Business Process Diagrams: An UML Extension

Peter Rittgen
TU Darmstadt, Hochschulstr. 1, 64289 Darmstadt, Germany
phone: +49(6151)16-4416, fax: -5162
E-mail: rittgen@bwl.tu-darmstadt.de

ABSTRACT

Although UML offers models that can be used to describe business processes, many practitioners nevertheless prefer to employ languages that are specifically designed for this purpose. These business process languages typically provide only a weak integration with software modeling languages such as UML. To enhance the support of software development we therefore suggest to extend UML's activity diagrams with a business process semantics which leads us to Business Process Diagrams (BPDs). We show how to derive BPDs from the well-known business process language of Event-driven Process Chains (EPCs). We use Petri nets as a common formal process meta model for both.

INTRODUCTION

When analyzing a company for potentials of information systems support, a major task consists in identifying the relevant business processes and describing them in a suitable modeling language. Many such languages have been developed over the years such as IDEF (Integrated DEFinition, (Bruce, 1992)), Role Activity Diagrams (Ould, 1995) and ARIS/EPC (ARchitecture of integrated Information Systems / Event-driven Process Chain, (Scheer, 1999)) to name but a few. They share a common characteristic in that they are not equipped to support the design of software. The Unified Modeling Language (UML), cp. e.g. (Rational Software et al., 1997), on the other hand, does provide the features pertinent to software engineering but it is less qualified for domain-oriented models. In practice this leads to a separation of concerns but also to heterogeneous usage of modeling languages: domain experts using business languages, and software engineers using UML. This entails an undesirable gap between domain and software models representing a source of mistakes that are hard to correct.

Hence we suggest Business Process Diagrams (BPDs) – a language closely related to UML activity diagrams – for both worlds taking a closer look at a typical business process language called Event-driven Process Chains (EPCs). Our main objective is to ensure that all features of EPCs are present in BPDs, too, and at the same time make certain that the semantics of corresponding EPC and BPD diagrams coincide. We achieve that by using the formal, i.e. mathematical, process language of Petri nets as a common meta model to define the meaning of both diagram types. That will allow us to conclude that the resulting language of BPDs is suitable for designing processes not only in the software but also in the business domain. It also enables an automatic transformation from EPC to BPD and back, i.e. switching between business and software view.

In the following sections we first introduce Event-driven Process Chains as typical models for business processes and we define their semantics in the light of the common meta model of Petri nets. We then go on enhancing the suitability of UML activity diagrams for business modeling which leads us to Business Process Diagrams. Their semantics is also based on the common meta model which ensures the compatibility of both EPC and BPD. We conclude by showing examples of typical business processes and their representation as EPCs and BPDs.

EVENT-DRIVEN PROCESS CHAINS: A SPECIFIC BUSINESS PROCESS LANGUAGE

Event-driven Process Chains were introduced to draw a graphical representation of a business process. They consist of the following elements shown in Figure 1.

Since the EPCs were introduced by Scheer 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). The AND stands for parallel threads, the XOR for mutually exclusive alternatives and the OR for selecting arbitrarily many alternatives. Events are instantaneous happenings which trigger a business function or process. They may also be the result of the finishing of a function or process. A function is an elementary business activity, a process is a business activity which is refined through another EPC. A function/process can have attached to it the organizational unit responsible for it and data containers from which it gets input or to which it stores output. Figure 2 gives an example of how an EPC looks like.

