SYSTEM OF SOFTWARE PRODUCTION MANAGEMENT

Galina Setlak, Sławomir Pieczonka

Abstract: *The paper presents the concept of managing the production and maintenance of software in a medium-sized and large business, whose main aims are: modeling, designing and analyzing production processes, and monitoring resources and performed works. The process of making changes to software was used as an example to present the* ***System of Software Production Management (SSPM)****, which due to its capabilities to define processes and resources of a business as well as forms with automatic generation of a database schema can be used both as a tool aiding management at all manufacturing stages, with the use of various methodologies for creating software, and an integrated environment for building and running applications.*

Keywords: *information system, application development, Petri nets*

ACM Classification Keywords: *J. Computer Applications, J.6 Computer-aided engineering, K.6 Management of Computing and Information Systems*

Introduction

Software manufacturing is a complex process that requires performing numerous tasks at various stages of system building. The aim of a modern *System of Software Production Management* (*SSPM*) for large and medium-sized businesses is creating a *RAD* (*Rapid Application Development*) environment which will allow to design and implement successive tasks in the production process and manage data and resources associated with them. *SSPM* will make it easier to model, automate, control and monitor resources and performed works through distributing tasks and information among participants in a project according to established rules and procedures. Easy access to information and sharing it in a system will improve the process of communication between participants in a project of software production.

Figure 1 shows the structure of the *System of Software Production Management* which was designed in such a way as to allow, in a short period of time, without engaging teams of programmers, to create software solutions oriented to the needs of production process, at the same time reducing resources needed to provide technical support required by traditional systems.

SSPM as a system designed for businesses creating software with the use of both traditional and light methodologies, such as extreme programming (XP), must meet their functionality requirements at each stage of creating and maintaining a system (conception, analysis, design, implementation, testing, deployment, technical support etc.), which can be described as follows:

- managing production processes,
- capability to define and modify a process (*Process Definition Module*), data and user interface (*Forms Definition Module*),
- synchronous performance of tasks and processes,
- support for rule-based manual and automatic tasks,
- support for sub-processes,
- algorithm for assigning a person to perform a task based on roles and abilities,
- list of user tasks, managing and displaying a list of tasks,
- definition of the structure of a business, resources (users, products, versions etc.), relations between them, roles (*Resources Definition Module*),
- support for handling emergency situations (escalation, notification),

- informing about critical data in the form of reports,
- tracing and displaying a set of defined reports, detailed and summary information about data describing production process, activity, progress in performing assigned tasks,
- system monitoring, registering events and changes in the system,
- administration of the system, user accounts.

