DOI: 10.2298/CSIS110111015S

# Formalizing Business Process Specifications

Andreas Speck[1], Sven Feja[1], Sören Witt[1], Elke Pulvermüller[2], and Marcel Schulz[3]

[1] Christian-Albrechts-University Kiel
Olshausenstrasse 40, 24098 Kiel, Germany
{aspe,svfe,swi}@informatik.uni-kiel.de
[2] University of Osnabrueck
Albrechtstr. 28, 49076 Osnabrueck, Germany
elke.pulvermueller@informatik.uni-osnabrueck.de
[3] Intershop Communications AG
Intershop Tower, 07740 Jena, Germany
marcel.schulz@intershop.com

**Abstract.** The behavior of commercial systems is described with business process models. There are different notations and formalism to express business processes. Many of these notations such as BPMN or ARIS EPC models are widely used in commercial projects.
In the paper we focus on formalisms to express rules and specifications for the business processes. Temporal logic in general is a suitable formalism to express rules for dynamic processes. CTL is one kind of temporal logic focusing on branches and paths in particular. With CTL it is possible to formulate rules about different paths in business processes. Since the textual formulae of CTL are not very suitable in the development of commercial systems we introduce a graphical notation (G-CTL) based on the business process notation ARIS EPC. Moreover, we add to the CTL semantics specializers to differentiate between the element types in business process models and provide wildcards which allow the user to check for unknown elements or elements with only partially known properties.

**Keywords:** formal business process rules, temporal logic, model checking, extended graphical-CTL.

## 1. Introduction

Business process models are used to describe the behavior of commercial systems. There are different notations of business process models. Especially the formal business process models may be the base of an automated checking. In order to reach this goal we need also formalisms to express the rules or specifications for the business processes.

Temporal logic in general and the **C**omputational **T**ree **L**ogic (CTL) [7] in particular are promising in order to express business rules and temporal dependencies of business processes. Since CTL focuses on the branches and paths of processes it allows to distinguish between elements in different paths. Moreover, there are well established checking tools like the **S**ymbolic **M**odel **V**erifier (SMV) [39] which may be applied or at least serve as base for extensions [43].

In the following section 2 we examine business process characteristics and the business process modeling. In the following section 3 first CTL as basic notation is introduced and the more suitable graphical notation G-CTL is presented. In section 4 the extensions of the G-CTL (specializers and wildcards) are motivated by checking examples and then presented. Section 5 gives an overview about the related work.

## 2. Business Process Models

### 2.1. Business Process Modeling

There are different types of business process models. For instance, BPMN (**B**usiness **P**rocess **M**odel and **N**otation), ARIS (**Ar**chitecture of integrated **I**nformation **S**ystems) or UML Activity Diagrams are well-known approaches supporting the modeling of business systems in general. However, the basic expressiveness of these model types is quite similar. They provide functions (or activities) and events, various types of connections (like control or sequence flows or associations), splits and joins (mostly combined with logic operators such as `AND`, `OR` or `XOR`) and different elements for further information. This convergence of business process model types allows reducing the models to a formal nucleus: an automaton model. Business process models may be transformed or reduced to states and transitions between the states [38]. Furthermore, such automaton models may be subject of automated checking. Two typical approaches for such transformations may be found in [3] and [42]. [3] transforms the business processes to Petri nets, followed by a transformation into Kripe structures which are then checked. [42] transforms the business processes directly into Kripke structures. The Kripke structures are a base for model checking.

In this paper we focus on the application domain of e-commerce systems in general and in particular on one of the largest standard e-commerce systems: *Intershop Enfinity*. *Intershop Enfinity* is modeled with an ARIS profile *ARIS for Enfinity* [11].

Although, Enfinity-based e-commerce systems are modeled using various model types, we concentrate on the model type mainly used to describe business processes: the **E**vent-driven **P**rocess **C**hains (EPCs). [4]

The EPCs are used to model the business processes on a specific detail level (cf. model elements description in figure 1). EPCs are more concrete than value added chains and present the business aspects of the processes very well. However, they are no concrete implementation models e.g. like UML sequence charts. The EPC models are ideal for the communication between the domain experts (economists) and the computer scientists, since they are still understood by both groups.

[4] Further model types are value added chain models or function hierarchies. These model types are not issue of the paper.

Based on the EPC models the design of the implementation may start. In the case of Intershop Enfinity, executable workflow models (called *Pipelines*) represent the design and are executed by the system's application server.

When the domain experts want to check the business process descriptions of an e-commerce system, this is generally performed on the level of EPC models. Therefore, business rules, regulations and system specific requirements which have to be implemented by the system should be verified on this business process (EPC) level, too. If the EPC models do not represent the required rules and regulations correctly then the resulting system will hardly meet the needs. Therefore it is essential to verify the EPC models.

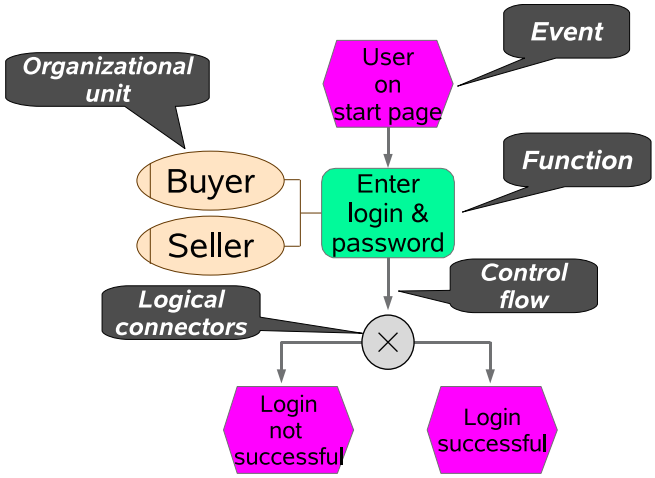



**Fig. 1.** Basic Elements of Event-driven Process Chains (EPCs).

The main elements in the EPC models are (cf. figure 1):

- **Functions** are considered as active elements in EPCs. They describe functionality such as tasks or activities. Functions represent transformations from one state to another, follow-up state. If different follow-up states can occur, the selection of the respective follow-up state can be modeled explicitly by *logical connectors* (as described below). Functions may be refined into another EPC (hierarchical functions). In the EPC model rounded rectangles represent functions.
- **Events** are passive elements which describe the conditions or circumstances which result from *functions* or are triggering the execution of *functions*. An *event* is represented by a hexagon.
- The **control flow** connects *events*, *functions* or *logical connectors* creating a chronological sequence and depicts the logical interdependencies between them. *Control flows* are represented by arrows.