A PETRI-NET MODEL FOR EVENT-DRIVEN PROCESS CHAINS

The precise meaning of an EPC is largely subject to personal interpretation, especially where unmatched joins and timing are concerned (Rittgen, 1999). This seeming deficiency is not the result of a bad language design but rather a requirement for the analysis stage where the participants typically do not yet have a deep and clear understanding of the business process. So freedom of interpretation is a pro at this stage
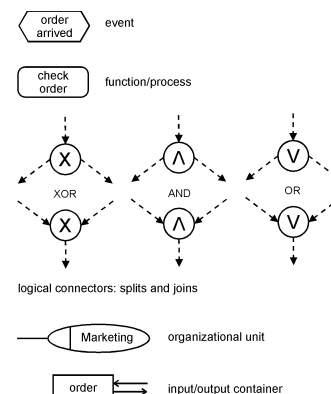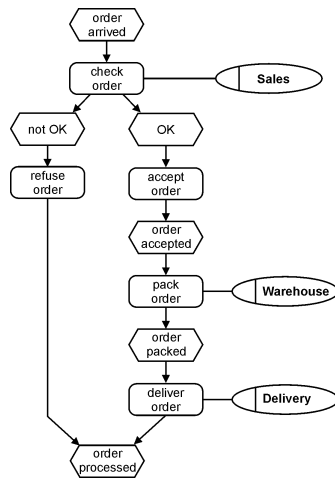
Figure 1: EPC elements

Figure 2: Example EPC

and therefore a precise semantics was left unspecified intentionally by the proponents of the EPC. But at later stages we do require more precise models and hence the issue of a formal semantics for EPCs has been studied quite thoroughly (e.g. (Chen & Scheer, 1994), (Langner et al., 1997) and (Rump, 1997)). All approaches have in common that they transform the ambiguous EPC into a Petri net with a unique and precise meaning. This transformation entails that a number of possible interpretations are thrown away. Only the "correct" one is represented in the Petri net. But who is to say which interpretation is correct? This way we might easily eliminate the one which was intended by the modeler or the one which would have come out as the result of the collective process driven by the people involved in building an informations system: users, domain experts, software engineers, (project) managers etc.

In (Dehnert & Rittgen, 2001) we therefore suggested that the transformation into a formal language should keep all possible interpretations. As a consequence we arrive at the rules displayed in figure 3. The left side of each bar shows an EPC element, on the right side you find the corresponding Petri net element. For a detailed description of this process we refer the reader to (Dehnert & Rittgen, 2001). On the basis of these rules and a Petri net semantics for BPDs (enhanced Activity diagrams) we will be able to translate between EPCs and BPDs and hence between business and software views on processes.

ACTIVITY DIAGRAMS AND BUSINESS PROCESSES

Although the proponents of UML themselves suggest that activity diagrams can be used for business process modeling (cf. (OMG, 2000, p. 1-9)) there is some doubt as to whether they actually cover all aspects

Figure 3: Transformation EPC ® Petri net
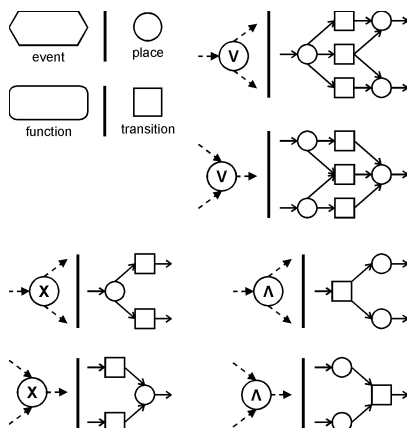
required for modeling business processes (cf. (Nüttgens et al., 1998). (Simons & Graham, 1999) point out that activity diagrams are better suited for this purpose than use cases and other UML diagrams and (Barros et al., 2000) identify areas of improvement. Among the business process features missing in UML 1.3's activity diagrams are:
•   an event-control mechanism,
•   handling of errors / exceptions,
•   flexible assignment of organizational units responsible for an activity,
•   assignment of data containers (here: objects) to activities.

(OMG, 2000, p. 3-146) defines:
An activity diagram is a special case of a state diagram in which all (or at least most) of the states are action or subactivity states and in which all (or at least most) of the transitions are triggered by completion of the actions or subactivities in the source states. The entire activity diagram is attached (through the model) to a class, such as a use case, or to a package, or to the implementation of an operation. The purpose of this diagram is to focus on flows driven by internal processing (as opposed to external events). Use activity diagrams in situations where all or most of the events represent the completion of internally-generated actions (that is, procedural flow of control). Use ordinary state diagrams in situations where asynchronous events occur.

