Asynchronous Runtime Verification of Business Processes: Proof of Concept

Ivo Oditis
DIVI Grupa Ltd
Riga, Latvia
ivo.oditis@di.lv

Janis Bicevskis
University of Latvia
Riga, Latvia
Janis.bicevskis@lu.lv

***Abstract*— This paper describes a new approach to the runtime verification of business processes as well as discusses its approbation. This approach utilizes the idea of a multi-agent system, presenting a runtime verification system that is able to perform verification without affecting business processes that are running in a heterogeneous environment. The verification mechanism monitors business process execution and verifies the compliance of the execution with the description of the verification process. This research also led to the development of a domain-specific language to describe business process verification. The prototype of verification mechanism is developed and tested on real business processes, and verification performance, produced overhead and limitations are evaluated.**

Keywords- business process verification, runtime verification, multi-agent systems

I. INTRODUCTION

There have been burning issues related to the development of information systems (IS) ever since the very beginning - whether the IS operates correctly, whether the system's results are adequate, and whether the IS is in the correct state in terms of the relevant business [1]. An exceptional situation can be caused by a number of circumstances such as poor quality, insufficient testing and incorrect use of system: time limits, or the order in which various operations must be performed, and others.

The origins of discussions about these problems can be found in [2], [3], [4] with subsequent discussions found in [5], [6], [7]. Most of these researchers have looked at the best way of embedding supportive components into software so as to control and verify the execution of software and the communications process during runtime.

The authors of this paper argue that it is often impossible to add any component to existing software, e.g., software source code is not available or there is not enough know-how about software implementation. Even if software involved in business processes could be partly supplemented with instrumentation, it is still doubtful that all the necessary software could be prepared for required business process verification.

The solution offered by the authors implies the construction of an external process verification mechanism, which relates to the scientific novelty of this paper. Unlike the typical runtime verification (not monitoring) mechanism, it is verification by observing processes from aside without intervention into execution of processes. According to the business process description, events confirming process step execution are collected and verified. All events are detected by event agents and sent to centralized controller for verification. Agents are developed for different components (databases, file systems, email servers etc.) and not implemented into software under verification. Thereby proposed mechanism allows verifying business processes executed by more than one system, running over several platforms and even provided by more than one operator (e.g., data published in web).

The first position paper containing the idea of solution was published by authors in 2010 [8] with the following publication of detailed mechanism in May, 2014 [9] and June, 2015 [10]. This paper provides short introduction to the solution and aims to describe the prototyping of the solution and draws conclusions on its limitations and usability.

The second chapter of the paper deals with the problem, the third analyses related verification solutions. The fourth describes the proposed solution to the problem – as the architecture of the business process verification mechanism and the domain-specific language that describes the process verification – and the fifth chapter introduces a prototype used to test this concept, also offering related observations.

II. BUSINESS PROCESS VERIFICATION

A. *The problem*

The authors of this paper have investigated the use of business processes at large companies where many different types of software are used [1]. The establishment of a heterogeneous environment in large and long–lasting companies is inevitable, because the organizational size and functions are subject to changes over the course of time and develop gradually.

Serious problems are caused by a distributed environment in which many systems are running simultaneously on different platforms and are exchanging data among themselves. As a general rule, service staff must keep track of the correctness of the process, particularly if

the system has no built–in process controls. Many business processes, however, involve two or more systems, and these are typically controlled in a manual way. This means that the smooth usage of the systems mostly depends on the technical skills of the staff and the precise execution of operations by employees.

Automated runtime verification is proposed in order to address the aforementioned issues, also reducing the dependency of system usage on the subjective factor of employees. The business process runtime verification might identify the following business process execution problems:

- problems arising from changes in execution environment (in real life such problems are rarely identified by testing);
- process execution delays (this is particularly important for processes that require a quick response or for long working processes);
- data exchange between two or more systems implementing one business process;
- and others.

B. The essence of the solution

When it comes to aspiring business process verification, this paper proposes the development of a separate verification process for each controllable business process (base process). Verification processes are described in domain–specific language that has been developed by the authors, and it is elaborated in further sections of this paper. Base processes typically involve IS software, while verification processes should be executed on the basis of independent and external controlling software (a controller). The steps of the verification process are linked to base process steps and are described by events that acknowledge the execution of the base process step. Base processes and verification processes are executed independently.

Execution of a base process modifies IS memory, for instance, a file system or database. The controller receives acknowledgement from event agents about these modifications of memory, also identifying inconsistencies between the information that is received and the description of the verification process. If inconsistencies are detected, then they are reported to system support staff. The proposed approach, therefore, can verify a base process only if IS memory changes that are caused by process execution can be detected by event agents.

III. RELATED RESERCHES ANS SOLUTIONS

Several authors have argued that static verification and testing of software are insufficient aids for modern business process verification that relates to a complex and a heterogeneous environment [11]. Processes are implemented by many components which change independently over the course of time. This means that the process runtime verification must be conducted through the entire lifetime of the process [12], [13].

A. A verification process that is integrated into software which executes the base process

Verification of a business process execution can be implemented in an IS if the execution verification is specified in the relevant software requirements. If there are time limits related to document processing, for instance, then time limit controls can be applied to the IS, and the IS must warn users when the time limit is expiring.