– **Logical connectors** express the logical relationships between elements in the *control flow* (*events* and *functions*). The relationships correspond to the logical operations AND, OR and XOR. Figure 1 depicts the graphical representation of an XOR relationship. The notation elements for AND or OR are similar and with the corresponding Boolean symbol within the circle.
  An XOR in a *control flow* defines a branching point or branch, respectively. There, a decision is required which follow-up state (or path, respectively) is to be taken exclusively. The counterpart of a branch is a merge which means that different branches are merged into one. Branches as well as splits use the same symbol.
  An AND may represent the split or join in the *control flow*. A split activates the outgoing *control flows* in parallel. The join synchronizes incoming *control flows*.
  OR is the weakest relation. An opening OR connector activates one or more control flows and deactivates the rest of them. The counterpart of this is the closing OR connector which activates the *control flow* when at least one of the incoming *control flows* is activated.

Besides these EPC model elements there are several others. A further remarkable element is the **organizational unit** and its assignment which describes the connection between an organizational unit (a person or an organization responsible for a specific function) and the *function* it is responsible for.

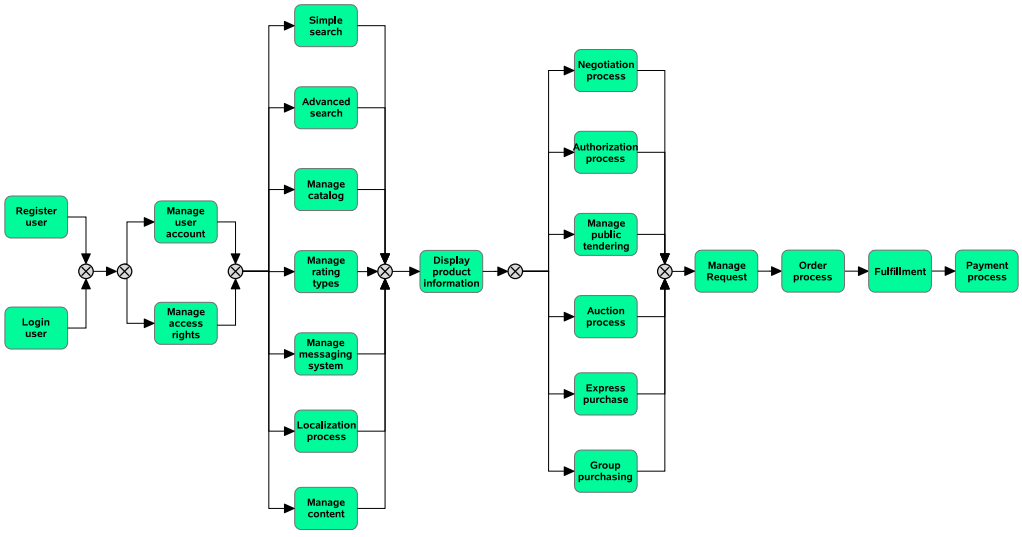


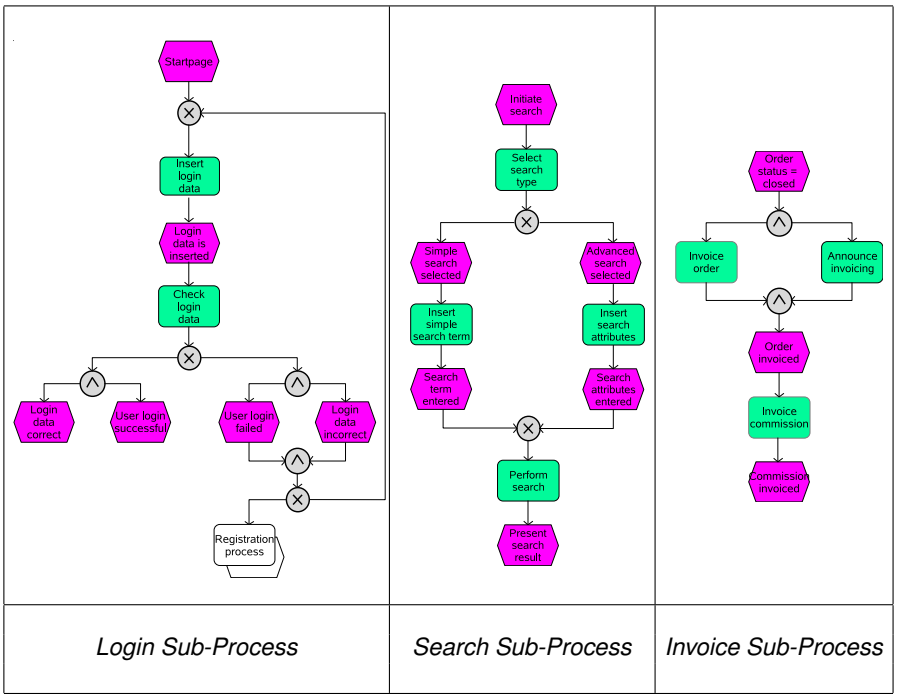

**Fig. 2.** Example of a Function Flow in an eProcurement System.

## 2.2. Commercial Information System Models

As all comparatively large systems commercial information systems may be modeled on several levels of abstraction. This approach is backed by the modeling concepts such as ARIS. Overview models are used to express the major parts of the system on a very abstract level. Figure 2 depicts an abstract function flow. Here, the details of the complete process are hidden.