This means that in activity diagrams a transition is typically not associated with an event because it is triggered by the completion of its predecessor. Nevertheless it does no harm to allow for this because the activity diagram is formally a special case of a state chart diagram and the latter provides event-controlled transitions. We only have to define what an event-controlled transition leaving an action state means, especially in the presence of additional, event-less transitions, because that is not covered by the original definition (see figure 4).

If the *event* takes place before the *activity* is finished, the *activity* is aborted and state *S2* is entered. Otherwise *S1* is entered upon termination of the *activity*. This mechanism can also be used to handle errors / exceptions where the error is an event aborting the current activity and entering a state of error handling / recovery.

To assign organizational units to activities (OMG, 2000) suggests the use of the so-called swim lanes. A swim lane is a rectangular area separated by vertical lines from the swim lanes to the right and the left. Each swim lane represents an organizational unit. All activities that fall within the responsibility of this organizational unit are collected within its swim lane. While this notation is quite useful for smaller diagrams it becomes more and more tedious as the number of activities increases. There is a trade-off between the vertical partitioning into organizational units and the horizontal ordering along the control flow that often results in a 'messy' arrangement of the transitions: activities that follow each other temporally might be in distant swim lanes and activities in adjacent swim lanes might be far removed from each other concerning the logic of execution. In such cases the diagram cannot be arranged clearly.

The first step of solving this problem consists of relaxing the strict geometric shape of a swim lane from a rectangle to an arbitrary closed curve (a 'cloud'). Figure 5 shows how a cloud can be employed to group the activities for which the sales department is responsible. But even a cloud cannot group activities satisfactorily that are scattered over a large area with many unrelated activities in between. In these cases it makes sense to attach the organizational unit directly to the activity as

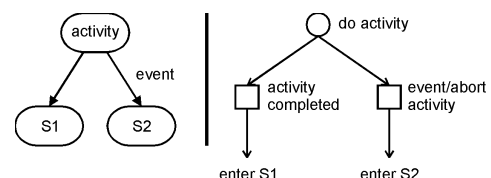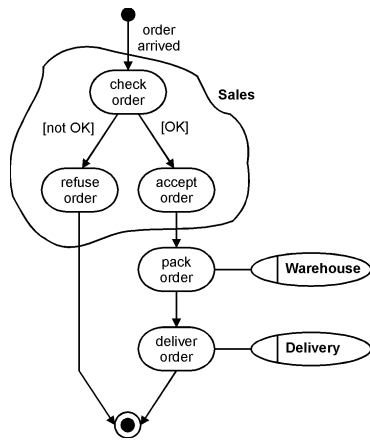*Figure 4: Events in activity diagrams*

Figure 5: BPD for the EPC of figure 2

Figure 7: Guarded branch
(notation and semantics)

done in EPCs, i.e. in the form of an ellipse with a vertical bar. Such a notation is also useful when an organizational unit is responsible for only one activity in the whole BPD as the example in figure 5 shows.

To summarize the argument we suggest to use a mixture of notations to represent the organizational aspect of a business process, i.e.:
• swim lanes as proposed for activity diagrams in (OMG, 2000) (for small diagrams),
• clouds for grouping neighbouring activities (in larger diagrams), and
• EPC-style ellipses for scattered or individual activities (in larger diagrams).

A similar notation can be used for the assignment of data containers / objects.

PETRI-NET SEMANTICS OF BUSINESS PROCESS DIAGRAMS

If we intend to use Business Process Diagrams for both business processes and the dynamics of software we have to go into the precise meaning of these diagrams first. Activity diagrams are defined in (OMG, 2000) as a variation of state machines where each state represents the performance of an activity. It is drawn as a rectangle with rounded vertical lines. The state contains a do activity and optionally an entry and/or an exit action. Entry and exit actions are performed on entering or leaving the state respectively. Their execution is considered to be instantaneous. The do activity is performed while being in the state. It can take any amount of time. Upon termination of this activity the state is left.