Many researchers (e.g., [5], [6]) have argued that runtime verification can be effectively executed if it is installed into IS software. In practice, however, very few information systems have functionality in relation to runtime verification. Most rely on event logging in case of a failure.

One of the related authors has proposed the use of installed self–testing components for runtime testing [14]. This ensures the construction of a software testing code inside the software itself, thus making it possible to test software not just before deployment, as is suggested in classical tests, but also in a production environment that involves runtime data.

B. Business process runtime verification with the use of separate components

Various authors have suggested runtime verification solutions for SOA–based Web services. Verification description language WSCoL is developed [13]: it makes it possible to verify Web service calls by adding pre and post–conditions. Monitoring involves a specific proxy server – it verifies all service requests and responses. WSCoL was updated with service execution timing, and the result was an extension that is known as "timed WSCoL" [15].

Other authors have suggested solutions similar to WSCoL. One example is the LTL–FO+ [16] language (Linear Temporal Logic with Full First–Order Quantification). This makes it possible to describe the interdependence of SOA message services, also providing their verification in runtime.

All of these solutions make it possible to develop verification descriptions independently from the tested software. As a rule, moreover, these solutions are focused on a heterogeneous environment, because SOA serves as "glue" in such environments. The drawback of these solutions is orientation only on SOA, therefore these are not applicable for legacy systems.

C. Business process runtime verification via combined solutions

One of the so–called combined solutions is based on trace analyses. The software must have a built–in trace writer or logging mechanism.

The trace or log that is created by this process is analyzed via a separate monitor process. Two types of trace analysis algorithms can be used. The first algorithm analyses the trace after the execution of the process has been finished [17], i.e., post–factum. The second analyses the trace in parallel with the traced process [18].

Another combined solution is based on the generation of a monitoring application [19]. This idea relies on the automatic generation of a monitoring application and

instrumentation from the system's specification. This means that the solution requires very detailed software specification, as well as the availability of a source code.

IV. Business Process Runtime Verification

Automated workstations, servers and network monitoring that are supported by a number of software and controlling systems have become a part of our daily working routines. Business process runtime verification, however, is not so commonplace and presents more sophisticated issues:

- Business process models tend to be ambiguous, thus enabling incorrect interpretations and often missing information about time control;
- Software implementing business processes is not properly tested for non–standard situations, e.g., data exchanges with other systems are typically treated as being reliable and without errors;
- IS–based business processes often have no common interface, and data exchange is conducted manually by users.

Efforts to provide the verification of business process have been focused on three main objectives:

- Verification of processes involving more than one system, i.e., heterogeneous systems;
- Verification of a wide range of system platforms (not just Web services);
- An early warning system for staff support.

This paper proposes the construction of an autonomous process verification mechanism to reach these objectives. The proposed automated process verification mechanism checks whether a described process is running in accordance with the verification process descriptions, i.e., verifying the sequence of activities and the timing of operations. If a discrepancy is detected between the description and the ongoing process, then the control mechanism sends a report to system support staff. The proposed verification mechanism operates asynchronously to the base process. If unconformity is detected, the base process is not stopped, because the verification process runs in parallel to it.

A. The architecture of runtime verification – the controller and the agents

The mechanism that has been proposed by the authors verifies the compliance of business process execution with the process verification description. The description of the verification process must specify two aspects – event–confirming step completion and the time when each step in the process must be finished. Therefore, the proposed architecture of the process verification mechanism contains two major services – an event detector (agent) and a controller. The agent is software that can check the occurrence of a specific event such as "file deleted". The controller is software that can analyze process verification descriptions, require that agents detect events specified in the verification description, collect event acknowledgements, and verify flow compliance with the verification description.

Fig. 1 presents an example. Two systems (A and B) provide one business process. The process starts by receiving a data file from the client via FTP in system A. The file is processed, and data are saved in the system's A database. Then system A exports the received data as a structured text file for system B. System B reads the file and saves the data in its database. The process verification must check four events: a new file created on an FTP server, new data records are added to the system's A database, the export file is created for system B, and new data records are saved in system B's database.

Here process verification involves two types of agents – a file system event agent and a database event agent. The controller must take four steps in relation to the process verification:

- First the controller requires the file system agent to check the creation of new files on the FTP server. As soon as notification about a new file is received from the agent, the controller must create a new instance of the verification process.
- According to the process verification description, the next base process step prescribes the addition of data to system A's database. Therefore, the controller requires the database event agent to check corresponding updates to the database.
- Upon receipt of a database update notification from the database event agent, the controller requests and awaits notification from the file system agent about the generation of an export file.
- As soon as the generation of the export file is verified, the controller sends an inquiry to the database event agent about a notification about the updating of system B's database.

Obviously this verification process can operate in parallel with systems without affecting any of their operations and without any modification of the systems. Thus the mechanism is applicable to the verification of any process that is provided by an IS, assuming that:

- There is enough information about events that indicate the execution of base process steps;
- Agents are available in environments in which the system that provides process execution is running.

B. The idea of a domain–specific process verification language

Several languages can be used to describe business processes, including UML activity diagrams, Event–driven Process Chains, Business Process Model Notation or the Business Process Execution Language. These languages make it possible to describe base processes in detail, but they fail to provide the following process verification data:

- How to determine that a step has been finished;
- When the process step must be completed;
- Where the process steps are executed.