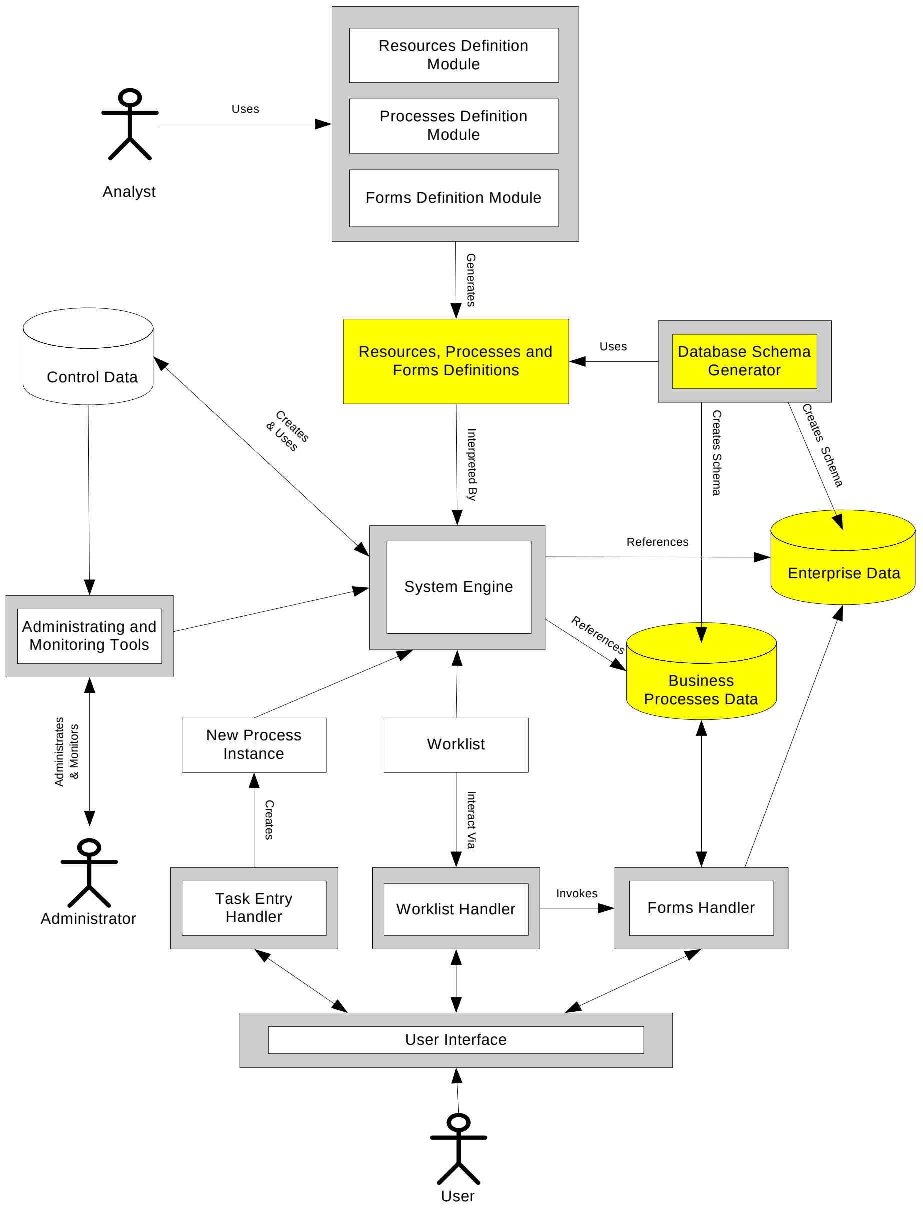


Figure 1. Context diagram of the System of Software Production Management (SSPM)

Making changes to software was used as an example presenting the application of the *SSPM*. Management of changes is a key element in manufacturing information systems and its aim is to increase the quality and effectiveness of activities. The tools (processes) created in the *SSPM* should ensure steady and efficient performance of tasks making use of the abilities of all participants in a project.

Figure 2 shows a simplified diagram of this process, from the moment of Requesting a change by a person related to the product, e.g. system user, tester, programmer, as a result of an error detection or request of adding new functionality to the system, up to the moment of completing tests confirming the correctness of the changes made.

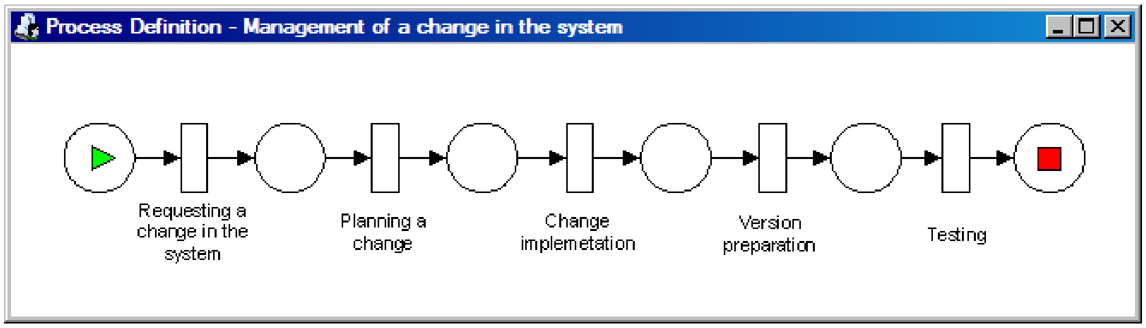

Process Definition - Management of a change in the system
Requesting a change in the system
Planning a change
Change implemetation
Version preparation
Testing