Figure 6 shows a generic activity state together with its semantics in Petri net notation. A Petri net is a bipartite graph of places (circles) and transitions (squares) where a transition can fire if all incoming places are occupied by tokens. Upon firing a token is removed from each incoming place and thereafter one token is put on each outgoing place. See (Peterson, 1981) for a detailed treatment of Petri nets. In the Petri net of figure 6 the entry action is performed (on entering the
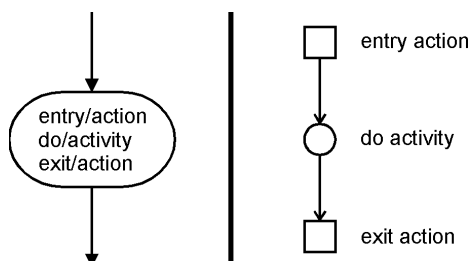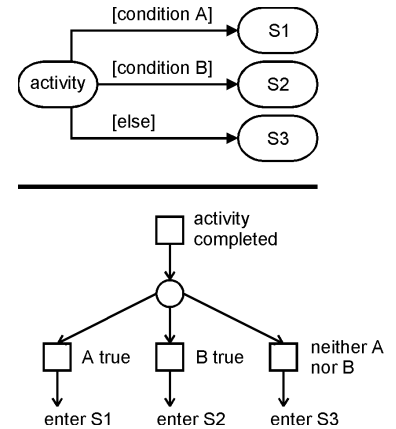
state) and a token put on the place. While the activity is performed the token remains on the place. Upon its termination (which coincides with performing the exit action) the token is removed.

If we want to express non-sequential behaviour there are two different ways of splitting the path of execution: branching into alternative paths and forking into parallel paths. The branch is simply denoted by more than one arrow leaving a state (see figure 7). A guard should be specified to determine which path is selected. A guard is a Boolean condition written in square brackets. If it evaluates to true the corresponding path is chosen. An else guard can be specified which holds if all other guards are false. It should in fact be present if such a situation can arise to prevent the process from blocking. Exactly one path is chosen so the branch corresponds to the exclusive OR (XOR) connector of EPCs. If more than one guard is true one of the associated paths is selected arbitrarily. To avoid misinterpretation in this case it is recommended to cascade the decisions in the form of a binary tree as shown in figure 8.

The notational element for cascading guards is the diamond shape. It has no semantics of its own and only serves as an anchor point for splitting a path. The design in figure 8 ensures that condition A is given preference if both A *and* B are true.

The second way of splitting up the execution is the fork. It is denoted by a bar (see figure 9). If it is unguarded it simply refers to the parallel, independent execution of both paths (called thread 1 and thread 2). In this case it corresponds to the AND connector of EPCs. The join of the parallel paths is synchronous, i.e. it waits for the completion of both paths. If a path is guarded and the guard is false it is neither taken nor waited for. A fork where all paths are guarded corresponds to the (inclusive) OR connector of EPCs as arbitrarily many paths can be taken. The rules governing activity diagrams demand that guarded threads have to be well-nested, i.e. jumps into or out of such a thread are not allowed except for synchronizing between threads. Please observe that this restriction is problematic in the case of business process modeling as it drastically limits the number of valid models. A detailed discussion of this problem in the context of EPCs can be found in (Rittgen, 2001).

