Annales UMCS Informatica AI VIII, 2 (2008) 75–87
DOI: 10.2478/v10065-008-0026-0

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI
http://www.annales.umcs.lublin.pl/

Test scenarios generation for a class of processes defined in the BPEL language

Krzysztof Sapiecha, Damian Grela*

Department of Computer Science, Cracow University of Technology, Warszawska 24, 31–155 Kraków, Poland

Abstract

The main purpose of this research is adaptation of critical paths method [**1**] to the processes defined in BPEL. The critical path method is the specification based and simulation oriented method. In the paper it is show that under some assumptions the BPEL process may be considered as an embedded system, in which tasks are like services and communication between tasks is like coordination of the services according to the task graph of the system. An example is given where a set of test scenarios is presented.

1. Introduction

BPEL4WS (Business Process Execution Language for Web Services) makes it possible to implement business process as an orchestration or a choreography of services distributed over the Web [**2**]. The main idea of the orchestration is a central coordinator, which invokes individual services of business process. In the choreography services invoke each other [**3**].

A definition of the process in BPEL is a composition of simple activities that combine and manage Web Services. An implementation of the definition can be done almost automatically, due to generally available CASE tools [**3**]. On

*Corresponding author: *e-mail address:* dgrela@pk.edu.pl

the other hand a validation of such designed and implemented process is still under discussion.

More or less effective methods for validation of computer systems have already been developed [**4, 5, 6, 7, 8**]. They fall into two categories: specification based and implementation based. In the first case a system is validated against specification requirements [**8**]. In the other one a set of test scenarios is generated to exercise implementation of the system or a model of an implementation of the system is created and some interesting features of the model are formally proved. Specification based validation makes it possible to detect design errors very early in the multi level procedure of implementation of the system. The most popular technique of specification based validation is simulation [**4**]. In the case of simulation test scenarios should be generated for validation of the system against temporal and functional requirements. An advantage of the simulation is that the generated validation tests may be used on different levels of designing of the system. However, nowadays systems are very complex. Therefore, the problem of generation of practical and useful test scenarios (providing correct validation result in acceptable time) is of the most significant.

In[**4**] a test scenario set validating a system against functional requirements is automatically generated on the basis of automaton modelling of these requirements. For the same purpose a genetic algorithm is used in [**6**]. In [**7**] there is used the modified ATPG algorithm to validate functionality of the systems described on RT–level. In [**1**] description of tools allowing automatic test generation for the systems in which functional requirements specification includes correct input values is given. The situation is a like, as far as validation of the system against temporal requirements is concerned. An approach to automatic test scenarios generation for the embedded system against temporal requirements (critical paths method) is presented in [**9**]. In [**10**] this method is extended to both temporal and functional requirements. In the sequel, in [**1**] an improved method for generation reduced set of test scenarios for validation of both temporal and functional requirements is presented.

The main purpose of this research is adaptation of critical paths method [**1**] to the processes defined in BPEL. The critical path method is the specification based and simulation oriented method. In the paper it is shown that under some assumptions the BPEL process may be considered as an embedded system, in which tasks are like services and communication between tasks is like coordination of the services according to the task graph of the system.

The problem is stated in section 2. Section 3 outlines adaptation of the critical path method for BPEL processes. An example is given where a set of test scenarios is generated. Section 4 presents conclusions.

2. Problem statement

For implementation of business processes recently the orchestration of Web Services has been used more often than their choreography. It relies on a central coordinator (Fig. 1a) which interacts with service receivers and service suppliers. It receives data, sometimes processes them, and distributes results. This is the way how all temporal and functional requirements for a process are met.

A process may use a data flow model (service provider waits for all necessary data and then executes the service and returns all results to a service caller) or may be scheduled (both starting and ending points of the service are fixed) according to an agreement established between the service caller and the service provider at the beginning of their cooperation. Usually a process as well as providers and recipients of services obey less or more critical time constraints. Service provider provides services for many service callers which in general do not like to wait. On the other hand, a service caller has its own temporal requirements that synchronize activities of the services. Hence, using scheduled services is advantageous for both service receiver and service provider. That is why such a model of cooperation between service provider and service caller is assumed in the paper. This, in turn, means that the validated BPEL processes meet the following requirements:
– the process is executed according to the schedule settled together by services providers and services callers,
– the process is functionally closed (cooperates with definite, invariable and finished number of services),
– the process has easily attainable initial state,
– for every service the time from invoking the service up to getting results of the service is steady.