These comparatively abstract process models may be further detailed and transformed into EPC models with different sub-processes. Finally, we will end up in very detailed sub-process models. In table 1 three examples of detailed sub-processes are presented which are typical for the e-commerce domain: *Login Sub-Process*, *Search Sub-Process* and *Invoice Sub-Process*.

**Table 1.** Detailed Business Process Patterns for eCommerce Systems.




Looking at the representation of processes in a commercial information system then there are mainly two alternatives to present the model: either a large printout as a wall paper or as comparatively small models of sub-processes in a modeling tool (such as ARIS). When we look at the wall paper we have to search manually.

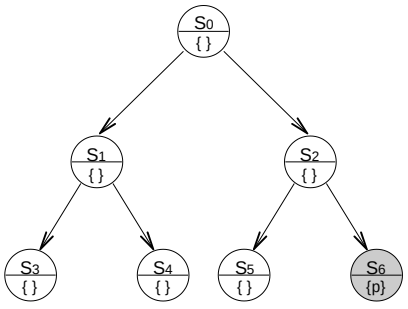
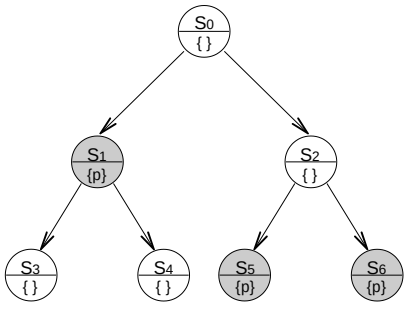
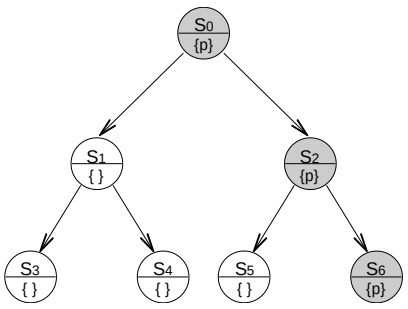
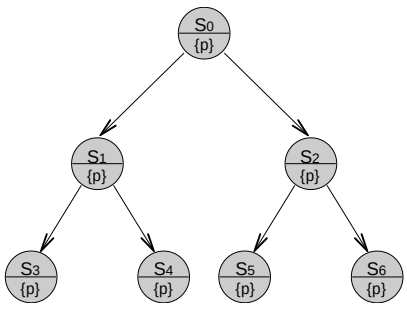
However, in each case it is difficult to keep the overview as well as the knowledge of the (important) details and the interactions between these de-

tails. The verification or checking of such systems is quite hard for human beings and requests automated assistance. An assistance which is able to read rules which have to be fulfilled by the commercial information system and to check the model automatically.

## 3. Formal Specifications

Formal specifications express the rules to be checked. In the following subsections we present temporal logic as base to represent the business rules.

**Table 2.** Examples of some basic CTL Operators `EF`, `EG`, `AF` and `AG` [18].

| | |
|---|---|
|  |  |
| $s_0 \models EF\ p$ | $s_0 \models AF\ p$ |
|  |  |
| $s_0 \models EG\ p$ | $s_0 \models AG\ p$ |

### 3.1. Computational Tree Logic (CTL)

Temporal logic as extension of Boolean logic may be used as formal language to express the rules. **C**omputational **T**ree **L**ogic (CTL) is the logic we use in our research.

As already mentioned CTL is based on Boolean logic:

$$\Phi ::= \bot \mid \top \mid p \mid (\neg\Phi) \mid (\Phi \wedge \Psi) \mid (\Phi \vee \Psi) \mid (\Phi \rightarrow \Psi)$$ [5]

Additionally, there are temporal operators in CTL. These operators are called *Temporal Connectives* and are used pairwise:

$$\Phi ::= AX\Phi \mid EX\Phi \mid A\,[\Phi\ U\ \Psi] \mid E\,[\Phi\ U\ \Psi] \mid AG\Phi \mid EG\Phi \mid AF\Phi \mid EF\Phi$$

$A$ means *Always*
$E$ means *Eventually*
$G$ means *Globally*
$X$ means *Next*
$U$ means *Until*

Table 2 shows four general CTL operators. $EF\Phi$ means that $\Phi$ potentially holds. $AF\Phi$ means that $\Phi$ will occur on each potential path. $EG\Phi$ expresses that $\Phi$ is true in each states of one complete path. $AG\Phi$ is an invariant: $\Phi$ is true in each state [16].

Table 2 omits the operator pairs with *Next* and *Until* operators. Examples for these operator pairs are the following:
$EX\Phi$ means that there is a path in which in the next state $\Phi$ holds.
$AX\Phi$ means that in all paths in the next state $\Phi$ becomes true.
$E(\Phi U\Psi)$ means that there is a path in which $\Phi$ is true until $\Psi$ holds.
$A(\Phi U\Psi)$ means that in all paths $\Phi$ is true until $\Psi$ becomes true.

When we apply CTL for expressing rules for business processes these may look like the examples below:

- *Customer* may always *Catalog Browse*
  `AF Catalog_Browse`
  (**A**lways in the **F**uture *Catalog Browse*)
- There is a path to *Product Search*
  `EF Product_Search`
  (it **E**xists in the **F**uture *Product Search*)
- *Centralized Buyer* will get a *Personal Content* and *Personalized Offer*
  `AG (Customer_is_Centralized_Buyer ->`
  `AF (Personal_Content ∧ Personalized_Offer))`
  (**A**lways **G**lobally if *Customer is Centralized Buyer* is true implies that **A**lways in the **F**uture *Personal Content* and *Personalized Offer* will be true)
- *User not logged in* until *User login successful*
  `AG (¬User_logged_in U User_login_successful)`
  (**A**lways **G**lobally *User logged in* is false **U**ntil *User login successful*)

[5] $\Phi \rightarrow \Psi$ is a logic implication.

