

FROM PROCESS MODEL TO ELECTRONIC BUSINESS PROCESS

Peter Rittgen

Institut für Wirtschaftsinformatik, Universität Koblenz-Landau, Rheinau 1, 56075

Koblenz, Germany

email: rittgen@uni-koblenz.de

ABSTRACT

Since the early nineties business processes have been coming more and more into the focus of IS research. As a consequence numerous approaches to process modelling have been developed such as Event-driven Process Chains (EPCs, ARIS House of Business), Semantic Object Model (SOM), Bonapart and INCOME/STAR. Many of these approaches concentrate on the socio-technical dimension of business processes. Hence, apart from technical aspects such as sequences, alternatives and parallel executions, they also consider strategic goals, objectives and organizational structures. Nevertheless, current trends in the areas of “Electronic Commerce” and “Electronic Office” indicate that in future more and more business processes will be run electronically. This requires a far-reaching automation of the concerned processes. But how do you automate a process model, e.g. an EPC? Where are the obstacles that block the way from such a process model to an electronic business process and how can we overcome them? The following sections show some of the important problems arising in this context and suggest possible solutions.

1. INTRODUCTION

Business processes have been at the heart of IS research for many years if the evidence of many publications concerned with this topic is anything to go by. As a result, the amount of different approaches is equally high: ARIS/EPC, SOM, Bonapart and INCOME/STAR to name but a few. Despite this fact, one of these approaches plays a more predominant role, especially in practice, namely the Event-driven Process Chains (EPCs) of the ARchitecture of integrated Information Systems (ARIS) described in Scheer (1992). The reasons for this prevalence are manifold: on the pragmatic side, a commercial tool for EPCs (ARIS toolset) has been available for quite some time already. In addition, the great success of the SAP suite of business applications tremendously promoted the use of this method. On the other side, EPCs have also been investigated quite thoroughly in research, as the following sections point out.

When modelling business processes in ARIS, you identify the core processes of your business and you represent them as EPCs. An EPC consists of an alternating sequence of events (e.g. “invoice arrived”) and processes, also called functions (e.g. “enter invoice”). Moreover, logical connectors allow for the description of alternatives and concurrency. The resulting process model serves the documentation of existing processes, the planning of new processes to improve the company’s position, or a combination of both. If the EPC contains some new processes, the question of how to make them work arises. The “classical” way to do this is a reengineering project. Such a project identifies those process parts suitable for automation, which then are realized in a software project. All other parts are realized by organizational measures.

New areas of business such as “Electronic Commerce”, in particular, demand a high degree of automation and, due to quick changes in electronic markets, a fast realization of the process models. But the large percentage of electronic parts in the overall business process leads to huge software projects delaying realization. Two important reasons for this delay are:

1. A lot of details necessary for the implementation of the model typically require a substantial reorganization or renewal of the original model rendering it virtually useless as a starting point for software development. This is due to the fact that EPCs and similar models bear a very vague semantics.

- In EPCs, business objects such as information and documents that are handled and manipulated by a process are only loosely coupled with the latter. This weak integration makes it hard to use an object-oriented programming language for the implementation.

The first problem can be solved by making syntax and semantics of the modelling language as precise and rich as possible. Unfortunately there is a trade-off between precision and understandability. Exact formal models like Petri nets are usually not well understood by (non-IT) users. On the other hand, easy-to-understand models like EPCs lack formality. To bridge this gap we suggest to extend the EPC syntax so that a complete formal Petri net semantics can be given, thus providing sufficient precision to speed up the translation of EPCs into programs (see section 2).

The solution to the second problem requires the integration of business objects into the process model. Assuming that a language like Java is the standard for implementing electronic commerce applications, we choose an object-oriented concept for integration and extend EPCs to EMCs (Event-driven Method Chains), as it is shown in section 3.

2. SYNTAX AND SEMANTICS OF EPCS

Since EPCs have been introduced by Scheer in a rather informal way, there have been a lot of opinions about how a “correct” EPC should look like. Some suggestions concentrated on the syntactical structure (Which nodes may be connected with each other?). Others focused also on the meaning of these structures (What exactly does a connector do?). Concerning syntax some rules have been established that are now generally accepted. Examples for these rules found in Keller and Teufel (1997) are:

- K1: There are no isolated nodes.
- K3/4: Functions and events have exactly one incoming and one outgoing edge (except start and end events).
- K6: Connectors are either splits (1 input, several outputs) or joins (several inputs, 1 output).
- K8/9: An event is always followed by a function and vice versa (modulo connectors).