Figure 2. Simplified diagram of the process Management of a change in the system

Process Perspective

A Petri net was used as a language of process description in the *SSPM*. Classic Petri nets were proposed in the 60s by C. A. Petri [Petri, 1962]. Since that time, they have been used to, among others, model and analyse various types of processes, such as communication protocols, hardware, in-built systems, industrial systems, business systems.

The most important reasons for which Petri nets were chosen include:

– *graphical representation*. As a graphical language it is intuitive and easy to use, and can be understood by end users;

– *formal description*. The process of software production management has an accurate definition as Petri nets are formally defined;

– *extensibility and flexibility*. All constructions of modern management systems can be presented in the form of Petri net;

– *analysis*. Possibility of using various techniques in an analysis of the properties of a process that are provided by Petri nets allows to check, among others, waiting and response time, whether network is secure, whether there are any deadlocks.

Classic Petri nets were extended, forming new classes of coloured nets, and over time – hierarchical, logical ones. These extensions were of great importance, especially for very complex systems in which an important role was played by such factors as data and time. Examples of application of such extensions can be found among others in [Van der Aalst, 1994], [Van der Aalst, 1995], [Van der Aalst, 1996]. For the purposes of the *SSPM*, the *WorkFlow* (*WF-net*) class of net was used. The description of manufacturing process using this class of Petri net appears to be quite simple: *tasks* are presented as *transitions*, *states* and *conditions* as *places,* and *cases* are described by *markers*.

A formal definition of *WorkFlow* nets can be found among others in [Van der Aalst, 1998]. This chapter presents only two most important requirements that this net must meet.

– *WorkFlow* net has one input place *i* (in diagrams it is represented as a circle with triangle inside) and one output place *o* (represented as a circle with square inside). A *marker* put at input place corresponds to a *case* that is to be handled. A *marker* put at output place means a *case* that has just been handled.

– In *WF-net* there are no so called dangling *places* or *transitions*. Each *task* and each *condition* must be taken into account during processing a *case*. In other words, each *transition* and each *place* should be on one of the paths running from input place to output place.

Definition of the process of *Management of a change to the system*, used as an example here, created in *Process Definition Module* is shown in figure 3.