Thus process verification description should be implemented as extension of these languages.

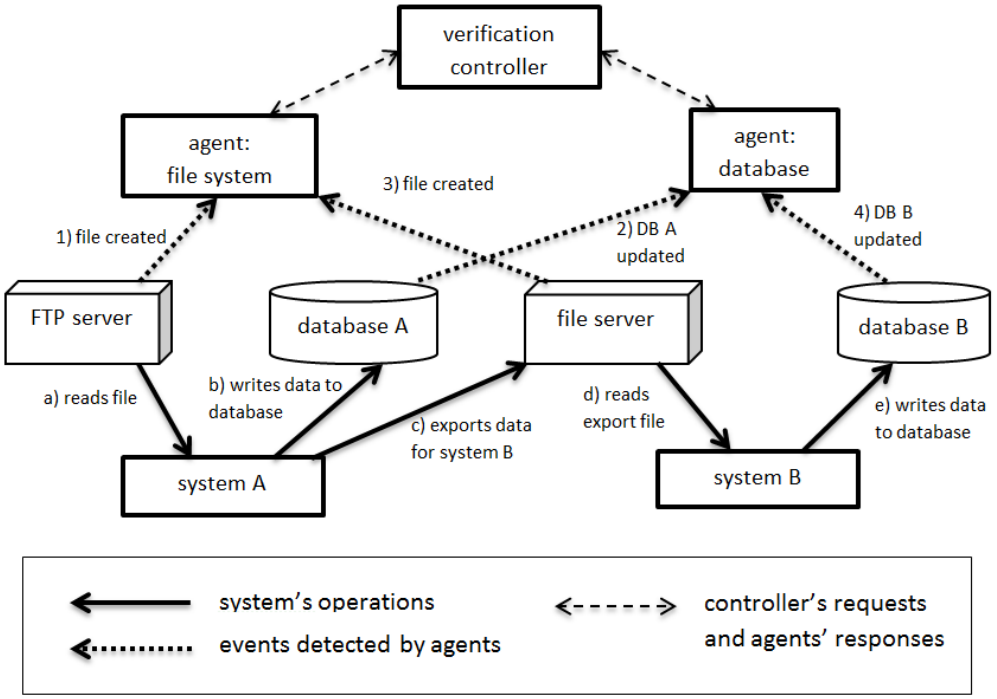

Figure 1. A schema of business process verification.

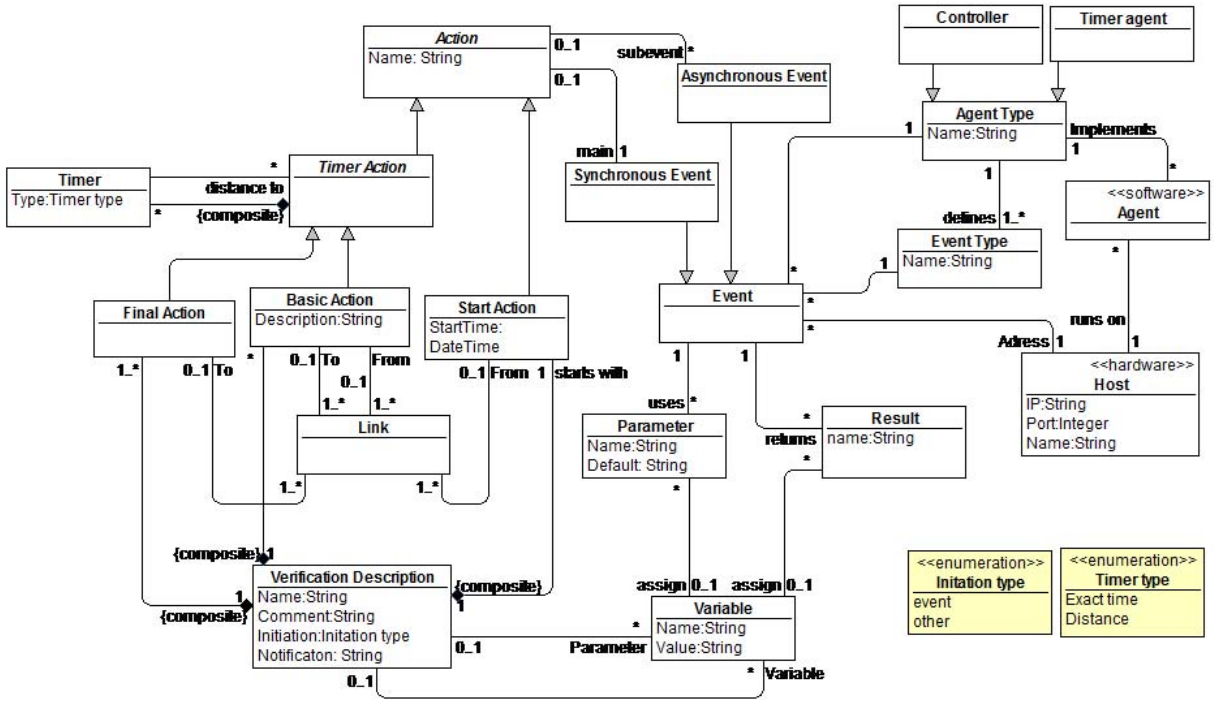

Figure 2. A class diagram of elements in the verification process description language.

