Process wizard for information systems

Mgr. Jaroslav Procházka

Abstract: *The new approach in information system automation is process or workflow management. For not skilled users is important when the business processes of company are described. Then, according to this description, are users led correctly in their work. The business (application) model can be caught in finite state machines. FSM can be then used as model in process wizard using model-view-controller architecture. We explain what process wizard is, what should contain and outline how it could be implement in IS QI.*

Keywords: *process, workflow, wizard, process wizard, FSM, Petri nets, MVC.*

Recent state and our objectives

In recent state the process support in information systems is not so spread. There exist several solutions on market, where user can define company's processes and this definition joins with information system's functions. Such system can show, what was previous step in process or user can see what are possible following steps according to current process state. One of these tools is IS SAP and process tool ARIS, other one is e.g. Baan IS. QI IS also contains support for process management. Useful tool for process (workflow) management is process wizard. This tool can lead user through whole process according to his operations and current system states.

My doctoral thesis deals with process wizard design and with generation source code of modeled macro language of IS QI, so some practical samples are shown in IS QI, but majority of appointed ideas is usable generally. This paper is promoted by internal grant IGA 03/2005 and by research intention VZ MSM 6198898701.

QI and processes

As said, QI information system contains also implementation of workflow management. Nowadays, in QI information system is implemented process management without possibility of automatic IS functions call. We briefly introduce technological solution of QI system function (form) implementation. Data is stored in database. This database is only storage; it does not implement any functions. Stored data does not make sense at all, for the sake of security. The most important part is application server (AS) – here is data collected together. AS consists of data interface, object server and stores application logics as well. Application forms display user data. Important component, which defines data in form, is called data set (DS). It is user defined selection set and represents part of application logics as well.

Win client (exe file) comprises general programmatic functions implementation (it means general form, general report – so called functional objects). DB stores not only user (application) data, but also concrete form and report definitions – their size, position, included components (such as buttons, bookmarks, fields, …) and connected data sets. Before the called form is opened, the general function of win client has to ask application server for these definition data. This implicates, that called form must exist, must be created first. Situation depicts figure 1.

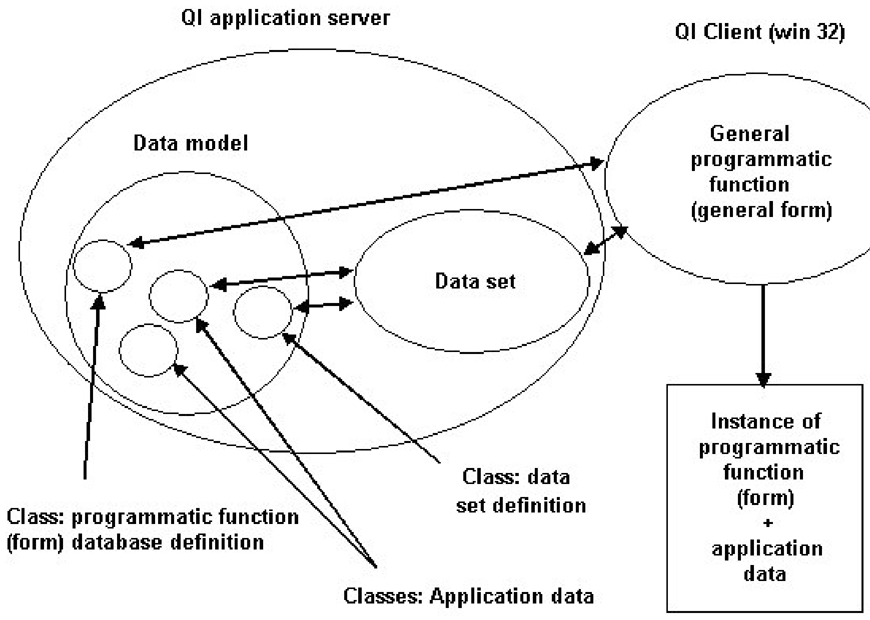
QI application server
QI Client (win 32)
Data model
General programmatic function (general form)
Data set
Instance of programmatic function (form) + application data
Class: data set definition
Class: programmatic function (form) database definition
Classes: Application data

Fig. 1: QI application technological construction

If we want to automate the process as deep, that the system functions will be opened (generated) automatically, there have to exist several variants of given functions (forms) for each branch of process or for different processes. Let us introduce an example. One bill form with appropriate checks and data fields is necessary for insertion to system. Other one (with quite different checks and fields) we need for supervisor's authorization. It denotes, except complete process description, to develop all possible variants of programmatic functions (forms and reports) that can be used in processes. Such approach is neither effective nor practicable. Besides, any change in process definition brings revision of all programmatic functions supporting this process. Therefore is our aim to design wizard to generate called programmatic functions automatically according to given templates or patterns with possibility of user modification (what data can user see and use).