Sometimes the implicit assumption is made that every opening connector (split) has a corresponding closing connector (join) of the same type. Although this need not be the case, we propose an extension to the EPC syntax to make this correspondence visible: connectors can be labelled with numbers where coinciding numbers refer to a matching pair of split and join. This number should be annotated directly at the connector in the EPC diagram (see fig. 1 on the right). It is also possible for an opening or closing connector to appear alone, e.g. if an EPC has several start events triggering the process independently (see fig. 1 on the left). Then a comment flag helps to specify the intended meaning.

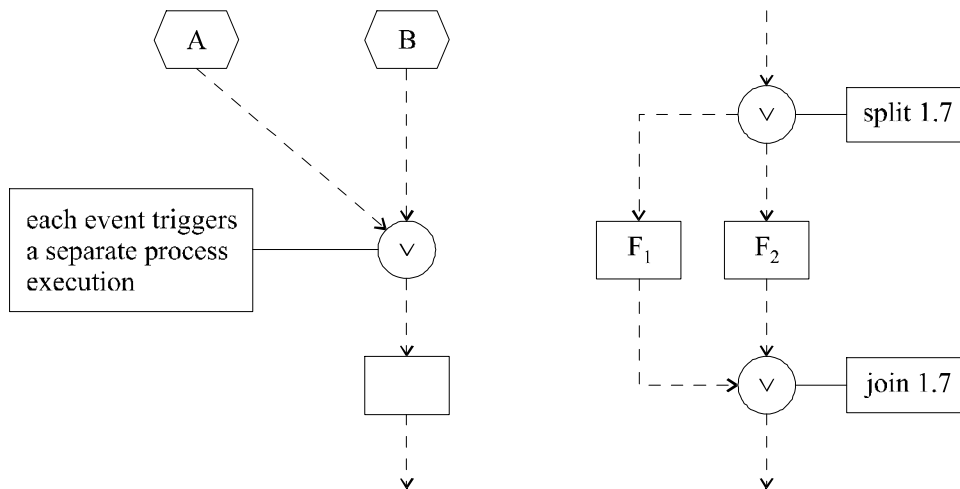


Figure 1. Example EPCs with annotated connectors

When we are looking for a precise and unambiguous semantics for EPCs, Petri nets provide an excellent basis. Fig. 2 shows the P/T nets for the connector pairs XOR and AND. In the case of the opening XOR either the upper or the lower transition fires thus activating the corresponding path (dashed arrow). In addition a control token is put on the place in the middle in any case. This prevents the XOR join from erroneous firing without prior activation of the split, e.g. by an external jump into one of the two paths. The join does not fire before both tokens have arrived, namely the control token as well as the one travelling along the selected path. The AND split sends one token each along both paths and one to the control place. The closing AND fires when all three tokens are present. The fact that the control token is always passed on directly from the split to the join makes it clear that the correspondence between them has to be represented on the syntactic level, e.g. by assigning the same number to the corresponding connectors as suggested above.

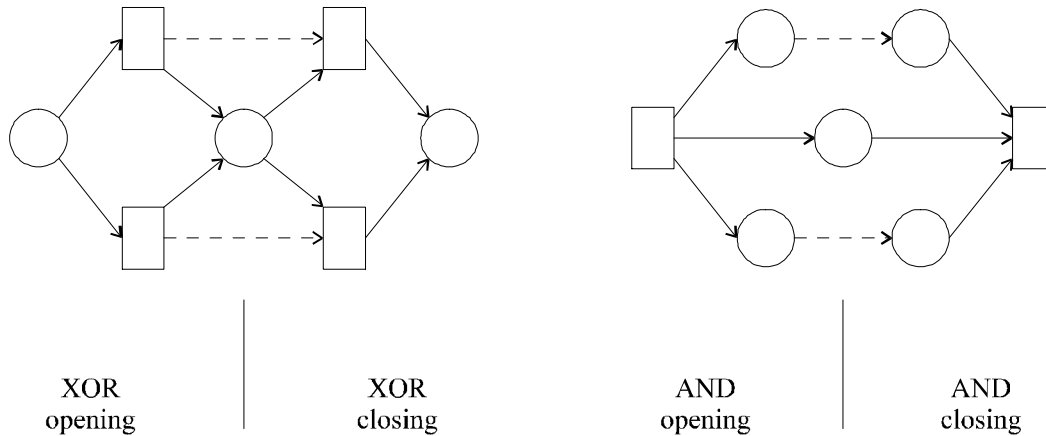


Figure 2. P/T nets for XOR and AND connectors

The case of the OR connector is more difficult: it is not enough for the split to tell the join that something is coming at all. The closing OR also needs some information on the amount of tokens to be expected because the split might have activated one or both paths. For this reason Chen and Scheer (1994) introduced different (coloured) tokens for the paths. In the case of an OR split with two paths, token "a" is sent along one path and/or token "b" along the other one. At the same time the OR join is informed which tokens have been sent so that it "knows" what to wait for.