Figure 6: Activity state (notation and semantics)
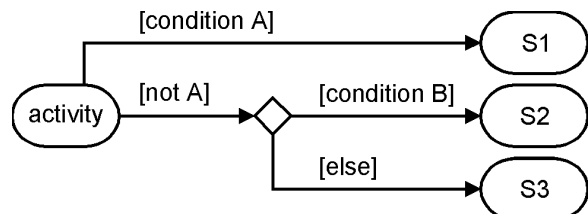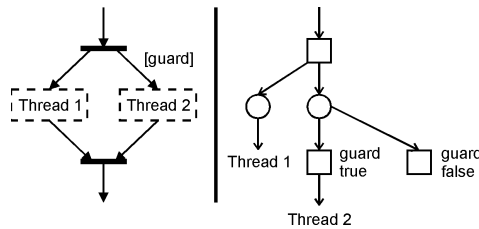

Figure 8: Cascaded branch

Figure 9: Guarded fork (notation and semantics)

OUTLOOK

The introduction of UML provided a standard which not only influenced the modeling of software but also that of businesses. Many methods that were so far primarily focused on the business domain are now being equipped with interfaces to UML. On the other hand early versions of the UML paid little attention to business applications. The use case seemed to be the only UML concept for business modeling and was hence often stretched beyond its original scope, e.g. by employing it to model business processes. Many authors have therefore tried to make use of the powerful UML concept of stereotypes to build application-oriented languages. Contrary to this development the ideas outlined here argue that we need more original support for business-oriented concepts in UML rather than having to specify them as an add-on. The reason for this is that the latter makes it much more difficult to establish a standard for the 'business add-ons' and to integrate the business models into the UML world. We have taken one step in this direction by adding the concept of business processes to UML. Others should follow to make UML a truly unified language for all aspects of developing information systems.

CONCLUSION

Building an information system is a complex, time-consuming and costly endeavour. Its success depends largely on the amount of understanding that can be established between the major players: users, domain experts, software engineers, managers etc. Understanding can be established through models that serve as a basis for the communication of ideas and designs by supporting the individual view of each group of players while at the same time making sure that they really talk about the same model. This requires the views to have a common semantical basis (or meta model) and to translate into each other. In the area of business processes, models have been suggested that support a business view (e.g. EPC) and others that support a software view (e.g. activity diagrams). But instead of being only different views on one and the same model they are actually completely different models that cover a similar area but that do not have a common basis (semantics or meta model). Hence translating between the views is difficult and the resulting communication between the players is weak and subject to misunderstandings which in turn leads to a poor system design.

We have therefore developed Business Process Diagrams which inherit the concepts of both EPCs and activity diagrams and put both on a sound common basis, the Petri net meta model. This facilitates the translation between the views and thus enables players from different groups to discuss the same model while retaining their individual views. In this way differing interpretations or expectations of the system can be identified early and can be prevented from evolving into serious mistakes in the software or disappointment for the users.

REFERENCES

Barros, A., Duddy, K., Lawley, M., Milosevic, Z., Raymond, K., & Wood, A. (2000). Processes, Roles, and Events: UML Concepts for Enterprise Architecture. In A. Evans, S. Kent, & B. Selic (Eds.). UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000. Proceedings. Lecture Notes in Computer Science 1939, Berlin: Springer, pp. 62-77.

Becker, J., & Schütte, R. (1996). Handelsinformationssysteme. Landsberg.

Bruce, T. A. (1992). Designing quality databases with IDEF1X information models. New York: Dorset House.

Chen, R., & Scheer, A.-W. (1994). Modellierung von Prozessketten mittels Petri-Netz-Theorie, Report 107, Institute of Information Systems, University Saarbrücken.

Dehnert, J., & Rittgen, P. (2001). Relaxed Soundness of Business Processes. In K.R.Dittrich, A. Geppert, & M.C. Norrie (Eds.). Advanced Information Systems Engineering, 13th International Conference, CAiSE 2001, Interlaken, Switzerland, June 4-8, 2001. Proceedings. Lecture Notes in Computer Science 2068, Berlin: Springer, pp. 157-170.

Langner, P., Schneider, Ch., & Wehler, J. (1997). Prozessmodellierung mit Ereignisgesteuerten Prozessketten (EPKs) und Petri-Netzen, WIRTSCHAFTSINFORMATIK, 39(5), pp. 479-489.