Process verification descriptions could be created using WSCoL [15] or LTL–FO+ [16], however first of these is applicable only for service oriented architecture and second – requires very detailed process description. Therefore, considering both, recommendations by other authors [20] and small amount of concepts required for proposed verification description, the authors decided to develop compact xml–based domain–specific language (Fig. 2) that enables specification of process verification. The language contains only concepts required by verification with insignificant overhead for workflow definition (states and links). It was implemented in the prototype of verification mechanism.

The process verification description that corresponds to a particular base process is represented by a directed graph whose vertices represent events that approve the execution of base process steps and arcs – the order in which events are executed. Fig. 3 shows a schematic example of base and verification processes. As can be seen, the number of events in the verification process is lower than the number of steps in the base process. This means that not every step execution can be detected by agents that are involved in the process verification.

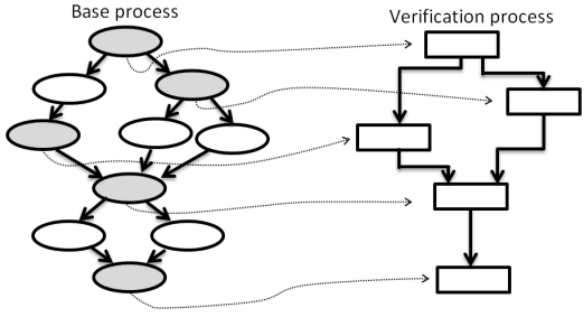

Figure 3. The base and verification process.

1) Elements of the process verification description language

Process verification description contains two types of elements – events and links between them. The term "event" within the scope of verification description language implies changes in the system's "memory" (file system, e–mail, database etc.) which can be handled by the verification mechanism agent. Links between events describe possible event sequences that reflect the actual execution of a business process. Each verification process must have one and only one start event, as well as one or more final events. It enables loops and parallel execution paths. Each verification description has the following attributes:

- A verification process instance load flag – whether the process instance is started by the occurrence of the start event or by another verification process instance;
- Parameters and variables that are used in process verification events;
- Report settings when errors occur or time limit warnings must be sent.

Given that the element "link between events" does not have any attributes, further exploration is devoted to the element "event." In fact, "event" describes action identified by event. Each of process actions being verified has a name, main event and potentially one or more sub–events. All of events have attributes:

- The agent and event type (e.g., agent – "FileSystem", event – "file created");
- The parameters of event detection;
- The agent's address, i.e., the server name or IP and related port;
- Assignment of results – when an event is detected by an agent, it returns notification with explanatory attributes related to the event;
- Event time restriction, which can be expressed as an exact time (e.g., before 14:30:20 and after 15:00:00), a distance of time from one or several previous events or combination of these two;
- Information about warnings and errors can be added to each of the events.

The process verification mechanism checks sub–events only when the main event has occurred. Obviously, the sub–events have no time restrictions.

A full list of items that are used for the description of a verification process is shown in a UML class diagram (Fig. 2).

2) Language implementation

Verification description is built using xml – based language. There is xml schema created for validation of description. Each description contains three sections: description heading, description of actions and action links. Description heading contains all common description items like name, comment and initiation type. Common parameters (directory names, server names etc.) and variables are defined here, too.

Action section includes all events grouped by actions they are describing. Each action should contain defined timer restrictions and main event. Also sub–events may be included there.

The last section contains action links. Links have three limitations: start actions may have only outgoing links, final actions – only incoming links and there must be path from each of start actions to at least one of final actions. Cycles are allowed.

C. Agents

Agents are one of two components in a process verification mechanism, and their job is to detect changes in the system's "memory." Agents can be universal software components (e.g., a file system event agent) or developed for a specific IS. The variety of available agents speaks to the possibilities of process verification – the wider the range of detectable events, the more detailed the process verification that can be implemented. Depending on the event type, there can be one of two types of agents – simple or thin agents, as well as complex agents.

1) Simple agents

The model of a simple agent presumes that the agent can detect all events without a controller request and that all of them will be of use in process verification. A dedicated agent that is developed for a single IS can be taken into account, and it is likely that all events that are detected by the agent will be of use in the process verification of the IS. A contrary database agent may not send all of the detected events to the controller, because most of them will not be required for verification and will be nothing more than spam.

2) Complex agents

Unlike thin agents, complex agents are used in environments in which only a few of many events must be delivered to the controller. A typical example is a file system event agent that runs on a corporate file server. Many file system events occur every second on these kinds of servers, but only some of them are required for business process verification. A complex agent model, therefore, is based on two–way communications (the request–response model). The controller asks the agent to detect certain types of events with specific parameters, the agent detects the required events, and it then sends a response to the controller. The complex agent model handles the processing of synchronous and asynchronous event requests.

D. An example of a verification description

Let's imagine a system which processes some files that contain messages (the process state chart is illustrated in Fig. 4). From the system's point of view, the file processing starts with the "File received" state, i.e., the file is copied onto the system operator's file server (via FTP or other file transfer services). Then the file is registered in the system's database, and the actual processing is launched:

- The file is decrypted;
- The file is parsed and all messages are extracted;
- All messages are checked in accordance with the particular business rules.

