: A non-alternating sequence of events and functions

We suggest dropping this syntactical requirement as dummy events might always be added later if this proves to be necessary. On a conceptual level, there are good reasons to be able to omit events as Figure 10 shows: if “enter invoice” and “effect payment” are performed as one unit (i.e. uninterruptedly) there is no need for an event to trigger the second activity. A similar reasoning leads us to allow two (ore more) consecutive events (see also Figure 10).

If we change the syntax of EPCs accordingly, we arrive at the so-called modified EPCs (or *modEPCs*). We specify the semantics of *modEPCs* inductively in terms of Petri nets, more precisely of place/transition nets. Events and functions correspond to places and transitions

respectively. The semantics of connectors is given separately for splits and joins. For convenience we assume that all splits have two outputs and all joins have two inputs. Defining the semantics of the splitting connectors is a straightforward matter: the AND puts tokens on both paths, the XOR on only one (see Figure 11).

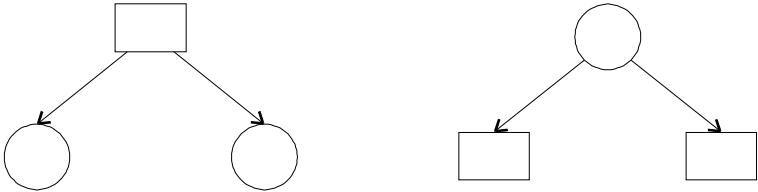


Figure 11: AND split (left) and XOR split (right)

Likewise the OR split activates the left path (left transition) or the right path (right transition) or both (centre transition). The corresponding Petri net is shown in Figure 12.

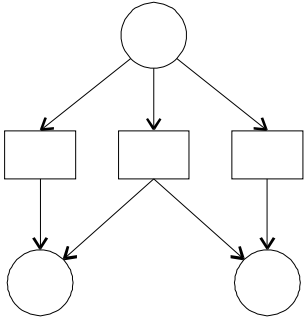


Figure 12: OR split

The semantics of the joining connectors is given in figs. 13 – 17. The AND join synchronizes the two paths (see Figure 13). The transition only fires when both paths have been completed. Observe that the AND blocks if only one path has been activated.

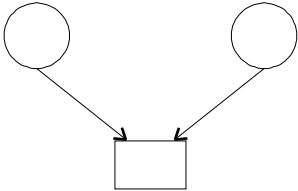


Figure 13: AND join

As already pointed out, the XOR join requires that the modeler identifies an explicit corresponding split. This might be supported by a modeling tool supplying a list of possible candidates. This split is represented in Figure 14 by the dashed places and transitions. Left as

it stands, it denotes an OR split. Omitting the centre transition (and its outgoing arcs) yields an XOR split. Doing the same with the left and right transitions instead leaves an AND split. Notice that the latter implies that the join always blocks.

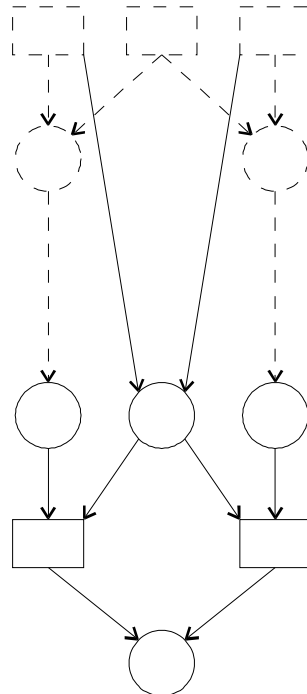


Figure 14: XOR join

The following three figures treat the different interpretations of the OR join. The simplest net is that for the every-time mode (see Figure 15). It just passes on every token it receives. So if both incoming paths have been triggered, the process following the join is executed twice.

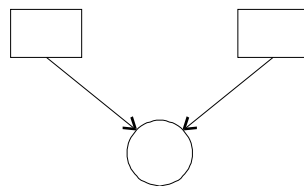


Figure 15: OR join (every-time)

For the first-come join we need a corresponding split that puts a token on the centre place to ensure that the join cannot fire twice (see Figure 16). Alternatively the token might be put there during start-up.

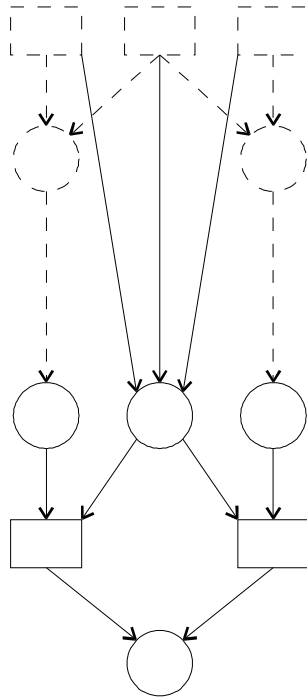


Figure 16: OR join (first-come)

The wait-for-all join needs a corresponding split, too, because only a common source of tokens for both inputs can “tell” the join whether to wait for a second token or not. If not it is immediately put there by the split (see Figure 17). The same semantics is used for a matching OR split.