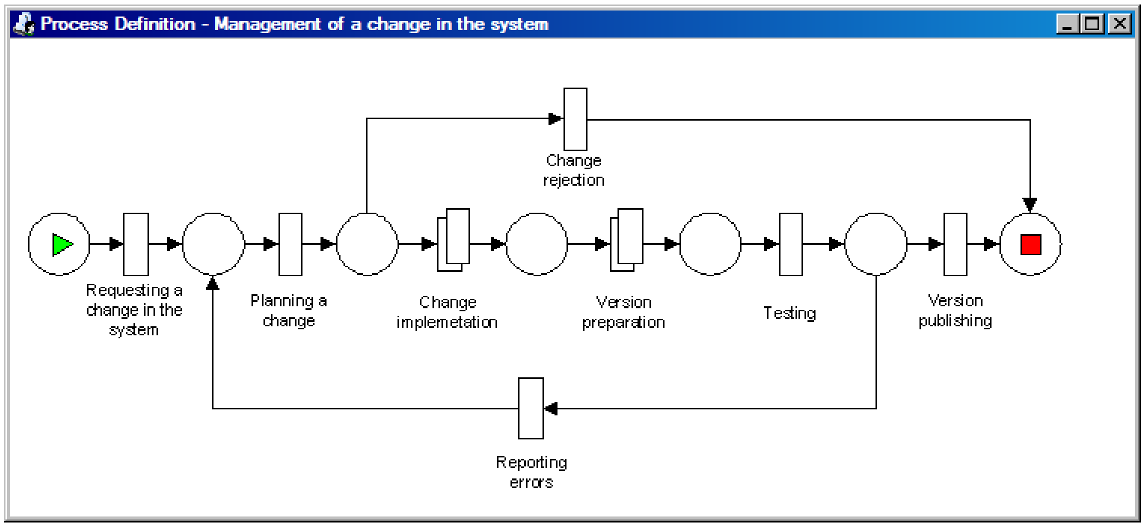


Figure 3. Diagram of the process Management of a change in the system

The process presented consists of both manual tasks, such as *Requesting a change in the system*, *Planning a change*, *Testing*, *Reporting errors*, *Change rejection*, *Version publishing*, and sub-processes (complex tasks), such as *Change implementation*, *Version preparation*. It is also possible to define automatic tasks.

Management of a change in the system (*marker* at input place) begins with execution of the task *Requesting a change in the system*, made by any user of the system, and then the task *Planning a change*, which consists in, for example, specifying in which version of the product a change should be made or deciding if a change should be made in the first place (change approval). If the change is not approved, the task *Change rejection* is performed, after which *marker* will be at output place and the process will be completed. If the change is approved, the sub-processes *Change implementation* and *Version preparation* will be then performed, followed by *Testing*. If testing is successfully completed, *Version publishing* follows and the process is completed, otherwise the task *Reporting errors* is performed, which leads to repeating the tasks performed starting with the task *Planning a change*.

Figure 4 shows the diagram of the sub-process *Change implementation*. When a change is to be introduced in the system, *marker* is at input place. Performance of the task *Implementation of changes* means execution and approval of changes. Then the task *Development testing* is performed. If it is completed successfully, the task *Code reviewing* will be then performed. If testing or code reviewing is not completed successfully, the task *Reporting errors* is performed, and it is necessary to make additional changes (*Change correction*). The task *Change publishing* is the last task of the sub-process (*marker* is at output place) and can be performed only if testing and code reviewing is successful.

Figure 5 shows the diagram of the sub-process *Version preparation*, which precedes testing. When a new version of the product is to be installed, *marker* is at input place *Version preparation*. It is then possible to perform the task *Preparation of installation parameters*, in which properties of an installation environment will be defined, such as for example, on which computer the product is to be installed, versions of operating system and other programs working with the product installed, type of the installation (from scratch, upgrading etc.)

The task *Starting the installation* can be performed after the task *Preparing installation parameters* is completed or directly, after staring performing the sub-process *Version preparation* (installation parameters are not necessary to define the way of installation or they are known at the moment of starting the sub-process, e.g. we use the setting of previous installation). Performance of the task *Starting installation* sets the status of the process to *Installation is under way*. The last task *Finishing the installation* completes the sub-process of installation (*marker* will be at output place *Version installed*).