The BPEL process which meets all described requirements is like an embedded system with closed functionality [**9**], in which tasks are like services and communication between tasks (data flow) is supervised by a coordinator acting according to a task graph of the system [**10**]. Additionally, tasks are distributed among various servers. On the basis of this analogy it might be possible to generate test scenarios for validation of the BPEL process against both functional and temporal requirements adapting a method developed for embedded systems. In the next section of the paper it will be shown that the critical path method introduced in [**1**] can be used for this purpose.

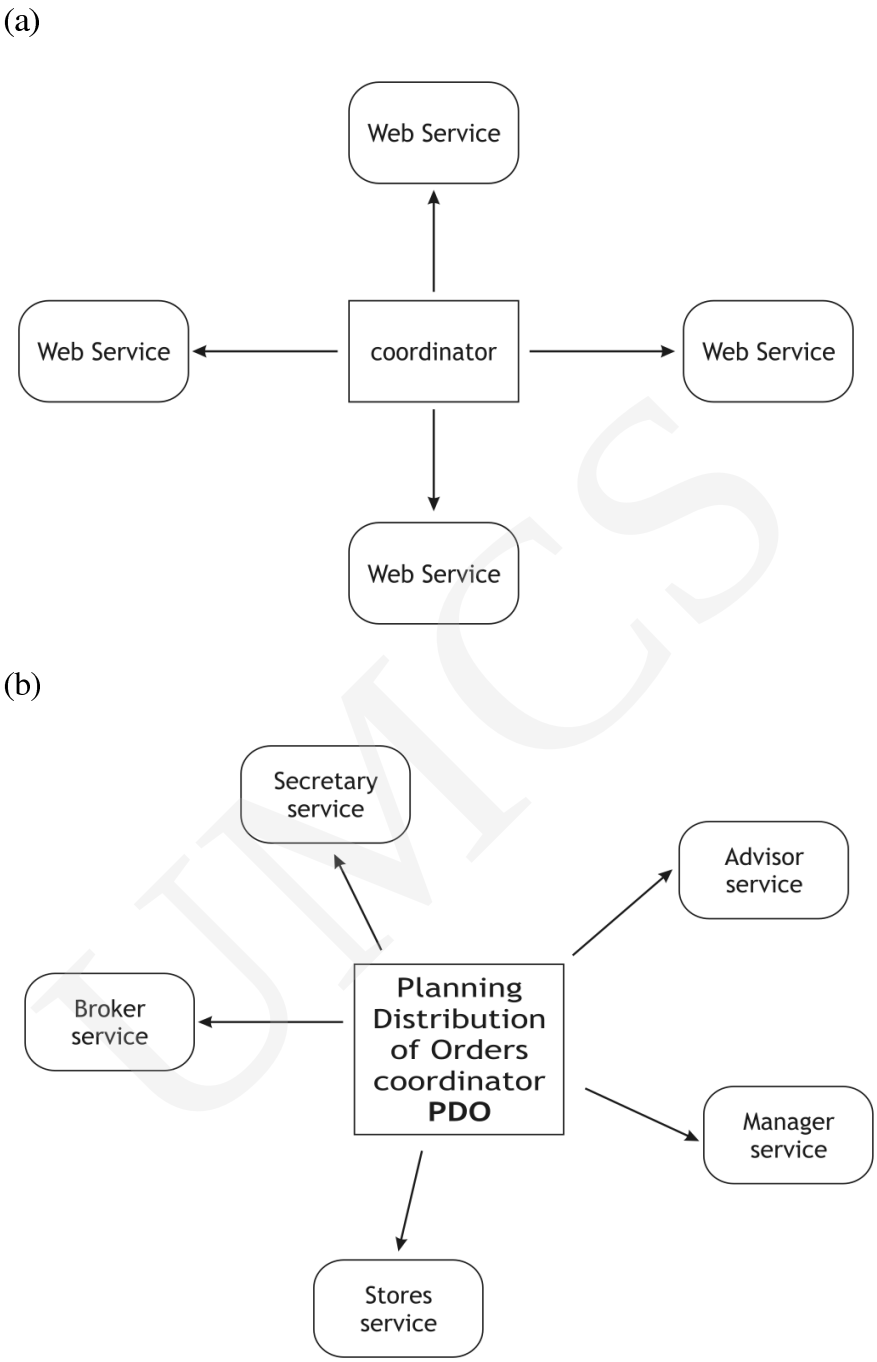

Fig. 1. Service orchestration (a) and its example – a process for planning distribution of orders (b)

3. Adaptation of the critical path method to BPEL processes

As in [**1**] the procedure of generation of a set of test scenarios for the BPEL process consists of the following steps:

1. formalization of specification requirements for the process with the help of SCR (Software Cost Reduction [**5**]) notation
2. creation of a model of the process; to this end so-called Functional Requirements Graph (FRG) is used,
3. derivation of Test Scenarios Tree (TST) from FRG, and finally
4. generation of a set of test scenarios from TST and FRG.

Specification requirements contain both functional and temporal requirements. A set of test scenarios generated with the help of the critical path method guarantees that each functional path (associated with functional requirement) and each critical path (associated with temporal constraint) are checked at least once. For the BPEL process functional requirements concern services and their coordination. A schedule of the process results in temporal constraints. The constraints define the minimum and maximum time $(t_{\min}, t_{\max})$ when a service has to be finished (actually it determines calls to providers).

A simple example of the process for Planning Distribution of Orders among stores (PDO) will serve as an illustration of the procedure. Theory and details of the procedure can be found in [**1**]. Table 1 contains functional requirements for the process. Temporal constraints will be given later on.