Process wizard

This chapter introduces what the wizard is, and what are its features. The name could in reader invoke mighty capability, but in IT, the wizard is usually called a program or a component, which helps user or developer to create or finalize some document, application class, application component, form or anything else. This is done step by step and it has its beginning and its end.

We can define software wizard as component with following qualities:

- All wizard data compose a single transaction.
- Steps are processed in sequence from given beginning to given end.
- There exist one starting step, several intermediate steps, and one ending step.
- The wizard validates its state before advancing to the next step.
- There can be several different paths to reach the ending step.
- It is possible to navigate back to review and update values entered in previous steps.
- A wizard can be cancelled before completion.

QI process wizard should lead user through whole process or through its important part. Every form is based on given data set (DS), if there is more than one DS included in form, there should exist hierarchy or synchronization of data sets. These hierarchies and synchronizations have to be included into wizard as well. From facts written above results fundamental assumptions for process wizard. For dynamical form generation according to valid process, we need:

- Process description – description of application domain, for this purpose can be used Petri net or finite state machine.
- User defined data – this is, what user want to see, update or validate.
- List of relevant DS (which can be used for given form) including their hierarchy and possible synchronizations.
- Storage for application data and for model data (process description) as well.

We come up from stored process description, user defines application data that he/she wants to work with, then is generated (dynamically created) form that matches all given requirements. Storing mechanisms are already included in IS QI and also are in all other IS.

MVC model

To insert concepts, said in this paper, to IT architecture, we use really known MVC model. This technology was first used in Smalltalk programming language class library. It is typically used in user interface (UI) programming, and it is also known from Java Swing class library. Though this technology is older than 20 years, it is still often used in IS and web projects. As said, Java Swing library implements MVC model, widely spread Jakarta Struts framework is based on it as well. Basic technology feature is implementing program component using three following classes:

- *View* – this object represents visual graphical component.
- *Controller* – object catches inputs via graphical component and reacts on it (perform action on model and view). Controller is binding between model data and their graphical representation – view.
- *Model* – object represents graphical component data (domain model).

Using MVC technology for implementing user interface, it is common, that model is shared simultaneously by more than one graphical component, e.g. the same data on user form could be represented in a textual form (grid) but also in a graphical form (as a tree). When user changes data by editing table cell, controller guarantee given change in shared model and its graphical representation (the tree) using view.

In our case, the wizard, user works with, is a view (the graphical component). Wizard can contain also controller, which handles all user inputs and perform these changes on model.

When we come back to the beginning of this chapter and read once more the 7 points, which define wizard, we realize, that model of wizard should be a directed acyclic graph with weighted edges. Model represents this graph with a modified adjacency list. Each node of the graph corresponds to one wizard step. A node either refers to or directly stores domain data relevant to the step. Besides that, a node holds references to the adjacent nodes.

Directed graph G is a pair $G = (V, E)$, where

$V = \{v_1, v_2, \ldots, v_n\}$ is a set of points called vertices or nodes
$E = \{e_1, e_2, \ldots, e_m\}$ is a set of lines or curves called edges.

A directed acyclic graph is a graph with no directed cycles. That is, for any vertex *v*, there is no directed path starting and ending on *v*.

When only graph does not suffice for adequate system description, it is possible to use number or string value assigned to edge or vertex – so called weight. This can represents the duration or cost of activity, probability of activity or strict condition. When all edges are weighted, we can describe every path through graph. There exist several ways, how to describe it. One is a list of names of weighted edges; the second one is a matrix or a table. If there exists an edge between two vertices in a graph, we write down weight to the given table cell (see two examples, table 1 and 2).

	rightLogin	wrongLogin	logout
Start (S)	rightLogin, L	wrongLogin, E	
Login (L)			logout, E
End (E)			

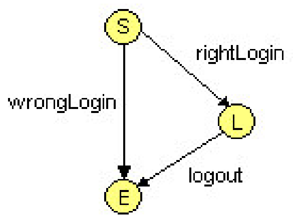

Table 1.: Representation of graph paths

	Start (S)	Login (L)	End (E)
Start (S)		rightLogin	wrongLogin
Login (L)			logout
End (E)			

Table 2.: Another representation of graph paths

It is important to stress, that terminology of graph theory is not unified, so above written definitions can be little bit different from these reader knows [De02].