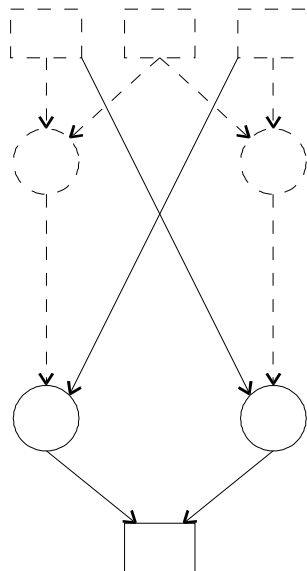


Figure 17: OR join (wait-for-all and matched OR)

THE EVENT-DRIVEN METHOD CHAIN

After having effected all the modifications suggested in the previous section, the resulting *modEPCs* can be used as a process model for application development because a unique and formal interpretation can now be given for any EPC, e.g. in the form of a Petri net. Hence *modEPCs* can be understood unambiguously by the protagonists of the design phase. This leads to less mistakes in the design model and less backtracking from design to analysis thus fulfilling the requirements of faster software development and phase integration.

If we put together *modEPCs* (process focus), classes with their attributes (structural focus) and the resources (resource focus), we arrive at the EMC. Its syntax is shown in Figure 18.

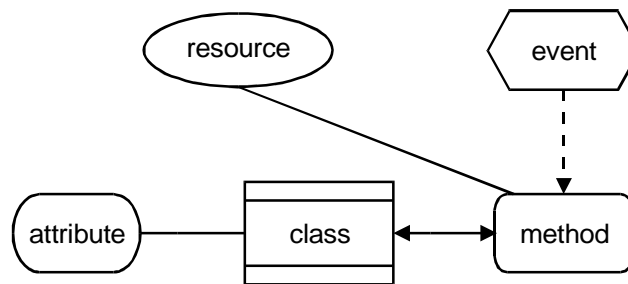


Figure 18: General syntax of an EMC

The function of a *modEPC* is now performed by some object and hence called the *service* (or method) provided by this object. Apart from this, the process part of an EMC, consisting of events, methods, connectors and control flow arcs (dashed arrows), follows exactly the syntax of *modEPCs*. The object classes involved in providing the service are linked via a double-headed, solid arrow. A class can have attached to it a list of attributes. Classes are also called the internal resources required by a service because they form an integral part of the information system. An external resource is denoted by an oval. It is connected via a solid line to the method requiring it.

Now, classifying objects into classes sharing common attributes and methods is an important object-oriented feature. Together with the characteristics provided by the MEMO-OML

(Frank, 1998), our approach fulfils all requirements of object-orientation. But how does EMC work in our example setting?