– *Login Data is inserted* next *Login Data Check*
`AG (Login_Data_is_inserted -> AX Login_Data_Check)`
(**A**lways **G**lobally *Login Data is inserted* **A**lways ne**X**t state will be *Login Data Check*)

### 3.2. Visualization of Graphical Specifications

Considering the examples in the previous section the CTL formula does not look very convenient. The likelihood that business modelers will accept the temporal CTL formulae is rather low. This leads us to a more suitable notation for the business modeling community: the graphical representation of CTL on base of EPC models: the *Temporal Logics Visualization Framework* (TLVF). TLVF describes how graphical specifications of the rules are derived from the business process models [23]. Our business process models are EPCs. However, the graphical notation may also be applied on other notations such as BPMN.

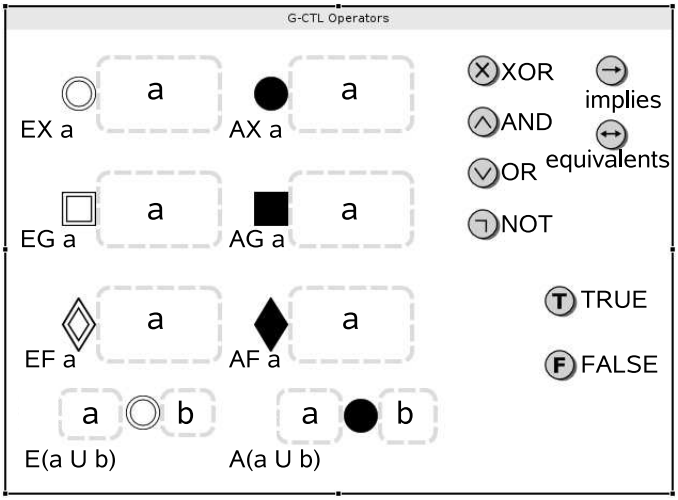
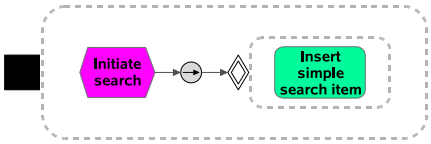



**Fig. 3.** G-CTL Operators.

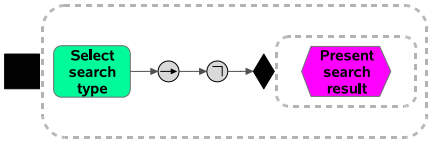
The EPC-based definition of the TLVF language elements of **G**raphical **CTL** (G-CTL) is shown in figure 3. G-CTL operators are based on CTL operators [17], [7]. Like CTL (introduced in the previous section 3.1) there are two types of operators which are combined pairwise: Path quantifiers always (**A**) and exists (**E**) which indicate the occurrence within a path. The temporal operators determine the temporal order. The G-CTL temporal operators are: in the future (**F**), globally (**G**), next (**X**) and until (**U**). Examples for pairwise combinations are: **AG** always globally or **EX** exists next.

These operators are represented by graphical symbols which may be combined with EPC notation elements in order to describe a specification.

**Table 3.** Examples of G-CTL Specification Patterns in the Search Sub-process.




CTL format: $AG\ Initiate\ search\ \rightarrow\ EF\ Insert\ simple\ search\ item$




CTL format: $AG\ Select\ search\ type\ \rightarrow\ !\ AF\ Present\ search\ result$

An example of two simple rules formatted in G-CTL graphical notation are depicted in table 3. Boolean logic operators (the implication in this case) are represented by symbols which may be connected with other operators (logical or temporal logical operators) and model elements such as events or functions. Temporal logical operators like the AG (always globally) and EF (exists in the future) are realized as containers since they embrace a sub-formula or element as operand.
The informal semantics of the two rules is:

1. Rule (top of table 3): Always globally it has to be true that when the event *Initiate search* has occurred (became true) there exists in the future the function *Insert simple search item* (logical implication).
   Or in other words: When the event *Initiate search* occurred in the following of the process there must be at least one branch with the function *Insert simple search item*.
2. Rule (below): Always globally it has to be true that if the function *Select search type* has occurred (became true) always in the future the event *Present search result* becomes not true (logic implication).

Or in other words: When the function *Select search type* occurred (which means that the search function is activated by the user) in the following of the process there is a branch in which the event *Present search result* does not occur. This means that we are looking for counterexamples of actually desired behavior: We want to assure that at least an empty search result is presented to the user when s/he has initiated a search.

## 4. Enhanced Checking

The visualization of rules is part of the user front end. The checking algorithms of the checking system are another issue. Basically, we rely on the CTL model checking algorithms e.g. like realized in the SMV model checker [39].

However, before we are able to apply model checkers (or any other checking concepts) we have to transform the models (the EPC business process models in our case) and the specifications to the formal representations used by the checkers.

The result of a checking is either that the specification is fulfilled or violated. If the model is correct according to the specification only the notification of ”true” is reported. In an error case the model checker answers with a textual description of a counter example. The result may be presented in the format of a textual description as in [26] .

### 4.1. Extended Model Checking

In general, model checkers need automata models as input for the checking procedure. Actually, the automata models are represented in a specific structure – the Kripke structure. Kripke structures may be considered as a specific expression of ordinary automata representations [7].

If we consider a direct transformation of the EPC models we might transform the elements **event** and **function** directly into states which are connected according to the control flow. This straight-forward approach is not necessarily wrong. In some cases it is sufficient and temporal specifications may be checked in a correct manner in compliance to the semantics.

However, there may be cases in which we would like to distinguish between the different model elements when we develop a specification. We propose an extension of the CTL notation with specializers which characterize the specific model elements.

Two examples in which a specification without distinction between the element types leads to an error are presented in figure 4. Both examples are supplemented with textual specifications. In both examples the upper specification is without additional specializers and the lower specification makes use of the two additional specializers *F* and *E* [6].

[6] These additional specializers are not to be confused with the operators *Future* and *Exists* which are always used in a pairwise manner.