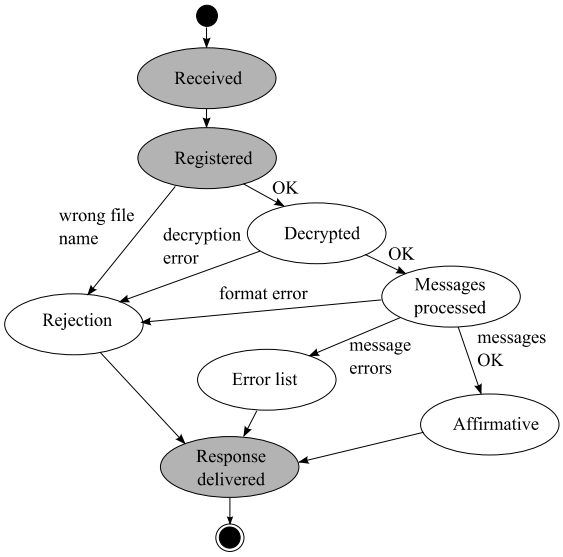

Figure 4. An example of a base process.

During the file processing, the system prepares one of three types of responses: affirmative (A–type file), an error list (E–type file) or a rejection of file procession (C–type file). When the response is prepared, it is created in a temporary directory as the respective type of file, it is encrypted, and it is delivered to the recipient.

In order to verify file process execution, indicative control states must be part of the file processing state chart. When adding verification points to a process description, it is necessary to think about whether it will be possible to detect a situation in which the process has achieved a specific state. For example, it is possible to detect the creation of a new file in a specified directory or a new record in a database table, but it is practically impossible to identify object value changes in the memory (RAM) of another process. In the example that is seen above, three states have been identified as possible verification states (marked in grey in Fig. 4) – File received, File registered and Response delivered. The process verification description can be created when these states and links among them are utilized.

According to the aforementioned process, control description can be created from the state chart, as well as from an activity diagram or other process descriptions. The only requirement is to add extensions such as event and time control concept.

E. Limitations of the solution

One of the most important aspects of any developed solution is awareness of its capability limits. Particularly in this case the mechanism has been developed mainly to serve business process verification purposes. Accordingly, limitations of proposed verification mechanism should be examined from two aspects:

- Does the mechanism provide business process verification?
- What are borderline cases of verification?

The scope of this work defines business processes as processes that to some extent involve operations executed by persons. Respectively, these processes are not computer intensive, e.g., document flow. Therefore, the authors claim that the proposed mechanism is able to provide business process verification since the resulting event load on either agents, or controllers is hardly comparable with agents' or controller's execution speed.

The more complex and sophisticated issue is presented by evaluation of borderline cases of verification. Evidently the verification mechanism is not able to verify processes whose execution speed is equal to execution speed of the verification process itself. Processing of each verification process' event involves many single steps executed by verification solution; therefore, intensity of verifiable events should be several times less than a number of verification steps. Moreover, verification of processes' instances can be executed for several processes simultaneously. Accordingly the more verifiable process instances are involved; the less process steps can be verified. However, certain limitations directly depend on implementation of verification solution and its operational environment. The next chapter "Proof of

concept" contains more detailed evaluation of the solution's limitations.

Eventual process problems that can or cannot be detected by the verification solution should be examined additionally to the technical limitations:

- The proposed mechanism can detect problems directly related to violation of verification criteria: failure to execute all process steps, failure to execute process steps in fixed sequence or failure to comply with time limits.
- The proposed solution does not verify internal algorithms of software operation, such as how wording in e–mail or file content have been prepared, i.e. it does not verify process consistency with specification.

However, the solution can help detecting collateral problems that impact execution of process steps. For instance, if system's operation has been interrupted due to internal error, it will probably result in failure of some process steps. Accordingly, incorrect execution of process instance will be detected and support team will receive notice containing information which process steps have been executed and which ones failed.

V. PROOF OF THE CONCEPT

Prototype of the process verification controller and two agents were developed so as to prove the concept and specify the details of the proposed model. The scope of the prototype included implementation of process verification description, as shown in Fig. 5. Two agents were implemented – the FileSystem agent for file system events and the DBEvent agent for database event. Whereas processes used form verification were running on Microsoft Windows platform, the controller and agents were implemented as Windows services using C# and Windows Communication Foundation.

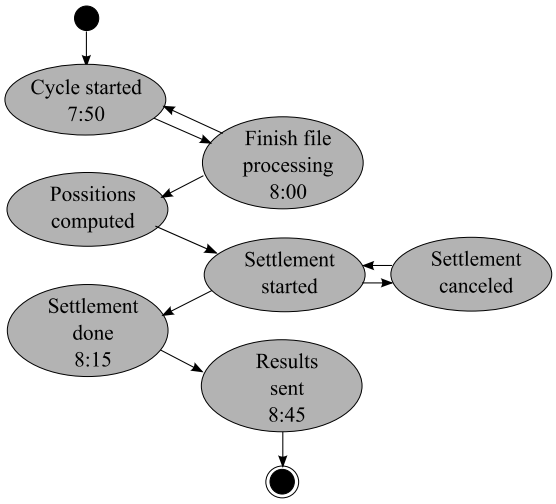

Figure 5. Settlement cycle processing.

Prototype was used in two separated environments:

- Some processes of Electronic clearing system were verified in order to evaluate the usability of solution and descriptiveness of supposed DSL for real life processes;
- Test environment with process generator was developed in order to test the solution's limitations under different workload.