Table 1.

R_{Id}	Description
R_1	Broker sends orders (a). Secretary registers the plan (b). Advisor plans a distribution of the orders among stores (c). Manager sends his comments (d) or accepts the plan (e). Advisor revises the plan (c). Stores send comments to Advisor (f) or accept the plan (g). Advisor publishes the plan, Broker accepts the plan (h).
R_2	When the plan is published, it is sent to Broker (a). When Broker accepts the plan it is removed from the system (b).
R_3	When orders arrive from Broker they are sent to Secretary (a). When Secretary registers the plan its previous version is removed from the system (b).
R_4	When Secretary registers the plan or when Manager or Stores send comments to the plan it is sent to Advisor (a). When the plan is sent to Manager its previous version is removed from the system (b).
R_5	When the plan is finished by Advisor it is sent to Manager (a). When Manager accepts (b) or comments (c) the plan its previous version is removed from the system.
R_6	When the plan is accepted by Manager it is sent to Stores (a). When Stores accept (b) or comment (c) the plan its previous version is removed from the system.

The process runs on a system of servers and uses choreography of Web Services. These are as follows: Broker, Secretary, Advisor, Manager, Stores. Each of the services is accessible on a different server and the whole process of planning is coordinated through the central coordinator.

The SCR notation is used to formalise functional requirements for a process. It enables description of a process in category of events calling out changes in its states. Each activity of a process is described in the SCR notation as a sequence of actions (e.g. queries to providers which will be formulated precisely later during designing of the process) which begins with appearing definite events and ends with obtaining results.

Using the SCR notation for description of BPEL process we should introduce each of providers as a pair of ports, input and output. Hence, the process (the coordinator) is going to have five input ports and five output ports according to the services (Table 2).

Table 2.