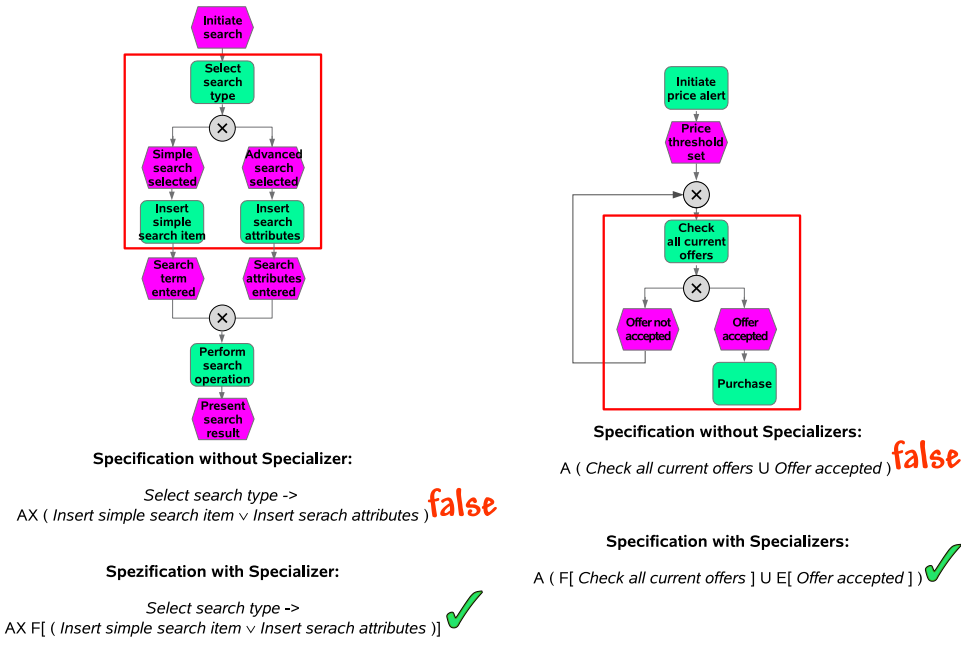



**Fig. 4.** Specializers in Temporal Logic.

1. In the example on the left upper specification (without specializers) of the model at the left requires that directly after the function *Select search type* the functions *Insert simple search item* or *Insert search attributes* have to follow. The model on which this rule is applied is the already known search example. Such a specification or rule may be defined in the situation when we would like to keep the denomination of the events after *Select search type* open (e.g. for customizing at design time) and are only interested that they are followed by standard functions (such as *Insert simple search item* or *Insert complex search item*). The second specification contains the specializer *F* (for function) directly after the *Always neXt* (AX). This indicates that only function elements have to be considered in the checking. An event (or an element of another type) is ignored. This specification is true (as we would expect it in the domain semantics).
2. The example on the right side of figure 4 contains a loop. The process is a price alert process. If a price falls below (or rises upon) a certain threshold there is a price alert and the system purchases.
   It may be of interest if the process *Check(s) all current offers* is performed until the price threshold is met and the *Offer (is) accepted*. The first specification without specializers turns out to be false although the model meets the requirement. When we use the specializers then the specification is correct. In our example *Always* the function *Check(s) all current offers* (due to

the specializer only functions are considered) *Until* the event *Offer accepted* becomes true. The event *Offer not accepted* is in the loop back to the function *Check all current offers*. Since *Offer not accepted* is an event and not a function as indicated by the specializer the checker does not care about it.

With this specializer concept in temporal logic specifications we are able to select specific element types and focus on these. This is an extension of the temporal logic CTL we call ECTL1 (Extended CTL). In order to handle the specializers the algorithm of the model checker has to be modified. A more detailed description of the modified checking algorithm may be found in [43].

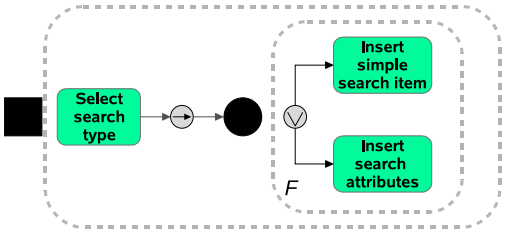



**Fig. 5.** Example of graphical Representation of Specializers (EG-CTL).

An example of a graphical representation of **E**xtended **G**raphical-CTL (EG-CTL) specification is depicted in figure 5. In the figure the rule of the left example in figure 4 is a little improved: An AG operator is added:
$AG\,(Select\ search\ type\ \rightarrow\ AX\ F[\,(Insert\ simple\ search\ item \vee Insert\ search\ attributes)])$

Due to the specializers the specification may be more precise. The expressiveness of the temporal logic is extended and captures different types of elements. This leads to the question to introduce uncertainty in the way of using wildcards in specifications.

### 4.2. Wildcards

The previous specifications of rules require that we know there must be a certain function or an event occurs at a specific moment. In other cases the explicit expected element is not clear. Figure 6 depicts an example: the payment process with some alternative payment functions. This is one possible implementation of the payment process. However, it is up to the wishes of the later shop owner which payment function is realized. E.g. in our example the payment via Pay Pal is not considered.

If we use wildcards we are able to specify rules which expect specific element types of business process models not knowing the explicit element. It is most likely that all web shops use a specific payment function or a set of

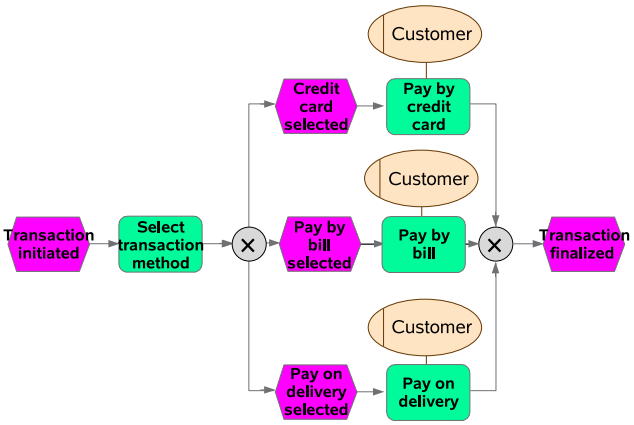



**Fig. 6.** Payment Process with alternative Payment Functions.

payment functions or at least in case of a free download a specific interaction function with the customer. We may express this function which is at the time of rule specification unknown by a wildcard. Of course a completely open wildcard may be critical since it may be meaningless. However, in our example we know that it is important to interact with the customer, that the function is customer driven. The relation of the unknown function to the organization customer expresses this.