Nüttgens, M.; Feld, T.; Zimmermann, V. (1998). Business Process Modeling with EPC and UML: Transformation or Integration? In M. Schader, & A. Korthaus (Eds.). The Unified Modeling Language - Technical Aspects and Applications. Heidelberg: Springer, pp. 250-261.

OMG (Ed.) (2000). OMG Unified Modeling Language Specification: Version 1.3, March 2000. Needham: OMG.

Ould, M. (1995). Business processes: modeling and analysis for re-engineering and improvement. Chichester: John Wiley and Sons.

Peterson, J. L. (1981). Petri net theory and the modeling of systems. Englewood Cliffs: Prentice-Hall.

Rational Software, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing, IntelliCorp, i-Logix, IBM, ObjecTime, Platinum Technology, Ptech, Taskon, Reich Technologies, & Softeam (1997). UML Notation Guide: Version 1.1, 1 September 1997. Available: http://www.rational.com/uml [1999, 04-17].

Rittgen, P. (1999). Modified EPCs and Their Formal Semantics, Report 19, Institute of Information Systems, University Koblenz-Landau.

Rittgen, P. (2001). E-Commerce Software: From Analysis to Design. In A. Gangopadhyay (Ed.). Managing Business with Electronic Commerce: Issues and Trends. Hershey: Idea Group, pp. 17-36.

Rosemann, M. (1996). Komplexitätsmanagement in Prozeßmodellen: Methodenspezifische Gestaltungsempfehlungen für die Informationsmodellierung. Wiesbaden: Gabler.

Rump, F. (1997). Erreichbarkeitsgraphbasierte Analyse ereignisgesteuerter Prozessketten, Technical Report 04/97, OFFIS Institute, University Oldenburg.

Scheer, A.-W. (1999). ARIS - Business Process Modeling. (2 ed.). Berlin: Springer.

Simons, A. J. H., & Graham, I. (1999). 30 Things that go wrong in object modelling with UML 1.3. In H. Kilov, B. Rumpe, I. Simmonds (Eds.). Behavioral Specifications of Businesses and Systems, chapter 17. Amsterdam: Kluwer Academic Publishers, pp. 237-257.

Related Content

Use of Bitcoin for Internet Trade
Sadia Khalil, Rahat Masoodand Muhammad Awais Shibli (2018). *Encyclopedia of Information Science and Technology, Fourth Edition (pp. 2869-2880).*
www.irma-international.org/chapter/use-of-bitcoin-for-internet-trade/183998

Weighted and Directed Graph Approaches
 (2018). *Security, Privacy, and Anonymization in Social Networks: Emerging Research and Opportunities (pp. 116-136).*
www.irma-international.org/chapter/weighted-and-directed-graph-approaches/198297

Mobile Enterprise Architecture Framework
Zongjun Liand Annette Lerine Steenkamp (2010). *International Journal of Information Technologies and Systems Approach (pp. 1-20).*
www.irma-international.org/article/mobile-enterprise-architecture-framework/38997

Illness Narrative Complexity in Right and Left-Hemisphere Lesions
Umberto Giani, Carmine Garzillo, Brankica Pavicand Maria Piscitelli (2016). *International Journal of Rough Sets and Data Analysis (pp. 36-54).*
www.irma-international.org/article/illness-narrative-complexity-in-right-and-left-hemisphere-lesions/144705

An Optimal Policy with Three-Parameter Weibull Distribution Deterioration, Quadratic Demand, and Salvage Value Under Partial Backlogging
Trailokyanath Singh, Hadibandhu Pattanayak, Ameeya Kumar Nayakand Nirakar Niranjan Sethy (2018). *International Journal of Rough Sets and Data Analysis (pp. 79-98).*
www.irma-international.org/article/an-optimal-policy-with-three-parameter-weibull-distribution-deterioration-quadratic-demand-and-salvage-value-under-partial-backlogging/190892