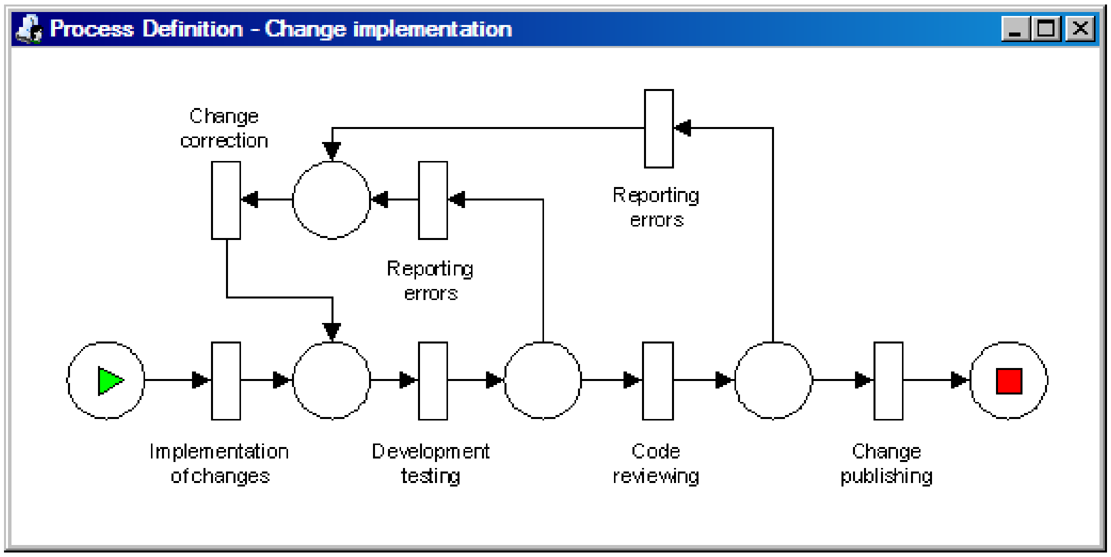


Figure 4. Diagram of the sub-process Change implementation

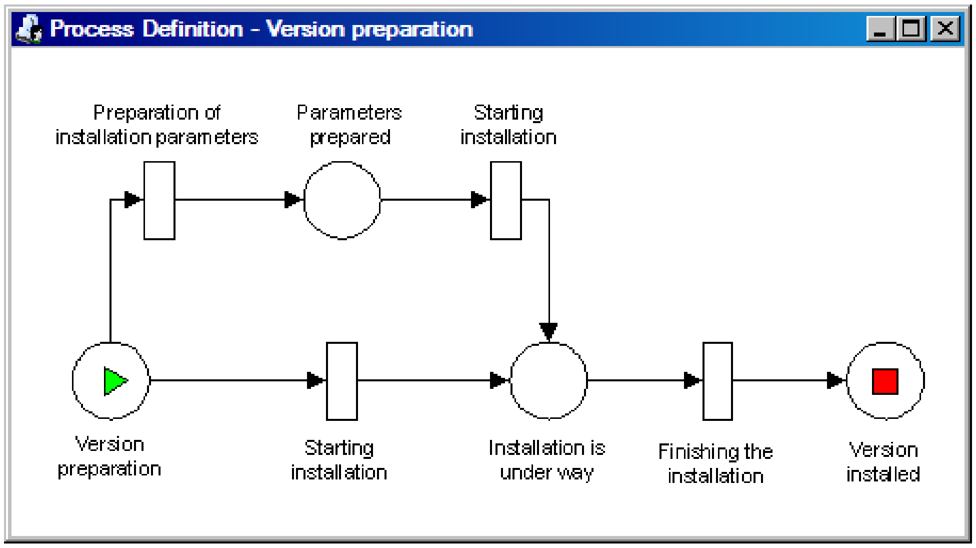

Process Definition - Version preparation
Preparation of installation parameters
Parameters prepared
Starting installation
Version preparation
Starting installation
Installation is under way
Finishing the installation
Version installed