The wildcard functionality may be expressed as:

$$AG\ (\ E[\ Transition\ initiated\ ]\ \rightarrow\ AF\ F[\ *\ \wedge\ O[Customer]\ ])$$

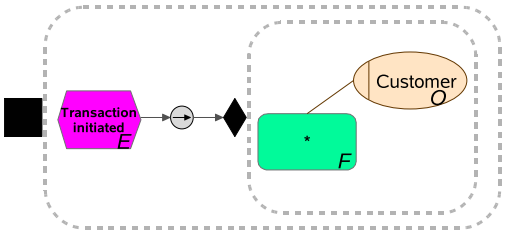



**Fig. 7.** Wildcard in Payment Function Rule.

The graphical representation of the rule is shown in figure 7. The specializers are indicated by a capital *E*, *F* and *O* [7]. The asterisk character symbolizes the wildcard.

[7] The *O* specializer represents an organizational unit which is by default connected to its function by a logical AND.

## 5. Related Work

Software models have been issue of verification like model checking for a rather long period. Examples for early approaches based on model checking are [21] or [39].

However, base of all the checking is the formalization of business process models like [32] (graph grammar based approach) and [8] which enable to apply formal methods for business processes. The transformation of EPC models to Petri nets also formalize them ([35] and [36]). In this case the semantics are restricted. The formalization proposed in [30] uses a fix-point-semantics-based definition of the semantics of EPCs which is also used for model checking.

The mapping of business process models on ASM (Abstract State Machines) allows to operate the business processes on an abstract level [9]. Many approaches (also the majority of the here referenced approaches) consider BPMN. However, executable models like BPEL are object of formalization as well [13]. Further formalization approaches are based on the pi calculus [37]. An issue of research to be addressed by formalization approaches are the joins after branching [8]. In the EPC model [45] the semantics as base of the formalization have been analyzed e.g. by [2] and [40]. An example for the BPMN analysis may be found at [28].

Examples for approaches employing model checking on business processes are [22], [33], [5] or [6]. [33] evaluates different checking technologies for being applied on business processes. [5] and [6] focus on the aspect of transactions in e-commerce systems. In [22] a large number of business processes have been investigated and different checking concepts are applied. One important conclusion is that several concepts could be combined in order to improve their effect.

An example of an approach for the verification of business process systems based on Petri nets is [19] using BPMN. With Petri nets the business processes are mapped to the Petri net elements similar to Kripke structure mapping. An alternative approach for Petri net based verification is based on bi-simulation and algebraic solutions (e.g. [41]).

The approach presented in this paper relies directly on the push-button model checking technology and temporal logic requirement specifications. Most approaches applying formal methods to business process models for the purpose of checking use straight-forward model transformations. These transformations result in a loss of information and, therefore, verification precision. The reason is the incompatible semantics of the business process models and the verification models which causes several problems resulting in different alternative approaches to tackle them [20]. Moreover, additional information (such as organizational units in EPC models) is lost during the transformation due to a surjective mapping. Two approaches transforming business process models to verification models are [42] (SMV Kripke structures) or [1] (Petri nets).

An approach which proposes a graphical representation of models and specifications is [26]. In this approach the business process notation are UML activ-

ity diagrams and the result of the LTL-based checking is presented in a textual manner.

In the domain of formal methods approaches may be found which concentrate on an increase of semantic expressiveness of the specification languages (e.g. the $\mu$-calculus [10] and [34] or in the multi-valued logic research as in [15]). Extensions to the temporal logic for LTL have been proposed in [14] or [29], for instance. In these approaches a link to software models or business process models is missing and the general idea of a specialization on different model elements is not considered. In contrast to [14], [29], [27] and [31] we are able to explicitly distinguish and mix specializers (and thus views) for different model elements.

## 6. Conclusions and Future Work

Most business process models are very large and consist of a large number of elements and flows. The checking of these large models by hand is time consuming and still not satisfying in all cases [8].

Formal verification methods may help to automate at least some kinds of checking (e.g. routine checks). The formal method we apply in our work is model checking. The rules to be verified must be able to express the temporal relations between the process elements (e.g. the control flow).

Due to this we use the temporal logic CTL (Computational Tree Logic) as a base. However, the pure CTL has some drawbacks.
It does not support a graphical notation and the semantics of CTL may be extended. We present G-CTL as a graphical notation of CTL supporting a similar element notation than EPC models and combining them with the temporal operators of CTL.
Due to the lack of expressiveness of CTL we extend the semantics of CTL by specializers (Extended Graphical-CTL, EG-CTL). These specializers allow distinguishing between different model element types (in our case EPC events, functions, organizational units and others). With the help of the specializers it is possible to check on the existence of a specific type in the process not considering elements of other types. In order to support the specializers the checking algorithm has been modified. Moreover, we introduced wildcard which allows defining a rule in a moment when the concrete modeling of a process is not clear. The wildcards keep the position in the process open. Nevertheless, by some additional information, e.g. the knowledge which concrete organizational unit will be related to the unknown element represented by a wildcard it is possible to complete the rule.

[8] The checking of large models may result in state explosion problems. However, there are different approaches to deal with this problem. One used by most model checking tools is to optimize the model structure by applying **O**rdered **B**inary **D**ecision **D**iagrams (OBDD) [12]. Other approaches are abstraction or partial evaluation and compositional model checking [25]. These different approaches are known to the authors and taken into account. Although, these approaches are not issue of the paper.

The different techniques of the graphical representation of specifications and the extension of CTL in order to represent different model element types as well as the introduction of wildcards are integrated in the Business Application Modeler (BAM) in order to improve the usability of this business process model checking concept.

At the moment we are developing a presentation tool on the base of Eclipse the **B**usiness **A**pplication **M**odeler (BAM) [24]. In detail, BAM is based on the Eclipse Graphical Editing Framework (GEF) [4]. GEF has been chosen since it supports the required presentation functions and is comparatively portable which means that the BAM editor runs on different operating system platforms. The goal of this Eclipse-based implementation is a high degree of portability and the ability to integrate transformation and checking systems as simple as possible.