Nr	Name	Type	Value
1	Broker_In	In	[None, Data]
2	Secretary_In	In	[None, Data]
3	Advisor_In	In	[None, Data]
4	Manager_In	In	[None, Yes, No]
5	Stores_In	In	[None, Yes, No]
6	Broker_Out	Out	[None, Data]
7	Secretary_Out	Out	[None, Data]
8	Advisor_Out	Out	[None, Data]
9	Manager_Out	Out	[None, Data]
10	Stores_Out	Out	[None, Data]

The plan can be in one of the following six states: Empty, Registered, Distributed, Revised, Commented and Accepted. A variable State corresponding to the current state of the plan is introduced. A state of the process is determined by a value of variable State and values of each of its output ports. Those variables are presented in Table 3. The process starts when State is Empty and on Broker_ In data appear (*this initial state is easily attainable*).

After transformation onto the SCR notation the functional requirements are given in Tables 4 and 5. The Tables show how each of the variables reacts for each of the events (Table 4 refers to variable State, Table 5 refers to the remaining 5 variables). Every row of the tables corresponds to some functional requirement (RId) and in the sequel to an execution one of the tasks (T_{Id}) for Coordinator. All tasks executed by Coordinator usually cooperate with Web Services (e.g. set values of output ports). It is important to emphasize that

Table 3.

Nr	Name	Value	Starting value	Type
1	State	[Empty, Registered, Distributed, Revised, Commented, Accepted]	Empty	Process state
2	Broker	[None, Data]	None	Output
3	Secretary	[None, Data]	None	Output
4	Advisor	[None, Data]	None	Output
5	Manager	[None, Data]	None	Output
6	Stores	[None, Data]	None	Output

not the Web Services (which by assumption are valid because they have been validated in sites of providers) but only Coordinator and its tasks are validated (*like task graph in the case of embedded systems*).

The first row of Table 4 describes the initial state of PDO process and corresponds to requirement R1a from Table 1 ("Broker sends orders"). To meet this requirement Coordinator (Fig. 1b) executes one of its tasks (TS_r). All tasks of Coordinator (TS_x) which changes a state of the plan are given in the last column of Table 4.

Table 4.

Old State	New State	Event	R_{Id}	T_{Id}
Empty	Registered	Broker_In=Data	R1a	TS_r
Registered	Distributed	Secretary_In=Data	R1b	TS_d
Distributed	Revised	Advisor_In=Data	R1c	TS_r
Revised	Distributed	Manager_In =No	R1d	TS_{d2}
Revised	Commented	Manager_In =Yes	R1e	TS_c
Commented	Distributed	Stores_In=No	R1f	TS_{d3}
Commented	Accepted	Stores_In=Yes	R1g	TS_a
Accepted	Empty	Secretary=None & Advisor=None & Manager=None & Stores=None	R1h	TS_e

The tasks implemented in the PDO process are as follows:
TS – change a state of the plan, defined in Table 4
TFB – forward the plan to Broker
TFS – forward the plan to Secretary
TFA – forward the plan to Advisor
TFM – forward the plan to Manager
TFT – forward the plan to Stores.

The last 5 tasks are defined in Table 5.

Table 5.

Variable	State	Value	Event	R_{Id}	T_{Id}
Broker	Accepted	Data	InMode	R2a	TFB_{on}
Broker	None	None	InMode	R2b	TFB_{off}
Secretary	Registered	Data	InMode	R3a	TFS_{on}
Secretary	Distributed	None	InMode	R3b	TFS_{off}
Advisor	Distributed	Data	InMode	R4a	TFA_{on}
Advisor	Revised	None	InMode	R4b	TFA_{off}
Manager	Revised	Data	InMode	R5a	TFM_{on}
Manager	Commented	None	InMode	R5b	TFM_{off}
Manager	Distributed	None	InMode	R5c	TFM_{off}
Stores	Commented	Data	InMode	R6a	TFT_{on}
Stores	Accepted	None	InMode	R6b	TFT_{off}
Stores	Distributed	None	InMode	R6c	TFT_{off}

For reactive embedded systems temporal constraints are put on a system and its environment [**11**]. Such constraints define the period of time, given as a pair $(t_{\min}, t_{\max})$, in which the system has to finish processing certain data. Processing of the data means an execution of certain tasks. Therefore, such kind of constraint concerns the time of execution of certain subset of tasks. Similarly, temporal constraint in the case of the BPEL process also defines for Coordinator the time of execution of certain subset of tasks and in the sequel the time of execution of certain services.