A. *Implementation of the controller*

Even though the scope of prototype was rather narrow it has final product features:

- It allows loading more than one process description;
- Many in parallel running process instances can be verified;
- Communication with agents is based on interfaces thereby new agents may be introduced without any changes in the controller.

Controller runs as separate process and all process descriptions are loaded automatically. When verification descriptions are loaded, controller sends start event request to agents (if verification process instance is started by an event, it must be start event). When start event notification is received, corresponding process instance is created and following event notifications are requested.

All event time limitations are checked by controller in parallel. To reduce timer mechanism impact on verification performance, actually only one timer is set – only the closest event in all verification process instances is selected. Timer is reviewed or reset upon reception of any event notification or when timer elapses. When controller detects some inconsistencies in process instances under verification (event timer elapses or notifications are received in wrong order), controller prepares informative messages for support staff (configured in verification process description) about exception. Thereby support staff is informed as soon as problem is detected.

B. *Implementation of agents*

Four event types were implemented in the FileSystem agent:

- FileCreated – an event is fired when a new file is created. This obviously is an asynchronous event, and it has three parameters – directory to watch, file name pattern, and subdirectory flag.
- FileModified – an asynchronous event is fired when a file is modified.
- FileDeleted – an asynchronous event is fired when a file is modified.
- FileExist – a synchronous event with three parameters – the directory name and the file name to check and existance flag (true – checks if file exists, false – check is file does not exists).

The FileSystem agent was created for the Windows operating system, and file system event watchers were used whenever possible.

The DBEvent agent was only implemented for one event – Request. It provides Microsoft SQL Server

procedure calls. When an event occurs, it returns the result of the procedure call. The agent was developed on the basis of timers. For this reason, some delays were observed.

C. Verified processes

1) Electronic clearing system

Prototyping involved two processes of Electronic Clearing System (EKS) [21]:

- Reception of Payment File (illustrated by examples Fig. 4);
- Execution of Settlement Cycle (see Fig. 5).

The first process allows evaluating performance options of the proposed verification mechanisms. File processing has to be executed relatively fast, because it determines the whole settlement process: if file processing gets slow, it can affect local settlement time and further participation in other international settlements. File processing is executed in parallel, and each file is processed relatively fast. The EKS holder requires that processing time for each file should not exceed five minutes, i.e., file sender must receive response on file processing result and payments included in settlement within five minutes of time. Also, the process itself is interesting because it has a database and file system activities. Therefore, verification requires involvement of two agent types. Furthermore, this process was verified in EKS test environment and agents were installed in distributed environment: file system agent was installed on file server, database agent – no SQL server and verification controller on separate verification server; thus providing collaboration framework for test agents and controller.

The second verification process relates to processing of settlement cycle. System runs seven settlement cycles on a daily basis. Each cycle has a fixed schedule including deadline for file processing, deadline for settlement and deadline for result file delivery. The next table presents examples of settlement cycle deadlines (Table I).

TABLE I. SETTLEMENT CYCLE DAILY DEADLINES.

Cycle	Files processed	Settlement	Results delivered
1.	08:00	08:15	08:45
2.	09:30	09:45	10:15
3.	11:30	11:45	12:15
4.	13:00	13:15	13:45
5.	15:00	15:15	15:45
6.	17:30	17:45	18:15
7.	18:30	18:45	19:30

As illustrated in Fig. 5 the supervision of settlement cycle is not very intensive process since event occurrence is relatively rare; though supervision process itself is crucial. If one of the settlement cycle deadlines is delayed, the next cycle will be potentially delayed. Each settlement cycle is regarded as a separate process being a component part in the whole settlement process; therefore an automated verification would significantly facilitate supervision of the whole settlement process. Execution of system's settlement cycles was verified in the live environment, for only the live environment provides acceptable execution schedule. In order to reduce eventual verification impact on verifiable environment, all components of verification mechanism (controller and data base agents) were installed on separate control server.

a) The results of the file processing verification

First prototypes provided an insight into process verification performance possibilities. There were 80 payment files generated for three EKS participants (Table II)

Three separate process verification descriptions were created for three system participants (system participants are commercial banks) – one description for each participant's file processing process. Therefore prototype used three process verification descriptions simultaneously (although these descriptions were similar, they differed from the controllers perspective) and process verification instances were attached to three different processes.

TABLE II. FILES USED FOR TESTING.

Participant	Files	Payments
Bank 1	50	3070
Bank 2	10	200
Bank 3	20	400
Total	**80**	**3670**

Test files were created in a way that differs from real life system usage, namely, all test files were copied to system directories simultaneously, while in real life due to network latency system's participants usually copy files gradually. Thus, creation of 80 process verification instances instantly resulted in a reasonable workload for file system agents and controller; though further in processing all verification instances were verified gradually. Considering that EKS processes several files in parallel, the controller received file processing event notifications alternately from several processes in parallel. Since EKS processed files alternately, finishing was not simultaneous for all 80 files. Some files were processed with delay. Verification mechanism identified delay for 11 files and this number was verified as correct using EKS data base.