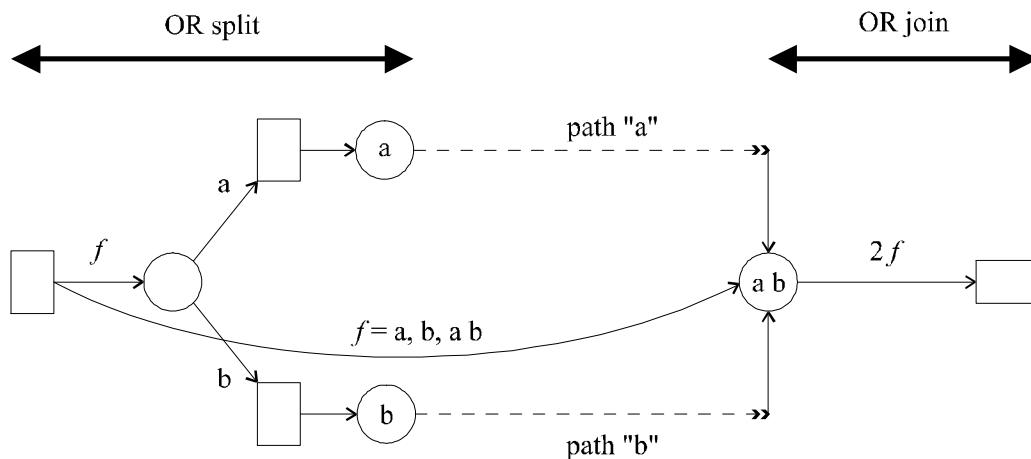


Figure 3. Coloured Petri net for the OR connectors according to Chen and Scheer (1994)

In the example of fig. 3 the OR split activates both paths. So f equals "a+b" and the first transition puts tokens "a" and "b" on each successor place. The first two travel along their respective paths, "a" along the

upper and “b” along the lower path, the other two tell the join to wait for the travelling tokens from both path “a” and path “b”. When both paths have been processed there are four tokens on the place of the OR join (i.e. $2f = 2a$ and $2b$) and the transition fires.

The same can be achieved with a simple P/T net, too (see fig. 4).

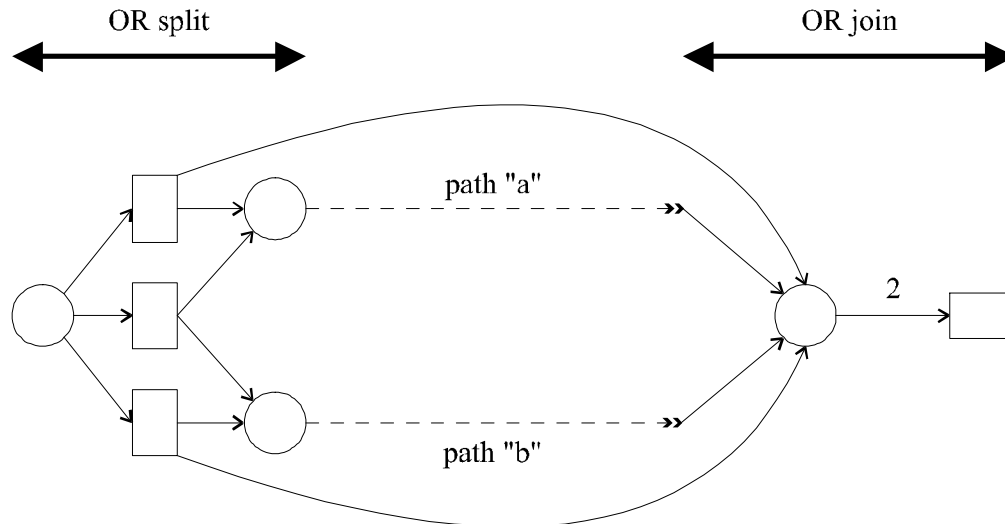


Figure 4. P/T net for the OR connectors

If the upper transition fires, a token to activate path “a” is generated and another one for the joining place. But due to the edge weight of two the join needs a second token to fire. That becomes available as soon as path “a” has been finished. The net proceeds analogously if path “b” is selected instead. Now, if the transition in the middle fires, corresponding to an activation of both paths, two tokens are generated again but this time they are both sent along their respective paths and the joining place is initially empty. The join still needs two tokens to fire, so it has to wait for the termination of both paths, which is the intended behaviour.