Table 6 contains temporal constraints for the PDO process written down with the SCR notation [**9**].

Table 6.

C_{Id}	Type	(t_{min}, t_{max})	T_{Id}	Conditions
C_1	P	(0, 1d)	-	{@T(Broker_In=Data)} → {@T(Secretary=Data)}
C_2	P	(0, 2d)	-	{@T(Secretary=Data)} → {@T(Advisor=Data)}
C_3	P	(0, 3d)	-	{@T(Advisor=Data)} → {@T(Manager=Data)}
C_4	P	(0, 4d)	-	{@T(Broker=Data)} → {@T(Broker=None)}

The BPEL process is validated against both functional and temporal requirements. The process is valid when all specification requirements are met. A set of test scenarios should check each of the requirements at least once [**1**]. If a requirement is not met then the process should behave incorrectly when a test scenario corresponding to the requirement is applied to the process. A test scenario is defined by the state of the process and states of its input ports.

Generation of test scenarios begins with developing a model of a process. To this end Functional Requirements Graph (FRG) [**4, 9**] is used. FRG models the process with the help of an automaton (idea very popular as far as generation of tests is concerned, e.g. [**4, 6**]). States of the automaton correspond to states

of the process (in the example a state is described by 6 variables). A transition between two states is labelled with a pair of events: initiating the transition and finishing the transition (initiating the query and receiving the response). The model is automatically generated from formal specification of requirements (from the SCR notation) and information about services.

The model of PDO process is shown in Fig. 2.

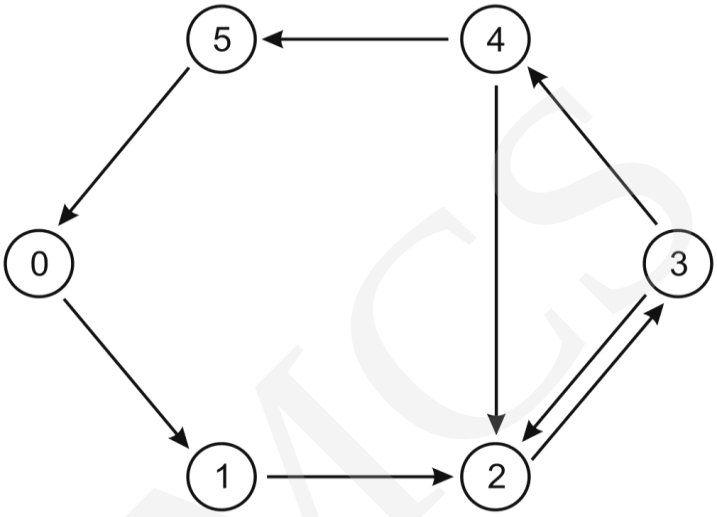

Fig. 2. FRG for the PDO process

For readability there are no values of variables describing states of the process (nodes of FRG) and labels describing transitions between states (edges of the graph). These are given in Tables 7 and 8. Moreover, in Table 8 for every transition there are shown identifiers of tested functional and temporal requirements.

Table 7.

Node	Value					
	State	Broker	Secretary	Advisor	Manager	Stores
0	Empty	None	None	None	None	None
1	Registered	None	Data	None	None	None
2	Distributed	None	None	Data	None	None
3	Revised	None	None	None	Data	None
4	Commented	None	None	None	None	Data
5	Accepted	Data	None	None	None	None

Table 8.