Process description

It is possible to describe application domain or process transitions using finite state machines (FSM) or Petri nets. These are strictly formal and have mathematical apparatus, thus they are suitable for automatic generation.

Petri nets

German mathematician C. A. Petri in his dissertation thesis first used this concept in 1966 as formalism for description of mutual dependence between modeled system conditions and events. His concepts come out from decomposition of system to subsystems described by finite state machines. These machines work independently, but can be coordinated. From its beginning, Petri nets has come a long way, and nowadays are intensively studied. Petri nets are used mainly for analyzing, designing and modeling parallel and distributed systems as well as for parallel architecture description, compiler or computer net description or programming languages semantics description.

Petri net is a special biparital graph. It is composed from several types of elements:

- Places – are used for expressing modeled system states, circles usually represent places.
- Transitions – describe system changes, places are usually represented by rectangles.

- Arcs – are radically oriented and connects place with transition or transition with place. Arc cannot connect two places or two transitions. Every ordered pair (place, transition) or (transition, place) can be at most connected by one arc. Arcs are usually represented by rectangles.
- Inhibitory arcs – are special sort of arcs, this arc can connects given place with chosen transition. Inhibitory arcs are usually represented by line with circle on transition's side.
- Marking – represents actual system state using tokens. Token is depicted by small circle and can be used only in places. Every place can have given number of modeled system tokens. System states are modeled by mark flow. Initial system state is usually called M_0.

Formal definition for Petri net (sometimes called P/T Petri net as Place/Transition) is following:

Not marked Petri net is a senary: $PTN = (P, T, A, IA, AF, IF)$

where P is a finite non-empty set of places
T is a finite non-empty set of transitions
$P \cap T = \varnothing$ (their intersection is an empty set)
$A \subseteq (P \times T) \cup (T \times P)$ is a finite set of arcs
$IA \subseteq (P \times T)$ is finite set of inhibitory arcs
$AF: (A \cup IA) \rightarrow N$ is an arc function
$IF: P \rightarrow N_0 \cup \{\omega\}$ is initialization function.

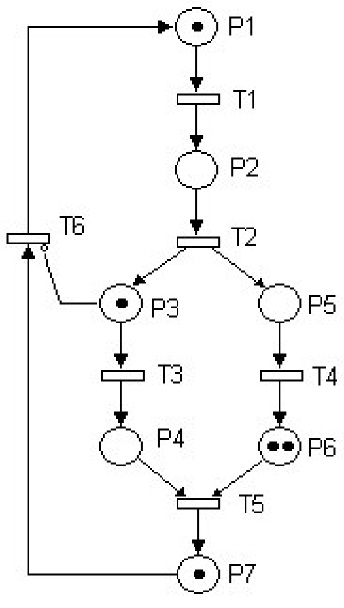

Fig. 2: Petri net

Figure 2 depicts example of Petri net with places P1, P2, …, P7 and transitions T1, T2, …, T6, there exists also inhibition arc (P3, T6). Initial marking of net using given ordering can be described by seven (1, 0, 1, 0, 0, 2, 1).

Petri nets are applied in many IT and automation fields. These are suitable for sequence flow problems modeling, especially with parallelism and synchronization. Processes often consist of parallel processes or are synchronized, so Petri net is a powerful tool for process management and workflow.

Finite state machine

As said in previous passage, Petri net concept comes out from decomposition of system to subsystems described by finite state machines. Now, we shortly outlined what FSM is. FSM is a model of computation consisting of:

- Set of states – define behavior and may produce actions.
- Start state – a state from the set of states, computation starts there (starting point).
- Input alphabet – set of valid characters or symbols, which can be combine to create an input string.
- Transition function – it maps input symbols and current states to a next state.

Computation begins in the start state with an input string. It changes to new states depending on the transition function. There are many variants, for instance, machines having actions (outputs) associated with transitions (Mealy machine) or states (Moore machines), multiple start states, transitions conditioned on no input symbol (a null) or more than one transition for a given symbol and state (Non-deterministic FSM). Formal definition defines FSM as a senary:

$$\mathrm{M} = (S, \Sigma, \Lambda, T, G, s) \text{ where}$$

S is a finite set of states,
Σ is a finite set called the input alphabet,
Λ is a finite set called the output alphabet,
T is a transition function ($T : S \times \Sigma \rightarrow S$),
G is an output function ($G : S \times \Sigma \rightarrow \Lambda$)
and start state $s \in S$.