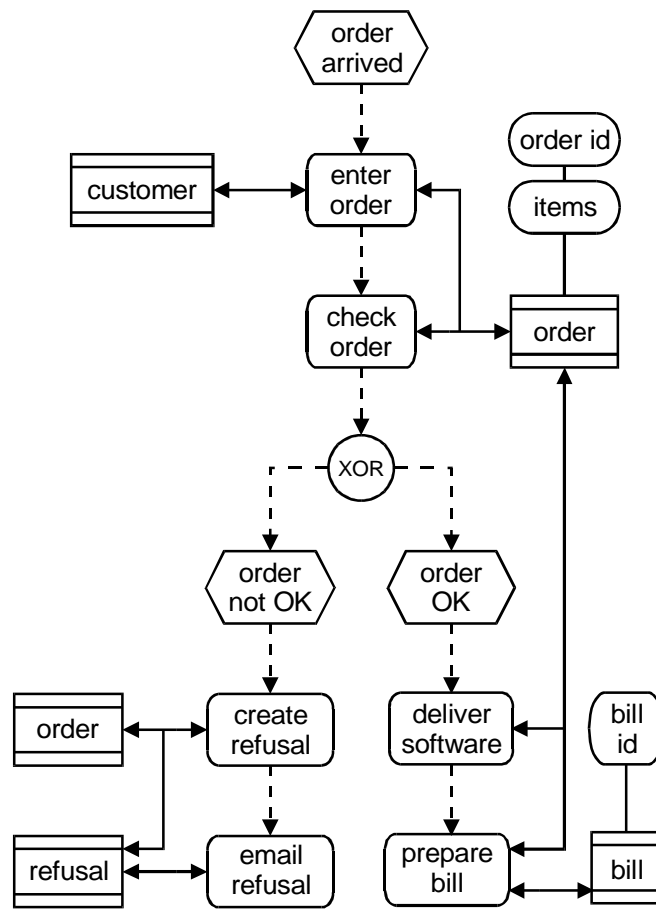


Figure 19: A part of the EMC for the example web application

Figure 19 shows a part of the EMC for the order processing within the web application outlined earlier. We assume that the process is fully automated, i.e. it requires no external resources. Upon the arrival of an order, it has to be entered into the system. This service is provided by the class *order* but it involves also the class *customer* because the respective customer has to be recorded in the order. Note: the fact that the service *enter order* is provided by *order* and not by *customer* is not represented in the EMC. Although it would have been possible to distinguish between providing and otherwise involved classes by employing different arrows, we decided against this option because during process analysis the modeler is primarily concerned with conceptual issues such as “who has to do with a service?” and not with the exact role a class plays in performing the service. But the latter

information, more precisely the assignment of services to classes, is vital for the following design phase and hence should be expressed in the object model (see Figure 20), i.e. while we are still in the analysis phase. According to the EMC, the attributes of *order* are *order id* (e.g. a number) and *items* (a list of ordered items and quantities). These attributes also constitute the skeleton of the respective class definition in the object model (see Figure 20).

After entering the order, it is checked for validity. The outcome of this check is represented by the occurrence of either the event “*order OK*” or the event “*order not OK*”. The XOR split denotes that these events are mutually exclusive. In the case of an invalid order, a refusal of the order is generated and sent via email. The items of a valid order, i.e. the software packages ordered by the customer, are delivered (e.g. via ftp) and the bill is prepared. After this, further processing may occur. Note that no attributes are specified for the class *refusal*. The reason might be that the modeler could not think of appropriate attributes and hence left this to the later stage of developing the object model.

On the basis of this EMC, an initial object model can be specified without further thinking: it simply consists of all classes and their respective attributes as found in the EMC. After this, the following steps yield a complete object model:

1. *assigning services to objects*: from the classes involved in a service, the one providing it has to be selected. There the service is recorded.
2. *finding missing attributes*: each class is thoroughly examined to check whether attributes have been forgotten e.g. because they are not necessary in the context of the current EMC. This step is best performed after all EMCs for the application lie before us.
3. *identifying potentials for generalization*: classes sharing common attributes or services are potential candidates for generalization inheriting these attributes from a common super-class.
4. *establishing associations between classes*: if more than one class is involved in providing a service, there is usually an association between the involved classes.

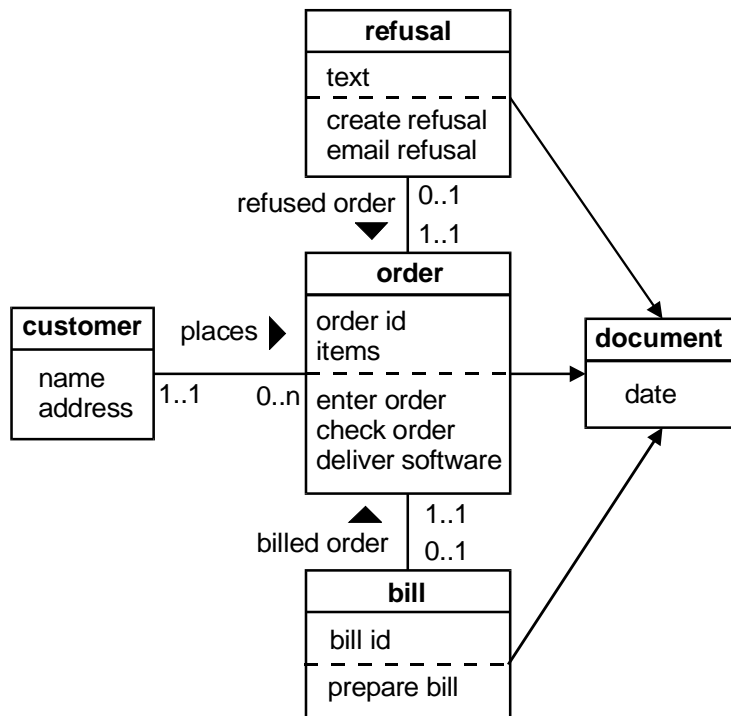


Figure 20: Object model (structural focus) for the example (in MEMO-OML)

Starting with the initial object model for the EMC of Figure 19, we assign the services to classes as indicated in Figure 20. The EMC gave no attributes for *refusal*. Looking at actual orders, bills and refusals stored in our file cabinet, we find that a refusal contains some explanatory *text* and that all letters carry a *date*. We update the classes accordingly (step 2), and we generalize them to the super-class *document* with attribute *date* (step 3). In the last step, we discover that *customer* and *order* are involved in *enter order*, which leads us to establish an association *places* between them, where a customer can place arbitrarily many orders (0..n) but an order is placed by exactly one customer (1..1). The black triangle indicates the reading direction: *customer places order*. In a similar way, *bill* and *refusal* are connected to *order*.