In our future work we intend to support further notations (e.g. BPMN). Already now we are working on an i* model support [44]. The interoperability between our BAM and other modeling tools like ARIS or ViFlow has not yet been realized. An intermediate (probably XML-based) data format may be useful.

## References


1. van der Aalst, W.M.P.: Formalization and Verification of Event-driven Process Chains. Information and Software Technology 41(10), 639–650 (1999)
2. van der Aalst, W.M.P., Desel, J., Kindler, E.: On the semantics of EPCs: A vicious circle. In: EPK 2002 - Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, Proceedings des GI-Workshops und Arbeitskreistreffens (Trier, November 2002). pp. 71–79 (2002)
3. van der Aalst, W.M.P.: Formalization and Verification of Event-driven Process Chains. Information and Software Technology 41(10), 639–650 (1999)
4. Anders, E.: Modellierung und Validierung von Prozessmodellen auf Basis variabler Modellierungsnotationen und Validierungsmethoden als Erweiterung für Eclipse, Diploma Thesis (2010)
5. Anderson, B.B., Hansen, J.V., Lowry, P.B., Summers, S.L.: Model checking for design and assurance of e-Business processes. Decision Support Systems 39(3), 333–344 (2005)
6. Anderson, B.B., Hansen, J.V., Lowry, P.B., Summers, S.L.: The application of model checking for securing e-commerce transactions. Communications of the ACM 49(6), 97–101 (2006)
7. Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P.: Systems and Software Verification – Model-Checking Techniques and Tools. Springer, Berlin, Germany (2001)
8. Börger, E., Sörensen, O., Thalheim, B.: On Defining the Behavior of OR-joins in Business Process Models. The Journal of Universal Computer Science (J. UCS) 15(1), 3–32 (2009)
9. Börger, E., Thalheim, B.: Modeling Workflows, Interaction Patterns, Web Services and Business Processes: The ASM-Based Approach. In: Proceedings of Abstract State Machines, B and Z, First International Conference (ABZ 2008). pp. 24–38. Springer LNCS 5238 (2008)

10. Bradfield, J., Stirling, C.: Modal logics and mu-calculi: an introduction. In: Handbook of Process Algebra, pp. 293–33. Elsevier Science Publishers (2001)
11. Breitling, M.: Business Consulting, Service Packages & Benefits. Tech. rep., Intershop Customer Services, Jena (2002)
12. Bryant, E., R.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers C-35(8), 677–691 (1986)
13. Cámara, J., Canal, C., Cubo, J., Vallecillo, A.: Formalizing WSBPEL Business Processes Using Process Algebra. Electronic Notes in Theoretical Computer Science 154(1), 159–173 (2006)
14. Chaki, S., Clarke, E.M., Ouaknine, J., Sharygina, N., Sinha, N.: State/Event-Based Software Model Checking. In: Proceedings of the 4th International Conference on Integrated Formal Methods (IFM). pp. 128–147. Springer, LNCS 2999 (2004)
15. Chechik, M., Devereux, B., Easterbrook, S., Gurfinkel, A.: Multi-Valued Symbolic Model-Checking. ACM Transactions on Software Engineering Methodology 12(4), 371–408 (October 2003)
16. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. ACM Transactions on Programming Languages and Systems 8(2), 244 – 263 (April 1986)
17. Clarke, E.M., Grumberg, O., McMillan, K.L., Zhao, X.: Efficient generation of counterexamples and witnesses in symbolic model checking. In: DAC '95: Proceedings of the 32nd ACM/IEEE conference on Design automation. pp. 427–432 (1995)
18. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge, Massachusetts; London, England, 3 edn. (2001)
19. De Backer, M., Snoeck, M.: Business Process Verification: a Petri Net Approach. Tech. rep., Catholic University of Leuven, Belgium (2008)
20. van Dongen, B.F., Jansen-Vullers, M., Verbeekm, H.H.M.W., van der Aalst, W.M.P.: Verification of the SAP reference models using EPC reduction, state-space analysis, and invariants. Computers in Industry 58(6), 578–601 (2007)
21. Emerson, E.A., Clarke, E.M.: Characterizing Correctness Properties of Parallel Programs Using Fixpoints. In: ICALP 1980, Automata, Languages and Programming, 7th Colloquium. pp. 169–181. Springer LNCS 85 (1980)
22. Fahland, D., Favre, C., Jobstmann, B., Köhler, J., Lohmann, N., Völzer, H., Wolf, K.: Instantaneous Soundness Checking of Industrial Business Process Models. In: Proceedings of the 7th International Conference on Business Process Management (BPM 2009). pp. 278–293. Springer, LNSC 5701 (2009)
23. Feja, S., Fötsch, D.: Model Checking with Graphical Validation Rules. In: Proceedings of the 15th IEEE International Conference on the Engineering of Computer-Based Systems (ECBS 2008). pp. 117–125. IEEE (2008)
24. Feja, S., Speck, A., Witt, S., Schulz, M.: Checkable Graphical Business Process Representation. In: Proceedings of the 14th East-European Conference on Advances in Databases and Information Systems (ADBIS 2010,). pp. 176–189. Springer, LNCS 6295 (2010)
25. Fisler, K., Krishnamurthi, S.: Decomposing Verification by Features. In: IFIP Working Conference on Verified Software: Theories, Tools, Experiments (October 2005)
26. Förster, A., Engels, G., Schattkowsky, T., Van Der Straeten, R.: Verification of Business Process Quality Constraints Based on Visual Process Patterns. In: Proceedings of the First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering (TASE '07). pp. 197–208 (2007)
27. Giannakopoulou, D., Magee, J.: Fluent Model Checking for Event-based Systems. In: Proceedings of the 9th European Software Engineering Conference (ESEC) held