The suggested semantics fits exactly the syntax with the numbered connectors as can be shown easily. This enables an unambiguous and correct interpretation of any EPC. The EPC can be translated into an executable Petri net directly and automatically without having to perform the usual manual modifications beforehand (see Langner et al. (1997), for example). The resulting net can be used to simulate the process or as a starting point for software development. So process models that are based on the syntax and semantics suggested above can speed up the implementation of electronic business processes considerably. Section 4 shows this for an example from electronic commerce.

3. INTEGRATION OF BUSINESS OBJECTS

A rapid and simple implementation of control structures alone is not sufficient to produce software from models of complex business processes. Almost any process in a company usually involves business objects such as documents, for example. If you do not consider them in your process model right from the start, a later integration will prove very difficult: substantial changes to the process model even going as far as a completely new design from scratch are the rule rather than the exception. See section 4 for an example of such a problem arising in the context of a company that plans to do business over the web. For this reason the EPCs have been extended by Scheer, Nüttgens und Zimmermann in Scheer et al. (1997) to cater for object-orientation. The basic principle of these so-called oEPCs is that an event no longer triggers a function but an object class. This class then looks after the treatment of the event activating the corresponding methods which are annotated on the right-hand side of the class. The necessary attributes are listed on the left-hand side (see fig. 5).

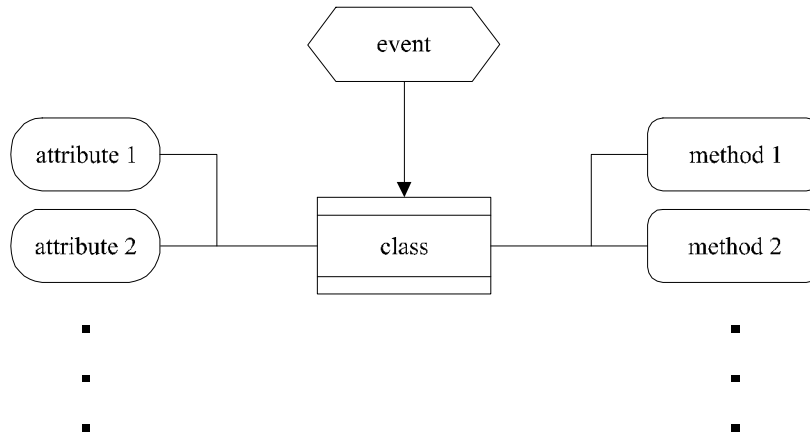


Figure 5. Syntax of oEPC

An example of an oEPC is drawn in fig. 6.