Please observe that the redundancy of having some of the information present in both EMC and object model (e.g. classes, attributes and services) helps to check inter-model consistency and thus serves model integration.

EVALUATION OF EMC

We conducted a student experiment in the course of an MIS class on process modeling. The objective of this experiment was to compare the quality of information systems built with the help of conventional EPCs and EPCs with the modified syntax. The class consisted of 20 students divided into two groups A and B. A group had 5 teams of 2 students. The experiment proceeded in 3 phases of 90 minutes each:

Phase 1: Each team had the task to model a core process of a hotel such as reservation, check-in, check-out and purchase. Group A used conventional EPCs, group B the modified syntax. A verbal specification of the respective process was given: “To process an invoice it has to be checked entered, payed and perhaps claimed. The check involves the ordered quantities, quality and the like. Payment is only effected after a positive check.” etc. The specification was incomplete and imprecise to resemble a ‘real’ one. It left enough room for interpretation and gave only a partial order on the events and functions. Consequently, no two EPCs delivered were the same.

Phase 2: Each team “implemented” its own model. As a target language Petri nets were chosen to avoid the intricacies of programming languages. The formality of Petri nets was sufficient to achieve the goal of this phase: to remove any ambiguity present in the EPC and to make explicit the information present in both the model and the modeler’s head.

Phase 3: Each team implemented a model of *another* team, a model of a process different from the one it designed in phase 1 to avoid an influence of the own ideas on the interpretation of the other team’s model. Again the result is a Petri net. The goal of this phase is to make explicit only the information present in the model.

From this follows that the difference between the two Petri nets for a model consists of the information added by the respective implementation team and not present in the model. The inverse measure, the congruence of the two nets, therefore represents the percentage of

information contributed by the model: the higher the congruence the clearer the model. To measure the congruence we proceeded as follows:

First we determined the node congruence by counting the coinciding nodes in both models, i.e. nodes labeled with the same event or function, and relating this to the total number of nodes in both nets. Then we computed the edge congruence for the subset of coinciding nodes in the same way. The overall congruence is the product of the node and edge congruences.

- node congruence = $2 \times \text{coinciding nodes} / \text{node total}$
- edge congruence = $2 \times \text{coinciding edges} / \text{edge total}$
- congruence = node congruence \times edge congruence

For example, if we have two Petri nets, one with 50 and the other with 60 nodes, and the nets share 33 nodes we get a node congruence of 2×33 (the shared nodes are present in both nets) divided by 110, i.e. 60%. If we further assume that 75% of the edges between the 33 shared nodes agree the overall congruence is 45%. Figure 21 shows the overall congruence of the two nets for each EPC model of the experiment.

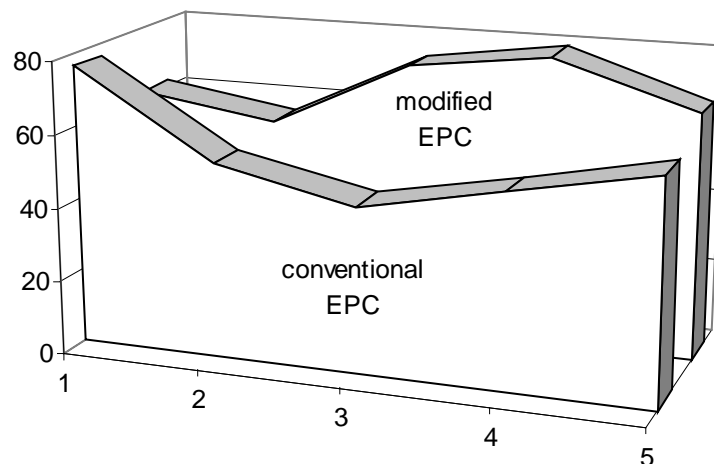


Figure 21: Net congruences for conventional and modified EPCs

The results indicate that in general a higher congruence can be achieved on the basis of the modified EPCs. The only exception was delivered by team 1 of group A. Although using the conventional, more ambiguous EPC they achieved a congruence of 76.2%, a value well above

the average and the second best score of all teams. When looking into the reasons for this exceptional result we found that the corresponding team drew a very simple EPC consisting of only 37 nodes (the others were between 100 and 200).