File processing was verified using two event agents: file system agent and database agent. File system agent is event based; therefore the agent gets active only after the required system activity has happened, i.e. an event has occurred. Accordingly, in a moment when the payment files were copied for processing, the file system agent received information that 80 events "file created" have occurred. All these events were received and processed by the file system agent simultaneously, thus creating insignificant overload for the agent. During further file processing when other file system events occurred progressively, process overload was not detected.

Contrary to the file system agent, the database agent is timer based. According to its configuration the database agent executed database requests every five seconds. Although SQL profiling tools recorded this activity, no really significant processor's overload was identified due to the small amount of verifiable process instances.

b) Verification of settlement cycles

The second example of verifiable processes (execution of settlement cycles) was verified in EKS live environment. Since there are seven daily settlement cycles in the EKS and every cycle has slightly different settings, the verification controller had to treat them as seven different processes, i.e., seven process verification descriptions were prepared. Contrary to the file processing, the settlement processes are executed rather slowly; therefore, verification configuration was set to check for cycle status changes once per minute and as a result, there were no signs of overload detected in the EKS live environment. Further execution of cycle processes was in compliance with all time constraints, thus all process instances were recognized as correctly executed.

2) Test environment

Test environment containing business process test generator was created to provide solution workload testing. According to the test plan, seven different complexity processes were described, and test generator was able to operate with all process instances simultaneously. In order to reduce the influence of various implantation factors to the assessment, all processes were related to the file processing (creation of new files, modification, deletion, copying and moving). Therefor only one file system events agent was used. According to the approach of the proposed solution, two nodes were used:

- Test generator and file system agent were running on one of the nodes (agent should be as near as possible to the place of events used for verification);
- The second node was used by the verification controller.

The frequency of event occurrence for process instances was limited to align instance execution intensity with agent's ability to handle filesystem events. Thus solution were tested on different number of simultaneous process instances.

The initial hypothesis was confirmed. Verification process operated as intended to a certain number of concurrent process instances: all events were handled and identified and all delayed process instances were noticed (Fig. 6).

Figure 6. Process instances and missed events.

By increasing number of parallel process instances (the intensity of the events were running over 33 events per second), some of the process instance events were missed and thus the controller identified those instances incorrectly as faulty (false negative). By increasing the number of instances the rapid growth of the number of missed event and instances identified as "false negative" was caused.

When the operations of the prototype were analyzed, it was found that the file system agent identified all the requested events. Verification errors were caused by the workload of the controllers prototype: the controller delayed further event requests because of processing of event notifications received from agents. Thus, the agent received requests for event identification after events occurred. Meanwhile, event controller were overloaded, agent it-self left insignificant effect on test generator operation. i.e., overhead created by agent were imperceptible even there were relatively high number of events to detect.

The detected problem of controller could be partly solved if controllers would request not only events of next direct process steps, but all further events as far as it is possible according process description (further event parameters could depend on previous events results). Thereby controller would have more time on instance verification upon arrival of the event's notification. Although these updates could improve controller's performance, however, also in this case there would be number of instances when controllers operations will become inefficient because of many "false negative" notices.

The other way to improve the operation of solution is to use simple (i.e., built in) agents as widely as possible, Simple agents detects events without controller's request, therefore controller's workload can be minimized and less time will be spent on communications: controller would not request all possible future process instance events, but it will just receive occurred events.

D. Prototyping conclusions

Even though first experiments have been rather inspiring, it is obvious that increased complexity of verification process descriptions will trigger increase of number of events to be detected. As a result this will increase agent impact on system under verification. Also, there is evidently a limit to the number of event notifications and process instances which can be processed by controller.

Thereby first prototype tests satisfied expectations of authors. Even tested verification mechanism is rather simple; it leads to some conclusions about its application:

- Solution can provide verification of business processes that run with typical business process intensity when process steps are executed or initiated by man;
- Solution provides verification for a wide range of processes;
- Verification of processes (create and add new description) can be easily accomplished and modified if necessary;

- Verification mechanism is extendable by adding new agents, thus providing verification for a wide range of environments and platforms;
- Higher process verification intensity can be achieved by use of simple (i.e., simple agents).

There were also several suggestions that must be taken into account:

- A fairly detailed base process execution model must be available so as to develop a detailed process verification description;
- Various event peculiarities may affect process verification description, e.g., when two events occur at the same time, but cannot be verified simultaneously. In such cases, sub–events can be used. Therefore, when file is moved, the main event is FileCreated in the destination directory, and the sub–event is FileDoesNotExist for the source directory.

VI. Conclusions

This paper has described an approach of business process runtime verification – the execution of a base process and a separate verification process in parallel. Also approbation is discussed for proof of concept.

The verification mechanism is based on two components: controller and agents. Event agents pass affirmation of base process step execution to the verification process. A formalized verification process description allows the controller to identify incorrect base process execution, to warn IS support staff immediately about any process errors that are defined, and to track the exact process step in which the error occurred. The proposed reaction contributes toward efficient reaction when there are incorrect business process executions. In comparison to other, similar solutions ([12], [13], [15], [18]), this solution has several positive features:

- The verification process can be defined without modifying the base process – the base process can have more than one verification process so as to verify all of its various aspects;
- The verification process runs in parallel to a base process and does not interfere with it;
- The solution is applicable to the base processes of heterogeneous systems, i.e., the business process is tracked as a whole even if it is being implemented by two or more systems;
- The verification process does not limit the language that is used for base process modeling and implementation;
- When verification process execution is tracked and logged, it can be used to gather statistics about the base process execution time, thus identifying bottlenecks and providing warnings about changes in the duration of the process execution [22].