How to implement (source codes or approaches) Petri nets or finite state machines is out of the scope of this paper.

Wizard design

As said above, Controller has to change model according to user changes and user interface and application data according to changes in a model. Model (process) is represented by directed acyclic graph with modified adjacency list. Each node of the graph corresponds to one wizard step. A node either refers to or directly stores domain data relevant to the step. Besides that, a node holds references to the adjacent nodes (previous nodes and possible following nodes). Wizard node class should contains functions for:

- adding outgoing edge of node (node can have more than one outgoing edges),
- returning the outgoing edge chosen for forward transition,
- storing and returning incoming edge used for reaching current node, it is used for backward transition,
- validation (if node validation) – should return valid edge.

All nodes refer to application domain data. This data is stored in a database, so Node class should contain functions also for storing, loading and validating data (e.g. right format, not empty values). Each wizard step need input data validation, if there is anything wrong taking into account domain model standing, the wizard does not traverse to another step. A node as well as an edge can implement this validation. Validation by node performs validation in node a traverse forward using valid edge. Validation by an edge is little bit different. We iterate over all outgoing edges and use the valid one. The last wizard step is confirmation. Confirmation node does not define any properties; it only applies wizard data to the application domain model, when the wizard finishes. The whole wizard transaction is then performed on a domain model.

An edge defines the path from one wizard step to another and holds the references to its source and target nodes. An edge class should implement the following important functions:

- get source node for an edge,
- get target node for an edge,
- validation (if edge validation) – returns true, if it is allowed move forward.

The wizard controller holds the references to the source and current nodes, and provides traversal functions. It can reference domain data like nodes do. The wizard controller class should implement following important functions:

- get source node and current node,
- get possible forward and backward traverses (for moving to next or previous node).

We outlined the model and controller main functions. These are edge validation rules, traversal commands, node edge list, and node and model synchronization. But for our purpose and for user-friendly usage, we need corresponding user interface (UI).

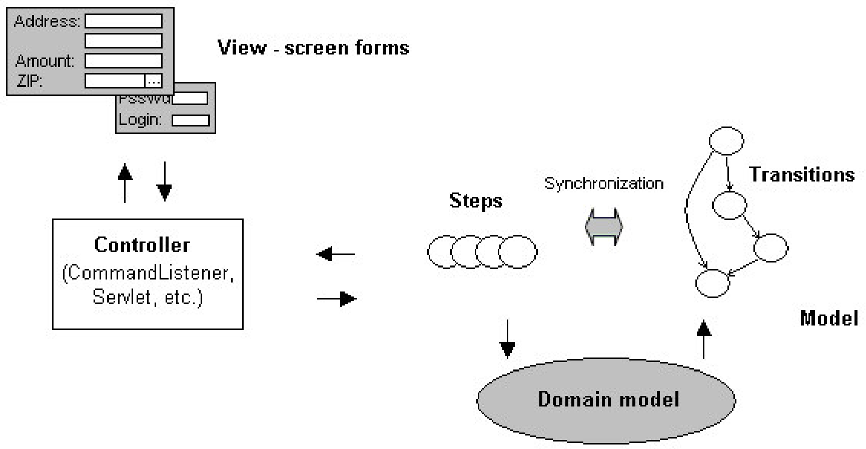

Fig. 3: MVC model for Wizard

User interface design

Now, we will discuss, how can we dynamically generate screen form according to the wizard step (machine state). First, we outline basic principles of user interface design, because we want to generate graphical forms, we discuss mainly graphical user interface – GUI. But some said principles and requirements are valid generally. These principles have to be included into process wizard (or its templates) to preserve current state of user interface in information systems. It is not possible to allow extreme changes in (generated) UI by implementing process wizard. Generated forms cannot be much different, they should be similar to previous versions, but they can demand less information on user, because of process automating (some information is valid throughout all process). For example, system automatically fills in today’s date and creator’s name, so these fields are not shown and not required on the form. User want to work with similar system, not to learn how to work with every new version because of UI and control changes. Basic (graphical) and general user interface design principles are:

- Use of metaphors – convey concepts and features using metaphors, it can help people to use computer similar to real life, e.g. a trash can for deleted (or deleting) files, folder to store documents.
- Transparency – it means mainly intuitive control and presumptive behavior of system.
- Consistency – consistency in the interface allows people to transfer their knowledge and skills from one application to any other. Use standards from operating system that your IS supports, e.g. users are used to close form using cross button in right-left corner, etc. Product should be consistent with earlier versions, with UI standards, with use of metaphors and also with people expectations.
- User control – user, not the computer, should initiate and control actions, but for situations where user can destroy data accidentally use warnings to notify them. This approach protects user, but he still remains in control.
- Feedback and dialog – keep users informed about what is happening with the system; provide some indicator when user performs an action. Provide understandable feedback, don't use messages like: "Error #11, stack trace …".
- Integrity – application has unsafe operation protection, undo functions or on-line help.
- Aesthetic integrity – information is well organized and consistent with principles of visual design, elements looks good and lucidly.
- Elasticity – flexibility of system to user behavior and his needs.
- Lucidity – for easy orientation of user.
- Productivity – system helps to grow productivity because it is based on simplicity, lucidity of screen forms and feedbacks.

This is not a full list of all user interface design concepts, principles and rules; these are only the basics and some of them overlays. This theme is extensive and is an object of human computer interaction (HCI) and usability engineering research. It is discussed in [Ap92], [An01] or [Kr03], and can be also found in [Pa03]. It is important to stress, that designing UI should respect international UI standards (mainly ISO norm 9241), cognitive psychologist's recommendation, platform UI rules and of course company's UI design rules.

Process wizard should implement (except general rules) mainly concrete rules for screen form design, some of them listed below are from MAC check list [Ap92], Microsoft Windows guide list is similar. Form graphics should resemble items that users are familiar with. The screen should look "clean" and free from clutter. Information in windows should be organized so that the most important information can be read first. Characters should be represented by 2 bytes (they are not necessarily 1 byte – East Europe languages, Chinese, Japanese). Text is not always left aligned and read from left to right (e.g. Arabic language). The last but not least is alignment of interface elements, use of fonts and colors. Fields should be logically divided using lines, grids or bookmarks; fields and labels should be adjusted. Use well-readable fonts and do not combine them on forms and use only one or two colors. Do not use flashy colors, they are absolutely inappropriate.

Implementing wizard to QI information system

There exist two approaches how to implement process wizard to QI information system. The first approach is a complete generation of a new form (we need to work with) using templates. Screen form is automatically generated always, whenever called. The second one uses complete existing form with marks. When user calls form, only specified parts are shown, using marks. First we need to examine the object model of QI, if there exist appropriate objects for process wizard. Basic data set dealing with processes and workflow in QI information system is Workflow DS (see fig. 4). The core of this DS is object Process, this has relation to Programming function object. Process wizard is a software component, which reads the Petri net state and calls/generates programming functions or its subclasses according to the state (understand, according to process description). Process wizard is a component without its own separate data objects, it reads existing ones. Because of this fact, we do not need alter data model or data sets. Now can we discuss two mentioned approaches.

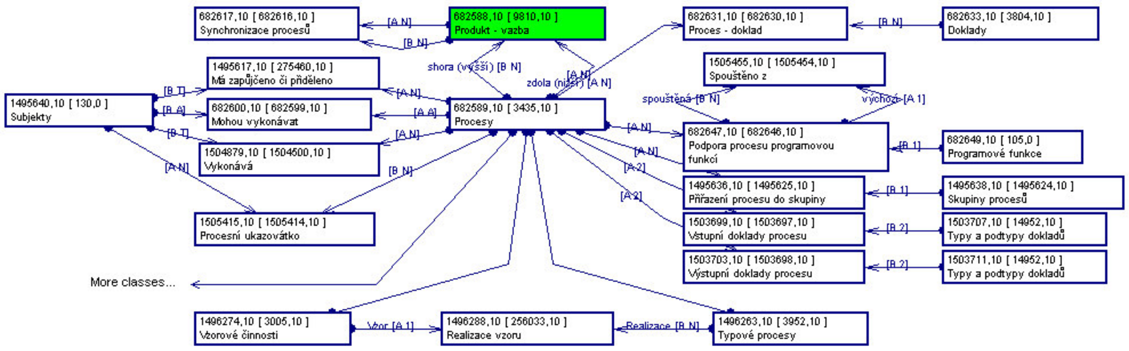

Fig. 4: Data set Workflow

The first approach – complete generation using templates – has some prerequisites. We need to group all possible data sets with abstract forms or with processes. This step is important to reduce the set of valid data sets. Why use all data sets, when the bill form uses only customer data set, bill data set and bill items' data set. If we don't do this, we can't prepare DS synchronizations. We use process approach, it means only necessary data is shown and used in every moment user performs the process. Other data is neither requisite (shown) nor accessible. That is one of process approach advantage. It results the necessity of only reduced count of data sets (only valid ones) at given process step. For form design and element positioning are used templates. Following example shows possible structure of a template.

```
<form name="Bill" title="Bill insertion">
   <container name="Customer" DS="Customer">
      <column title="Name" name="Name" field="Name">
      <column title="PaymentMethod" name="PaymentM" field="PaymentMethod">
      <column title="TotalAmount" name="TotalAmount" field="TotalAmount">
   </container>

   <container name="Items" DS="Item">
      ……
   </container>

   <synchronization>
      ……
   </synchronization>

   <control>
      <column="TotalAmount" controlType="NOT_NEGATIVE">
   </control>
</form>
```

Container is a logical part of a form, that can be visually divided by line, empty space or by bookmark. Generator (wizard) reads these templates and generates required form using data from valid data set. These templates can also contain sections, where the data set synchronizations can be defined as well as data controls. This approach needs to design all templates and implement into wizard generative mechanism.

Second approach deals with existing forms, so this doesn't solve our objectives, but it is also one possible approach. We need to design and implement complete maximal forms with marks. These marks are applied also on controls and data sets together with synchronization. If no field from DS is used, complete DS is used neither. Controls are joined with particular fields. When we don't include field, control is included neither. Using this approach we can simply solve data set synchronizations and data controls. In current application are controls and synchronizations designed by experienced engineers; it is not easy to automate this step. Finally, calling the form is a sort of customization. Before calling form, we only need to specify which marks use. This can be specified directly in process description or by user while operating. Final "generated" form has only data user needs. The problem is, that we need to design all forms supporting company's processes before using application. These state of application is presently almost the same, only difference is use of marks.

Wizard based on both approaches reads current state from Petri net or finite state machine (FSM), it means the process description. According to this description wizard calls programming functions, this means calling marked form or generating required form using template. Wizard methods mentioned above should be provided by formal mechanism (Petri net, FSM) and should be provided via interface.

Conclusion

This paper deals with workflow and its automation in information systems. We briefly described QI information system and a vision of a process wizard. We outlined our aims, process description using formal tools (Petri net, FSM), wizard functions and its design issues using MVC architecture. The last part deals with two possible solutions. The first one is based on generation using templates, the second one is based on existing marked forms customization.

Resources

[An01] Andrews, K.: *Human-computer interaction*. Lecture notes. Technical University Graz. 2001. http://iicm.edu/hci/.

[Ap92] Apple Computer.: *Macintosh Human Interface Guidelines*. Addison-Wesley. 1992. ISBN 0-201-62216-5.

[Bu92] Burbeck, S.: *Applications programming in Smalltalk-80: How to use Model-view-controller*. 1992. ParcPlace. http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.

[De02] Demel, J.: *Grafy a jejich aplikace*. Academia. 2002. ISBN 80-200-0990-6.

[DC1] DC Concept: *Solution engineer manual*. 2003. Internal document.

[He03] Herrington, J.: *Code generation in Action*. Manning. 2003. ISBN 1-930110-97-9.

[Kl03] Klimeš, C., Melzer, J.: *První elastický informační system: QI*. In. Sborník konference Tvorba software 2003. Tanger, s.r.o., Ostrava.

[Kl04] Klimeš, C., Procházka, J.: *Využití MDA pro integrovaný vývojový nástroj QI Builder*. In. Sborník přednášek konference Tvorba software 2004. Str. 106-111. Tanger, s.r.o. 2004. ISBN 80-85988-96-8.

[Kr03] Krug, S.: *Web design. Nenuťte uživatele přemýšlet*. Computer press. 2003. ISBN 80-7226-892-9.

[Ma] Martiník, I.: *Methodology of object-oriented programmatic system development using theory of object Petri nets*. Dissertation thesis. VŠB-TU. Ostrava. 1999.

[Pa03] Paleta, P.: *Co programátory ve škole neučí*. Grada. 2003. ISBN 80-251-0073-1.