All in all the experiment allows the (cautious) conclusion that the suggested modifications lead to fewer misinterpretations of the model thus improving the agreement between model and implementation. Hence they facilitate a smooth transition from the analysis phase to later phases speeding up the automation of business processes. We are currently planning a field study to verify these results in a practical setting of larger dimensions.

CONCLUSION AND FUTURE TRENDS

EMCs provide a way of an integrated analysis of the major aspects of an information system: its structure, its processes and the required resources. Because the underlying paradigm is object-oriented, it enables a seamless transition to object-oriented design and implementation necessary for the development of web-based applications. This is further supported by the unambiguous process semantics of EMCs. In addition, we assume that users already familiar with EPCs will experience few problems in handling EMCs because they resemble each other closely. The process-driven identification of objects helps modelers with less expertise in object-oriented design to create a complete object model, a task that is both highly abstract and challenging even for 'OO gurus'. Put together, these features facilitate an efficient development of web-based applications.

But nevertheless, substantial work remains to be done especially concerning the still blank spots in the MEMO matrix. First of all, we need a language to describe goals to be achieved by the information system, the organization and the company as a whole. More important still, the achievement of the IS goals must be reflected in the respective IS models i.e. the defined goals should somehow guide the development of these models in a goal-achieving direction.

Some preliminary work regarding the strategy and organization levels has been done already, in particular meta models for value chains (SPM) and organizational charts (OSM). However, the inter-level integration remains to be specified. Syntactical integration only requires some kind of common terminology. It can be established by creating a small meta-meta model in the terms of which the meta models for all languages are defined. Semantical integration, on the other hand, is harder to achieve. A possible approach is the object-oriented reconstruction of all models in MEMO-OML.

REFERENCES

- Chen, R., & Scheer, A.-W. (1994). Modellierung von Prozessketten mittels Petri-Netz-Theorie (Modeling of Process Chains with the Help of Petri Net Theory). *IWi-Hefte, 107*. Saarbrücken, Germany: University of Saarbrücken, Institute of IS Research.
- Frank, U. (1997). Enriching Object-Oriented Methods with Domain Specific Knowledge: Outline of a Method for Enterprise Modelling. *Arbeitsberichte des Instituts für Wirtschaftsinformatik, 4*. Koblenz, Germany: University Koblenz-Landau, Institute of Business Informatics.
- Frank, U. (1998). The Memo Object Modelling Language (MEMO-OML), *Arbeitsberichte des Instituts für Wirtschaftsinformatik, 10*. Koblenz, Germany: University Koblenz-Landau, Institute of Business Informatics.
- Keller, G., & Teufel, Th. (1997). *SAP R/3 prozessorientiert anwenden - iteratives Prozess-Prototyping zur Bildung von Wertschöpfungsketten* (The Process-oriented Use of SAP R/3 – Iterative Process Prototyping for Building Value Chains). Bonn, Germany: Addison-Wesley.
- Langner, P., Schneider, Ch., & Wehler, J. (1997). Prozessmodellierung mit Ereignis-gesteuerten Prozessketten (EPKs) und Petri-Netzen (Process Modeling with Event-driven Process Chains (EPCs) and Petri Nets). *Wirtschaftsinformatik, 39*, 479–489.

- Porter, M. E. (1985). *Competitive Advantage: Creating and Sustaining Superior Performance*. New York: Simon & Schuster.
- Rump, F. (1997). Erreichbarkeitsgraphbasierte Analyse ereignisgesteuerter Prozessketten (Analysis of Event-driven Process Chains Based on Reachability Graphs). *Technische Berichte, 04/97*. Oldenburg, Germany: University Oldenburg, OFFIS Institute.
- Scheer, A.-W. (1992). *Architektur integrierter Informationssysteme* (Architecture of Integrated Information Systems). Berlin, Germany: Springer.
- Speck, M. (1998). Akzeptanz und Operationalität von EPK in der Modellierungspraxis – ein Erfahrungsbericht aus einem Reengineering-Projekt (Acceptance and Operational Effectiveness of EPC in Modeling Practice – an Experience Report from a Reengineering Project). Talk presented at the meeting of the workgroup on Formalization of EPC. Münster, Germany, 1998/03/17. Retrieved August 23, 2000, from the World Wide Web:
http://www-is.informatik.uni-oldenburg.de/~epk/treffen_170398/speck.ps.