The proposed solution can be broadly applied. Although the primary research objective was to develop a runtime verification mechanism for business processes, the prototype solution showed that it can also be used for runtime verification in real–time and even embedded systems. The same verification description language and controller can be used in all cases, only adjusting the necessary event agents.

This research is based on years of experience in building systems related to business process verification abilities [23]. The experience of the authors contributed toward the definition of business process verification as a separate component [8], [9]. The research led to the development of a prototype verification mechanism, including two types of agents. This resulted in further research efforts that were focused on event agent standardization and unification, as well as on the development of the relevant language description.

Acknowledgment

The research leading to these results has received funding from the research project "IT Competence Centre" of EU Structural funds, contract nr. L–KC–11–0003 signed between IT Competence Centre and Investment and Development Agency of Latvia, Research No. 1.6 "Algorithms for verification of business processes".

References

[1] C. Atkinson and D. Draheim, Business process technology: A unified view on business processes, workflows and enterprise applications. Springer, 2010.

[2] D. Gelernter and N. Carriero, "Coordination languages and their significance," Communications of the ACM, vol. 35, no. 2, p. 96, 1992.

[3] T. W. Malone and K. Crowston, "The interdisciplinary study of coordination," ACM Computing Surveys (CSUR), vol. 26, no. 1, pp. 87–119, 1994.

[4] J.-M. Andreoli, Coordination programming. World Scientific, 1996.

[5] P. Ciancarini and C. Hankin, Coordination Languages and Models: First International Conference, COORDINATION'96, Cesena, Italy, April 15-17, 1996. Proceedings. Springer, 1996,vol. 1.

[6] J. A. Bergstra and P. Klint, "The discrete time TOOLBUS – a software coordination architecture," Science of Computer Programming, vol. 31, no. 2, pp. 205–229, 1998.

[7] C. Liu, Q. Li, and X. Zhao, "Challenges and opportunities in collaborative business process management: Overview of recent advances and introduction to the special issue," Information Systems Frontiers, vol. 11, no. 3, pp. 201–209, 2009.

[8] I. Oditis and J. Bicevskis, "The concept of automated process control," Computer Science and Information Technologies, Scientific Papers, University of Latvia, vol. 756, pp. 193–203, 2010.

[9] I. Oditis and J. Bicevskis, "Runtime verification of business processes," in Databases and Information Systems, Proceedings of the 11th International Baltic Conference, Baltic DB&IS 2014, H.-M. Haav, A. Kalja, and T. Robal, Eds. Tallin University of Technology Press, jun 2014, pp. 363–370.

[10] Oditis I., Bicevskis J. Asynchronous Runtime Verification of Business Processes. In Proceedings of the 7th International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN), Riga, 2015, pp. 103-108.

[11] C. W. W. Wu, "Methods for reducing monitoring overhead in runtime verification," Ph.D. dissertation, University of Waterloo, 2013.

[12] C. Ghezzi and S. Guinea, "Run-time monitoring in serviceoriented architectures," in Test and analysis of web services. Springer, 2007, pp. 237–264.

[13] L. Baresi and S. Guinea, "Towards dynamic monitoring of WS-BPEL processes," in Service-Oriented Computing-ICSOC 2005. Springer, 2005, pp. 269–282.

[14] E. Diebelis and J. Bicevskis, "Software self-testing." in DB&IS, 2012, pp. 249–262.

[15] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini, "A timed extension of WSCoL," in Web Services, 2007. ICWS 2007. IEEE International Conference on. IEEE, 2007, pp. 663–670.

[16] S. Halle and R. Villemaire, "Runtime monitoring of message-based workflows with data," in Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE. IEEE, 2008, pp. 63–72.

[17] D. Kortenkamp, T. Milam, R. Simmons, and J. L. Fernandez, "Collecting and analyzing data from distributed control programs," Electronic Notes in Theoretical Computer Science, vol. 55, no. 2, pp. 236–254, 2001.

[18] M. Ducassé and E. Jahier, "Efficient automated trace analysis: Examples with morphine," Electronic Notes in Theoretical Computer Science, vol. 55, no. 2, pp. 118–133, 2001.

[19] S. Bensalem, M. Bozga, M. Krichen, and S. Tripakis, "Testing conformance of real-time applications by automatic generation of observers," Electronic Notes in Theoretical Computer Science, vol. 113, pp. 23–43, 2005.

[20] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," ACM computing surveys (CSUR), vol. 37, no. 4, pp. 316–344, 2005.

[21] Electronic Clearing System EKS, Latvijas Banka, 2013. [Online]. Available: http://www.bank.lv/en/payment-and-settlement-systems/electronic-clearing-system-eks

[22] B. F. van Dongen, A. K. A. de Medeiros, H. Verbeek, A. Weijters, and W. M. Van Der Aalst, "The prom framework: A new era in process mining tool support," in Applications and Theory of Petri Nets 2005. Springer, 2005, pp. 444–454.

[23] J. Cerina-Berziņa, J. Bicevskis, and G. Karnitis, "Information systems development based on visual domain specific language bilingva," in Advances in software engineering techniques. Springer, 2012, pp. 124–135.