Tran-sition	Events		Identifiers		
	Initiating	Finishing	R_{Id}	C_{Id}	T_{Id}
0 -> 1	Broker_In=Data	Secretary=Data	R1a, R3a	C_1	TS_r, TFS_{on}
1 -> 2	Secretary_In=Data	Secretary=None & Advisor=Data	R1b, R3b, R4a	C_2	TS_d, TFS_{off}, TFA_{on}
2 -> 3	Advisor_In=Data	Advisor=None & Manager=Data	R1c, R4b, R5a	C_3	TS_r, TFA_{off}, TFM_{on}
3 -> 2	Manager_In=No	Manager=None & Advisor=Data	R1d, R5c, R4a		TS_{d2}, TFM_{off}, TFA_{on}
3 -> 4	Manager_In =Yes	Manager=None & Stores=Data	R1e, R5b, R6a		TS_c, TFM_{off}, TFT_{on}
4 -> 2	Stores_In=No	Stores=None & Advisor=Data	R1f, R6c, R4a		TS_{d3}, TFT_{off}, TFA_{on}
4 -> 5	Stores_In=Yes	Stores=None & Broker=Data	R1g, R6b, R2a		TS_a, TFT_{off}, TFB_{on}
5 -> 0	Secretary=None & Advisor=None & Manager=None & Stores=None	Broker=None	R1h, R2b	C_4	TS_e, TFB_{off}

From the first row of Table 8 it results that if the PDO process is in state 0 and on port Broker_ In Data appear then the process goes to state 1 and transfers Data onto variable Secretary. Moreover, going through this transition ($0- > 1$) the process checks requirements R1a and R3a, and tasks TS_r and TFS_{on}. Binding specification requirements with transitions in FRG is the main idea of the critical paths method [**1**].

Next, Test Scenarios Tree (TST) is derived from FRG. Starting with the initial state of the process consecutive critical paths (temporal requirements) and functional paths (functional requirements) are joined to TST. A path may be omitted when it is covered by TST. This results in reduction of the length of the set of test scenarios. Each branch of TST describes different behavior of the process. An algorithm for generation the TST is given in [**12**].

TST for the PDO process is shown in Fig. 3. State 0 is the initial state of the PDO process. Nodes labelled $C_{Id}v$ or C_{Id} ^ respectively begin or finish critical path associated with the temporal constraint C_{Id} (Table 6). It is seen that all critical paths (all temporal requirements) are included (are checked) into (along) TST. Sub–branch $3- > 4- > 2$ is added so that all functional paths are also included.

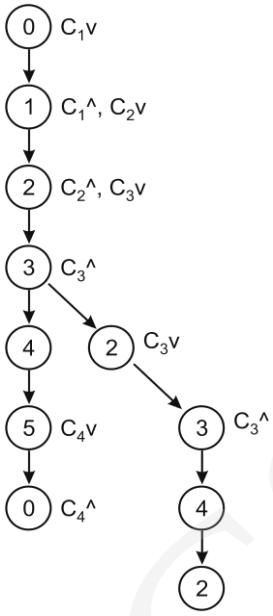

Fig. 3. TST for the PDO process

A single branch of TST determines one test scenario (TS). Each TS checks different functional requirements (TSF) along with their temporal constraints (TSC), if any.

Table 9 presents two test scenarios generated for the PDO process. The first column of Table 9 (TSF) shows the events (initiating/finishing) defining a test. The second column of Table 9 (TSC) shows symbolically written down moments of time (t_i/t_j) in which a finishing event should appear. Enumeration of execution times for services is also given.

Table 9.

TS1		
TSF1	TSC1	
	t_i / t_j	[Cid]: t_j-$t_i \in (t_{min}, t_{max})$
Broker_In=Data / Secretary=Data	t1 / t2	[C1]: t2-t1 ∈ (0,1d)
Secretary_In=Data / Secretary=None & Advisor=Data	t3 / t4	[C2]: t4-t3 ∈ (0,2d)
Advisor_In=Data / Advisor=None & Manager=Data	t5 / t6	[C3]: t6-t5 ∈ (0,3d)
Manager_In=Yes / Manager=None & Stores=Data	t7 / t8	
Stores_In=Yes / Broker=Data & Stores=None	t9 / t10	
/ Broker=None	t11 / t12	[C4]: t12-t11 ∈ (0,4d)