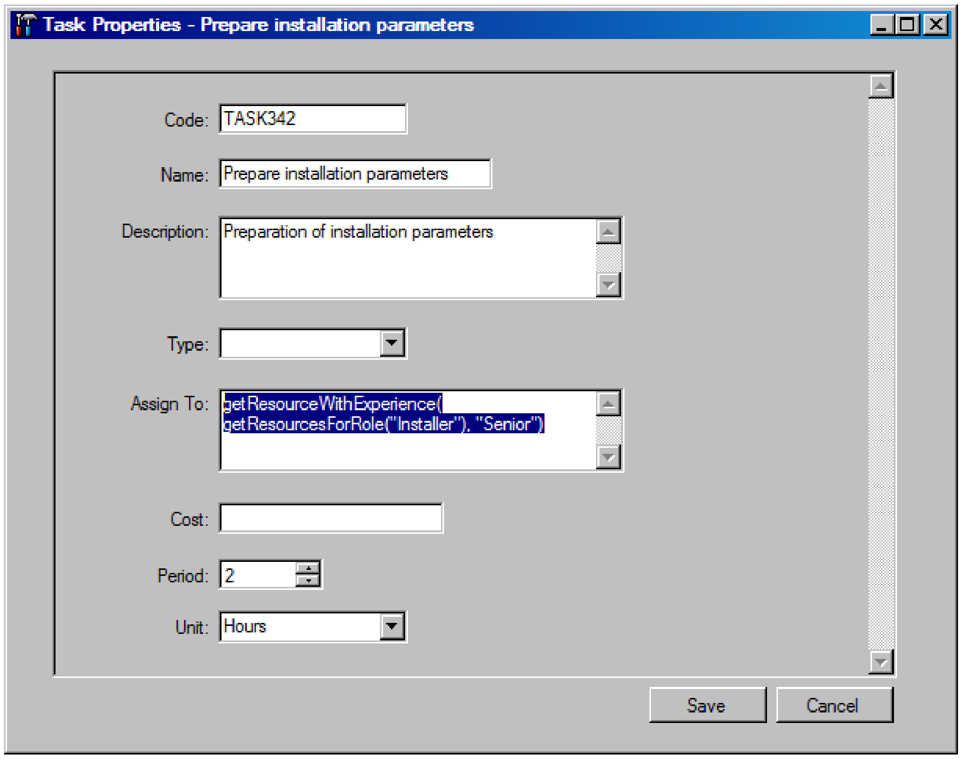
Figure 5. Diagram of the sub-process Version preparation

Resource Perspective

In many systems of software production management, process structure modeling is separated from an organizational model of a business and resources that are used. This division results from, among others, high complexity of a system and possibility of modifying a process without necessity to change the organizational model of a business. In this case, definition of a process consists in specifying what tasks are to be performed and in which order, without specifying the resources of a business that are used in this process. For assigning resources, the *SSPM* uses an algorithm based on definitions of the structure of a business, roles and skills and relation between them created in *Resource Definition Module* and directly connected with the definition of a process. Each task has a person assigned to perform it.

Figure 6, window *Task properties* (field *Assign to:*) shows how the task *Preparation of installation parameters* is assigned a person to perform it. In the example shown, the function **getResourceWithExperience()** assigns a person to perform the task based on two arguments, where:

- **getResourcesForRole("Installer")** is a set of all possible resources for the role *Installer*.
- **"Senior"** is a skill required from the resource.



Code:	TASK342
Name:	Prepare installation parameters
Description:	Preparation of installation parameters
Type:	
Assign To:	getResourceWithExperience(getResourcesForRole("Installer"), "Senior")
Cost:	
Period:	2
Unit:	Hours

Figure 6. Window of properties of the task Prepare installation parameters

Data Perspective

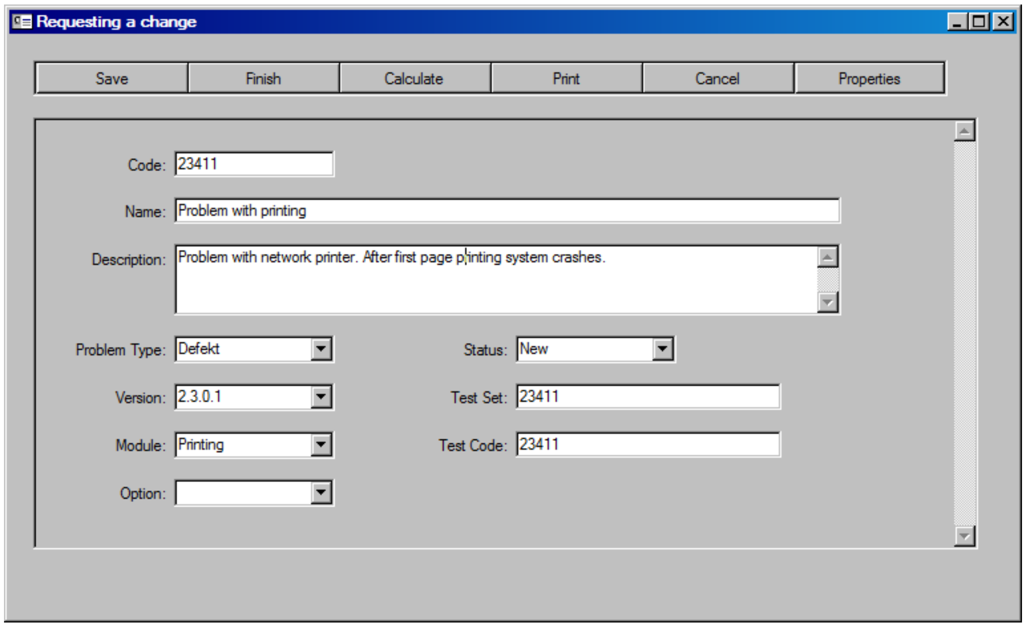
Each manual task must have a form defined (*Form Definition Module*). During defining a form we can use both simple elements, such as text fields, edition fields, buttons, drop-down lists, panels, selection options etc., which are usually associated with single fields in database tables, and complex elements, such as tables allowing a simultaneous access to many records in a database. Definition of a database can be generated (modified) based on the elements in a form.