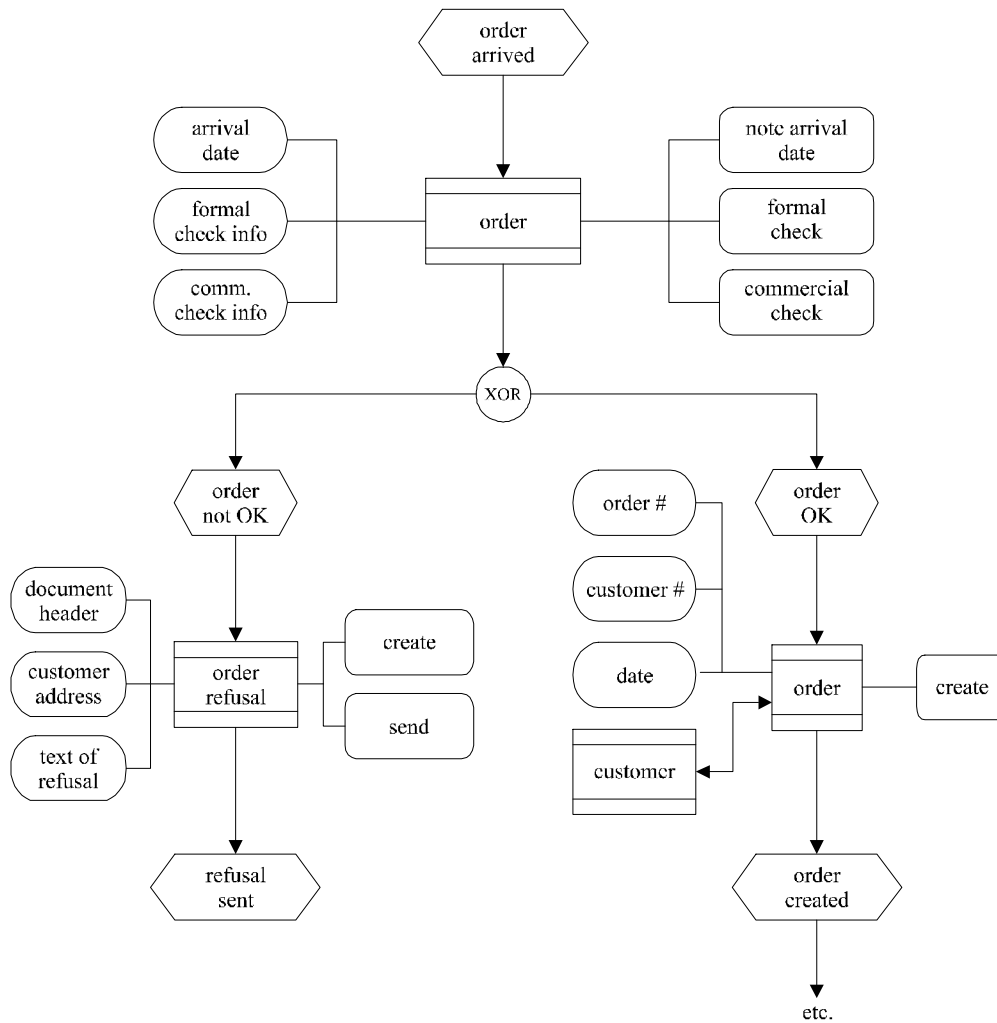


Figure 6. Example of an object-oriented EPC according to Scheer et al. (1997)

But is this concept really satisfying? Letting an event trigger a class appears to be counterintuitive. In addition, we only see which methods are invoked at all but not in which order. This is left for future phases of the software development. But again, this leads to the problem already mentioned above for the standard EPC: weak spots discovered in a later phase such as missing or unfeasible methods require a revision of the oEPC and hence a jump back to the modelling phase. To avoid such cycles the methods and not the classes should take the place of the functions. This would also come closer to the intuitive notion of an event triggering an activity (the method) and not an abstract concept (the class). The class is connected to its method via an edge. In the same way all attributes are connected to the class. The class itself is represented only once. We call the resulting diagram an Event-driven Method Chain or EMC.

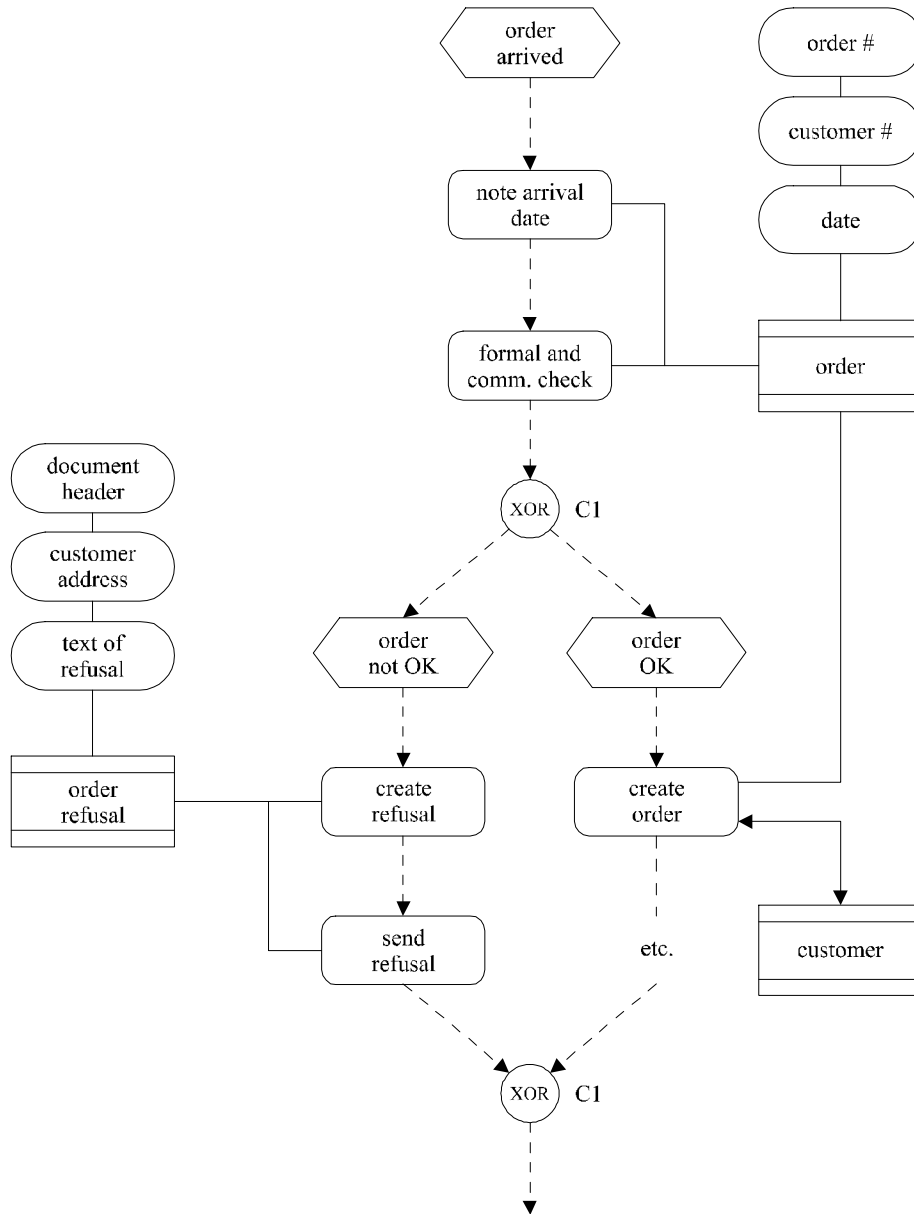


Figure 7. Event-driven Method Chain of the example process

The EMC for the example business process of fig. 6 is shown in fig. 7. Note that some slight modifications have been introduced. They are necessary because the original oEPC contains some modelling mistakes that are not obvious at first sight but become apparent when drawing the more

elaborate EMC. For example, the check bits of the oEPC are a negligible technical detail that becomes redundant when we bring into the open the structure of the test in the EMC.

The general syntax for EMCs is shown in fig. 8.

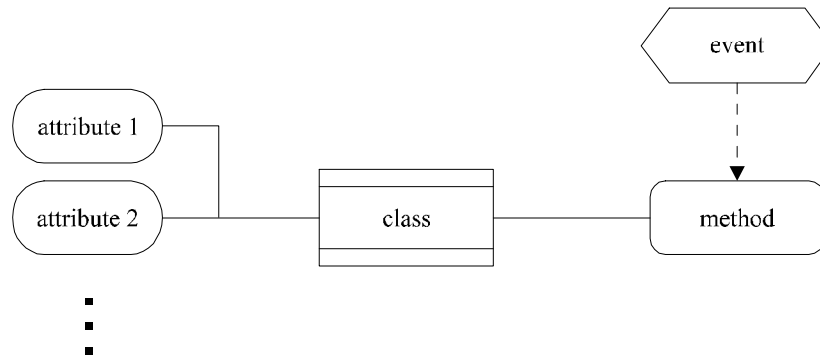


Figure 8. Syntax of an Event-driven Method Chain

The syntax of an EMC corresponds largely to that of an EPC with one major exception: we do not demand that in an EMC events and methods alternate strictly because there may be good reasons to allow a sequence of two events or two methods, respectively, as fig. 9 shows. You might argue that between the two events a check operation takes place but then you may choose to abstract from it because in this particular case the check is performed manually and is hence not needed in an IS model. Corresponding connectors are labelled with the same identifier (C1 in fig. 7) as already demanded for EPCs in section 2.

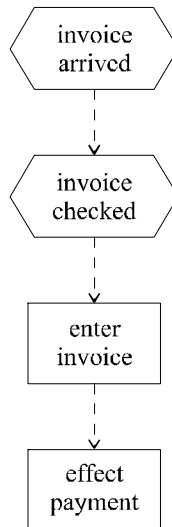


Figure 9. Two events followed by two actions

So all in all the concept of EMCs offers many advantages over that of oEPCs. In addition to a less ambiguous notation it provides a more comprehensive semantics: the example oEPC merely shows the fact that the class “order” requires the class “customer” whereas the EMC also tells you why it is necessary, namely to initialize the customer number when creating an order. While this is important even on the conceptual level, details that are only relevant for the implementation are neglected, e.g. the check bits of fig. 6.

Moreover, in an oEPC a class usually appears in several places of the diagram. Apart from giving a confusing impression, this makes the actual definition of the class in the object model very difficult because you have to collect its attributes and methods from all over the place. Naming classes, attributes and methods repeatedly also introduces unwanted redundancies which are a source of potential mistakes,

for example, using the same attribute with slightly different spellings. In the EMC, however, each attribute and each method is only mentioned once (as is the class).