TS2		
TSF2	TSC2	
	t_i / t_j	[Cid]: t_j-$t_i \in (t_{min}, t_{max})$
Broker_In=Data / Secretary=Data	t1 / t2	[C1]: t2-t1 ∈ (0,1d)
Secretary_In=Data / Secretary=None & Advisor=Data	t3 / t4	[C2]: t4-t3 ∈ (0,2d)
Advisor_In=Data / Advisor=None & Manager=Data	t5 / t6	[C3]: t6-t5 ∈ (0,3d)
Manager_In=No / Advisor=Data & Manager=None	t7 / t8	
Advisor_In=Data / Advisor=None & Manager=Data	t5 / t6	[C3]: t6-t5 ∈ (0,3d)
Manager_In=Yes / Manager=None & Stores=Data	t7 / t8	
Stores_In=No / Advisor=Data & Stores=None	t9 / t10	

TS1 covers the branch $0- > 1- > 2- > 3- > 4- > 5- > 0$, and TS2 the branch $0- > 1- > 2- > 3- > 2- > 3- > 4- > 2$ in TST. A set of algorithms for generation of reduced sets of TSi is given in [**13**].

4. Conclusions

An approach to generation of a set of test scenarios from specification requirements which was presented in the paper is simple and easy for application in practice. Human task consists of writing down specification requirements for the BPEL process in the SCR notation, only. All further calculations are automated [**12**].

The adaptation of the critical path method which was introduced here is restricted to a subclass of BPEL processes. However, if the process uses a service accessible in several versions (uses this version which has all necessary data at present) or a service is accessible on several servers with various efficiency (uses this server which is not occupied at present) then each of such services can be replaced by a subset of functionally equivalent services that meet the restrictions of the method. This complicates the model of the process and lengthens calculations, but does not lever up correctness of the method. Human tasks in Web Services may require special treatment but this could be taken into consideration when a schedule of the process is established by service providers and service callers.

A set of test scenarios for the example considered in the paper contains only 15 stimuli including initialization. It is very concise but all the requirements (functional and temporal) are checked at least once.

References

[1] Strug J., Sapiecha K., *Test Scenarios Generaion for Reactive Embedded Systems*, Real-Time Systems, WKŁ, (2005) 241, in Polish.

[2] Zakrzewicz M., *Introducing to Web Services: SOAP, WSDL & UDDI technologies*, XIII Semunarium PLOUG (2006), in Polish.

[3] Zakrzewicz M., *Business applications implementation at WS-BPEL technology*, XIII Semunarium PLOUG (2006), in Polish.

[4] Cunning S., Rozenblit J. W., *Automating Test Case Generation for Requirements Specification for Realtime Embedded Systems*, Proc. of the 1999 IEEE SMC'99 (1999).

[5] Heitmeyer C., Kirby J., Labaw B., *The SCR Method for Formally Specifying, Verifying and Validating Requirements: Tool Support*, Proc. of the International Conference on Software Engineering (1997).

[6] Lajolo M., Lavagno L., Rebaudengo M., *Automatic Test Bench Generation for Simulation-based Validation*, Proc. of the 8th CODES (2000).

[7] Zhang L., Hsiao M., *Automatic Design Validation Framework for HDL Descriptions via RTL ATPG*, Test Symposium, 2003. ATS 2003. 12th Asian (2003).

[8] Dalal S., Jain A., Patton G., Rathi M., Seymour P., *AETGSM Web: A Web Based Service for AutomaticEfficient Test Generation from Functional Requirements*, Proc. Of the 2nd IEEE Workshop on Industrial Strenght Formal Specification Techniques (1998).

[9] Strug J., Deniziak S., Sapiecha K., *Validation of Reactive Embedded Systems against Temporal Requirements*, Proc. of the 18th IEEE ECBS, Brno (2004) 152.

[10] Strug J., Deniziak S., Sapiecha K., *Validation of Reactive Embedded Systems against Specification Requirements*, Annales Informatica (2004).

[11] Dasdan A., Ramanathan D., Gupta R. K., *Rate Derivation and Its Applications to Reactive, Real-time Embedded Systems*, Proc. of the 35th Design Automation Conf. (1998).

[12] Sapiecha K., Strug J., Maksym P., *The Generator of Test Scenarios for validation of reactive embedded systems*, Technical Journal of Cracow University of Technology - Computer Science b.1-I, in Polish.

[13] Strug J., *Automatic test scenarios generation for valitation of reactive embedded system, Doctor thesis*, Warsaw University of Technology (2007), in Polish.