jointly with 10th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE). pp. 257–266. ACM Press (2003)
28. Grosskopf, A.: xBPMN. Formal control flow specification of a BPMN based process execution language, Master's thesis (2007)
29. Jonsson, B., Khan, A.H., Parrow, J.: Implementing a Model Checking Algorithm by Adapting Existing Automated Tools. In: Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems. pp. 179–188. Springer, LNCS 407 (1989)
30. Kindler, E.: On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In: Proceedings fo Business Process Management: Second International Conference, (BPM 2004). pp. 82–97. Springer LNCS 3080 (2004)
31. Kindler, E., Vesper, T.: ESTL: A Temporal Logic for Events and States. In: Proceedings of the 19th International Conference on Application and Theory of Petri Nets (ICATPN). pp. 365–384. Springer LNCS 1420 (1998)
32. Klauck, C., Müller, H.J.: Formal business process engineering based on graph grammars. International Journal on Production Economics 50, 129–140 (1999)
33. Köhler, J., Tirenni, G., Kumaran, S.: From Business Process Model to Consistent Implementation: A Case for Formal Verification Methods. In: 6th International Enterprise Distributed Object Computing Conference (EDOC 2002). pp. 96–106 (2002)
34. Kozen, D.: Results on the propositional mu-calculus. Theoretical Computer Science 3(27), 333–354 (December 1983)
35. Langner, P., Schneider, C., Wehler, J.: Prozeßmodellierung mit ereignisgesteuerten Prozeßketten (EPKs) und Petri-Netzen. Wirtschaftsinformatik 39(5), 479–489 (1997)
36. Langner, P., Schneider, C., Wehler, J.: Petri net based certification of event-driven process chains. In: Proceedings of Application and Theory of Petri Nets 1998, 19th International Conference (ICATPN '98). pp. 286–305. Springer, LNI 1420 (1998)
37. Ma, S., Zhang, L., He, J.: Towards Formalization and Verification of Unified Business Process Model Based on Pi Calculus. In: Proceedings of the 6th ACIS International Conference on Software Engineering Research, Management and Applications (SERA). pp. 93–101. IEEE Computer Society (2008)
38. Mahleko, B., Wombacher, A.: Indexing Business Processes based on Annotated Finite State Automata. In: IEEE International Conference on Web Services (ICWS 2006). pp. 303–311. IEEE Computer Society, Los Alamitos, CA, USA (2006)
39. McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers (1993)
40. Mendling, J., Neumann, G., van der Aalst, W.M.P.: Understanding the Occurrence of Errors in Process Models Based on Metrics. In: On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007. pp. 113–130 (2007)
41. Morimoto, S.: A Survey of Formal Verification for Business Process Modeling. In: ICCS 2008, 8th International Conference. pp. 514–522. Springer LNCS 5102 (2008)
42. Pulvermüller, E.: Composition and correctness. Electronic Notes in Theoretical Computer Science (ENTCS) 65(4) (2002)
43. Pulvermüller, E.: Reducing the Gap between Verification Models and Software Development Models. In: The 8th International Conference on Software Methodologies, Tools and Techniques (SoMeT 2009). pp. 297–313. IOS Press (2009)
44. Rusnjak, A., El Kharbili, M., Hristov, H., Speck, A.: Managing the Dynamics of E/mCommerce with a Hierarchical Overlapping Business-Value-Framework. In: 24th

IEEE International Conference on Advanced Information Networking and Applications Workshops (AINA Workshops), WAINA 2010. pp. 461–466. IEEE Computer Society (2010)
45. Scheer, A.W.: ARIS - Modellierungsmethoden, Metamodelle, Awendungen. Springer, Berlin, Germany (1998)

**Prof. Dr. Andreas Speck** is head of the ”Business Information Technology” research group at Christian-Albrechts-University of Kiel, Germany. Previously he headed the research group ”Application Systems (Software Engineering and eCommerce)” at the Friedrich-Schiller-University of Jena and led the research group of the Intershop Communications AG at Jena. His main research interests are the modeling and verification of commercial application systems and electronic and mobile commerce systems. Andreas Speck is member of the German Computer Society (GI).

**Sven Feja** studied Business Informatics at the Friedrich-Schiller-University of Jena, Germany, and received his degree in Business Informatics in 2006. After his graduation he joined the research group ”Application Systems (Software Engineering and eCommerce)” at the Friedrich-Schiller-University of Jena. Currently he is a research assistant at the Christian-Albrechts-University of Kiel, Germany, researching in the field of business process modeling and validation and verification of correctness of process models (including functional and non-functional aspects). Sven Feja is member a member of the German Computer Society (GI).

**Sören Witt** studied Computer Engineering at the Christian-Albrechts-University of Kiel, Germany. He received his degree as Computer Engineer in 2009. After his graduation he joined the ”Business Information Technology” research group at Christian-Albrechts-University of Kiel as research assistant. Soeren Witt is researching in the area of business process model verification and validation on basis of graphically represented specifications.

**Prof. Dr.-Ing. Elke Pulvermüller** is a professor (Jun.Prof.) in the Department of Mathematics & Computer Science at the University of Osnabrueck, Germany. There, she is head of the Software Engineering research group. Between September 2009 and March 2010 she has been temporarily appointed as an Acting Full Professor of the Institute of Software Technology and Programming Languages at the University Luebeck, Germany. Previous to her appointments at Luebeck and Osnabrueck she has been a senior researcher / research assistant at the University of Luxembourg (2006 - 2007), at the Friedrich Schiller-University of Jena (Germany) and at the Universitaet Karlsruhe (Germany). She received her doctoral degree from the Friedrich Schiller-University of Jena in 2006. Her research focuses on new approaches in software and quality engineering. Elke Pulvermueller is a member of the German Computer Society (GI) and the ACM.

**Marcel Schulz** studied Business Informatics at the Friedrich-Schiller-University of Jena, Germany, and received his degree in Business Informatics in 2009. He gained interactional experience as project manager for commercial information systems in Shanghai, China working for American customers from 2007 till 2009. Currently he is member of the Intershop Commuincations research group. His research interests are business intelligence, data mining and simulation.

*Received: January 11, 2011; Accepted: May 5, 2011.*