4. EXAMPLE

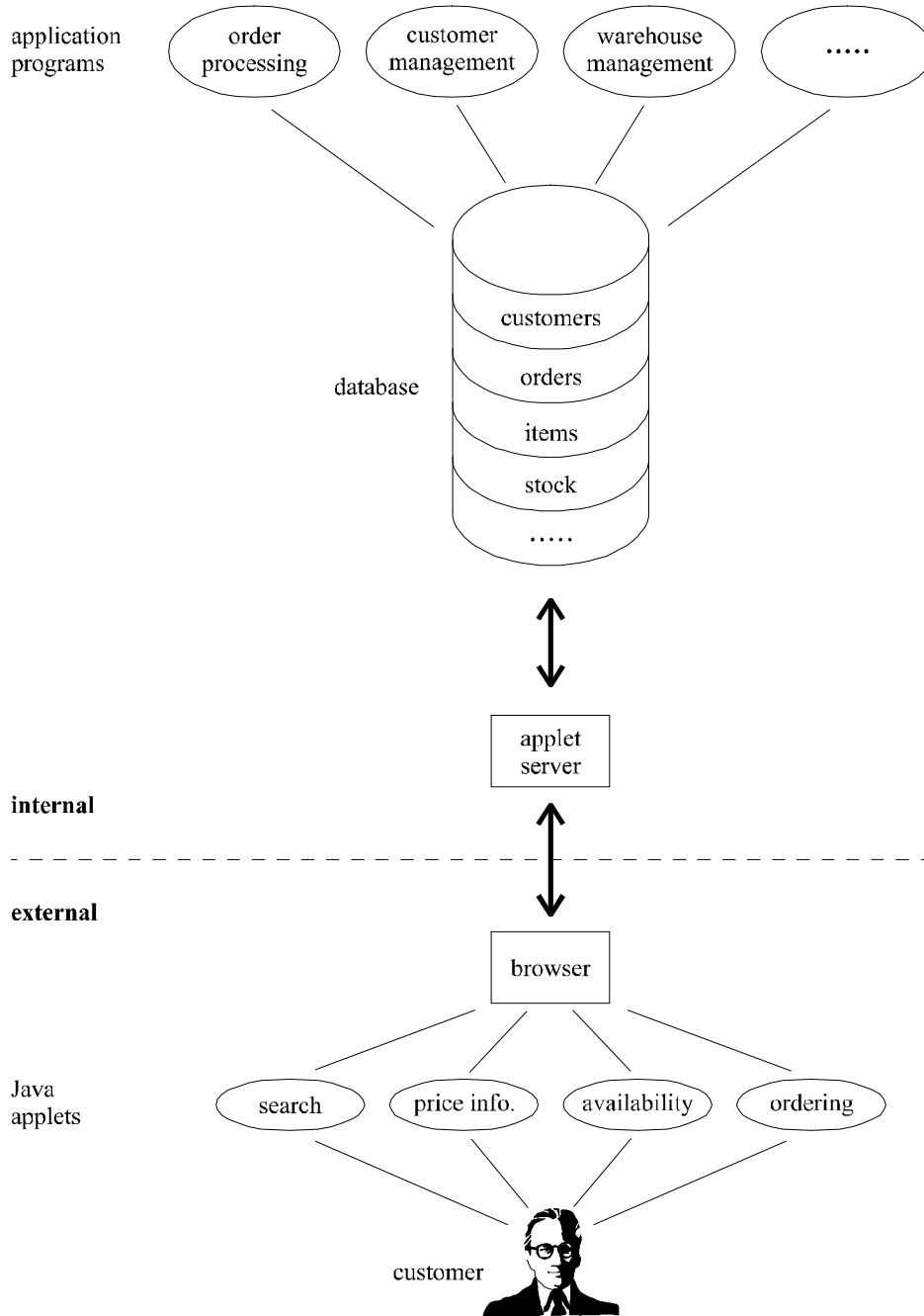


Figure 10. Architecture of a web order system

Imagine a mail order company that is planning to operate on the internet due to a decrease in customers ordering via phone and conventional mail. At the moment the situation is like this: many tasks are still carried out purely manually, e.g. the processing of mails and telephone calls. Others are supported by

computer applications each working on a distinct database. Now the head of the IT department comes up with a rough sketch for an architecture of the new web order system (see fig. 10). In order to build such a system two major problems have to be solved:

1. Some of the manual processes have to be automated (e.g. order processing).
2. All internal applications have to operate on a common database.

The first point requires the modelling and implementation of the corresponding processes from scratch. For the second point one could imagine the integration of the existing applications. But the IT manager finds that most of the old data models are inconsistent and heterogeneous and the old applications do not support all the functionality needed for the new system. Hence the integration would be more expensive than modelling and implementing the new system from scratch.

In this scenario the head of IT might find the EMC method useful for his purposes:

- The close resemblance of EMCs to the well-established EPCs ensures a high user acceptance. This is important because people from various departments will be involved in the modelling process.
- The unambiguous semantics of EMCs allows a smooth transition from the analysis phase to the more formal phases of design and implementation.
- This is also facilitated by the underlying object-oriented paradigm which respects the requirements of programming in an object-oriented language like Java at an early stage already.

5. SUMMARY AND OUTLOOK

In section 3, we suggested the Event-driven Method Chain as a process model that is particularly suitable for the development of electronic business processes. In this approach, the centralized definition of a class together with its attributes and methods enables the immediate generation of a class skeleton for programming including attribute declarations and method heads. In addition, code can be generated, too, for the EMCs of methods according to the formal semantics outlined in section 2. Unspecified methods remain to be programmed manually but still the EMC is closer to the object-oriented program than an oEPC. So all in all, EMCs shorten the path from process model to electronic business process significantly.

Nevertheless, EMCs leave open a lot of problems with the modelling of information systems. The acceptance of EPCs (and hence EMCs) is low at higher management levels where such an approach is considered to be too formal according to Speck (1998). The relation of EPCs to process models prevalent in this area (network diagrams, project plans) remains to be investigated. The empiric evidence of Speck (1998) also shows the necessity of additional views on the information system (organizational, strategic etc.). Frank (1997), for example, suggests to enhance an object-oriented method by domain-specific knowledge thereby arriving at a multi-perspective approach combining process, structure, resource and goal-oriented views to cover all levels from strategy via organization down to the information system. The views and levels are integrated using a common object-oriented core. Unfortunately, methods of this type require quite a complex language design which makes them less accessible to formalization and hence automation. This in turn leads to long and costly software projects.

If we consider that management, on the one hand, demands an informal language and that programming, on the other hand, demands a completely formal language, how can we then bridge this apparent gap? The first solution that comes to mind is that we accept the situation as it is and try to improve the software process gradually. Another idea might be to bring the two extremes closer to each other. Bringing computers closer to humans is studied under the heading "soft programming". Going in the opposite direction can be done in at least two ways: the first way, "organizational programming", implies to create organizations to solve complex problems. Each organizational unit is independent (e.g. a company) and typically quite small. It solves only a small aspect of the whole problem which is easy to look over and hence to implement. The second way could be called "cultural programming". If we assume that sufficient complexity remains in an organizational unit so as to make complete formal treatment impossible, the members of the organization will have to have a certain understanding of how to specify an information system. Knowing modelling languages may well be as important in tomorrow's culture as being able to read and write is in today's.

But none of the outlined approaches has the potential of bridging the huge gap alone. It is a major challenge for IS research in the next millennium to make these approaches converge.

6. REFERENCES

- CHEN, R. and A.-W. SCHEER (1994). Modellierung von Prozeßketten mittels Petri-Netz-Theorie. IWi-Heft 107, Institut für Wirtschaftsinformatik, Universität Saarbrücken.
- FRANK, U. (1997). Enriching Object-Oriented Methods with Domain Specific Knowledge: Outline of a Method for Enterprise Modelling. Arbeitsberichte des Instituts für Wirtschaftsinformatik Nr. 4, Universität Koblenz-Landau.
- KELLER, G. and Th. TEUFEL (1997). *SAP R/3 prozeßorientiert anwenden: iteratives Prozeß-Prototyping zur Bildung von Wertschöpfungsketten*. Addison-Wesley, Bonn.
- LANGNER, P., Ch. SCHNEIDER and J. WEHLER (1997). Prozeßmodellierung mit Ereignisgesteuerten Prozeßketten (EPKs) und Petri-Netzen. *WIRTSCHAFTSINFORMATIK*, **39** (5), 479-489.
- SCHEER, A.-W. (1992). *Architektur integrierter Informationssysteme*. 2nd Edition. Springer, Berlin.
- SCHEER, A.-W., M. NÜTTGENS and V. ZIMMERMANN (1997). Objektorientierte Ereignisgesteuerte Prozeßkette (oEPK): Methode und Anwendung. IWi-Heft 141, Institut für Wirtschaftsinformatik, Universität Saarbrücken.
- SPECK, M. (1998): Akzeptanz und Operationalität von EPK in der Modellierungspraxis: ein Erfahrungsbericht aus einem Reengineering-Projekt. Arbeitskreistreffen Formalisierung der EPK, Münster, 1998-03-17, http://www-is.informatik.uni-oldenburg.de/~epk/treffen_170398/speck.ps