Fig. 7 shows a launched form for the task *Requesting a change in the system.* Each launched form (task) has additional in-built button bar (*Save, Finish, Calculate, Print, Cancel, Properties*) which allows to perform on a task such actions as saving partial changes, approving and finishing performance of a task, or cancelling a task.

Conclusion

The chapter describes the *System of Software Production Management* in a large and medium-sized business, using the process of managing changes in software as an example. Due to its capabilities of creating processes, defining forms with data, and automatic generation of database schema, the *SSPM* ranks among *RAD* (*Rapid*

Application Development) systems and can be used not only as a tool for software production management but also as an integrated environment for creating and running any types of applications.



Code: 23411
Name: Problem with printing
Description: Problem with network printer. After first page printing system crashes.
Problem Type: Defekt
Status: New
Version: 2.3.0.1
Test Set: 23411
Module: Printing
Test Code: 23411
Option:

Figure 7. Window of a launched form <u>*Requesting a change*</u> *defined for the task* <u>*Requesting a change in the system*</u>

Bibliography

[Petri, 1962] Petri C.A.: Kommunikation mit Automaten. PhD thesis. Institut fur instrumentelle Mathematik, Bonn, 1962.

[Van der Aalst, 1998] Van der Aalst. W. M. P.:The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers, 8(1):21–66, 1998.

[Van der Aalst, 1994] Van der Aalst W.M.P.: Putting Petri nets to work in industry. Computers in Industry, 25(1):45–54, 1994.

[Van der Aalst, 1995] Van der Aalst W.M.P.: A class of Petri net for modeling and analyzing business processes. Computing Science Reports 95/26, Eindhoven University of Technology, Eindhoven, 1995.

[Van der Aalst, 1996] Van der Aalst W.M.P.: Three Good reasons for Using a Petri-net-based Workflow Management System. In S. Navathe and T. Wakayama, editors, Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96), pages 179–201, Camebridge, Massachusetts, Nov 1996.

Authors' Information

Galina Setlak *- D.Sc, Ph.D., Eng., Associate Professor, Rzeszow University of Technology, Department of Computer Science, W. Pola 2 Rzeszow 35-959, Poland, and The State Professional High School, Czarnieckiego 16, 37-500 Jarosław, Poland, e-mail: gsetlak@prz.edu.pl*

Major Fields of Scientific Research: decision-making in intelligent manufacturing systems, artificial Intelligence, neural networks, fuzzy logic, evolutionary computing, soft computing.

Sławomir Pieczonka *- PhD Student, M.A. The State Professional High School, Czarnieckiego 16, 37-500 Jarosław, Poland: e-mail: slawomir.pieczonka@pwszjar.edu.pl*

Major Fields of Scientific Research: