Resolving Business Process Interference via Dynamic Reconfiguration*

Nick van Beest[1], Pavel Bulanov[2], Hans Wortmann[1], Alexander Lazovik[2]

[1] Department of Business & ICT, Faculty of Economics and Business, University of Groningen, Nettelbosje 2, 9747 AE Groningen, The Netherlands
[2] Johann Bernoulli Institute for Mathematics and Computer Science, Faculty of Mathematics and Natural Sciences, University of Groningen, Nijenborgh 9, 9747 AG Groningen, The Netherlands
{n.r.t.p.van.beest, p.bulanov, j.c.wortmann, a.lazovik}@rug.nl

Abstract. For business processes supported by service-oriented information systems, concurrent execution of business processes still may yield undesired business outcomes as a result of process interference. For instance, concurrent processes may partially depend on a semantically identical process variable, causing inconsistencies during process execution. Current design-time verification of service-based processes is not always sufficient to identify these issues. To identify and resolve potentially erroneous situations, run-time handling of interference is required. In this paper, dependency scopes are defined to represent the dependencies between processes and data sources. In addition, intervention patterns are developed to repair inconsistencies using dynamic reconfiguration during execution of the process. These concepts are implemented on top of a BPMS platform and tested on a real case study, based on the implementation of a Dutch Law in e-Government.

Keywords: business process, configuration, variability, concurrency

1 Introduction

Concurrent execution of business processes, supported by distributed or service-oriented information systems, may result in an unexpected behavior due to process interference. Different concurrent processes may partially depend on the same resource. Data, modified by an external process, may cause inconsistencies during process execution, resulting in the aforementioned undesired business outcomes. Such business processes are designed with an inherent assumption of independence of processes. Consequently, if business

* The research is supported by the NWO SaS-LeG project, http://www.sas-leg.net, contract No. 638.000.000.07N07.

processes are executed concurrently, it is implicitly assumed that they cannot affect each other. For many business processes it is not always feasible to foresee all interdependencies. Assuming independent execution of the processes, current design-time verification of process models is insufficient to identify all conflicts and, more importantly, resolve these issues during run-time [8].

Traditionally, locking mechanisms are used to resolve the issues with concurrent access to shared data. Unfortunately, in practice it is not possible to lock data for a long period of time, which is quite typical for long-term business processes. Consider, for example, a business process for issuing a wheelchair for disabled people in the Netherlands. It takes up to 6 weeks from sending the initial request to receiving an actual wheel chair. What if in the meantime the person, which requested the wheel chair, has moved to a different place, e.g., to a care home with a nursing support? The original process has to be adjusted, either by forwarding the wheel chair to a new place, or by cancelling the request (whatever is more suitable in a concrete situation). To resolve this issue by locking, one would "force" the person not to change his address until the first process is finished, which is unfeasible in real life.

In this paper, process interference is prevented by awareness of process dependencies and automatic execution of compensation activities. Dependency scopes are introduced to represent the dependencies between processes and data sources. In addition, intervention patterns are developed to repair inconsistencies during execution of the process. These modeling concepts can be seamlessly integrated in existing Business Process Modeling platforms. The Cordys Business Operations Platform is used as a basis for the implementation of the proposed concepts. The implemented solution is demonstrated and tested on a real case study from e-Government that concerns the Dutch WMO law business process.

The remainder of the paper is organized as follows. Section 2 provides a background showing the current methods for dealing with the problem of concurrent process execution. A detailed description of the problem and the research methodology are both presented in Section 3. The developed modeling concepts are presented in Section 4. The introduced approach is applied to the WMO law case study in Section 5. In Section 6, an overview of the implementation of the dependency scopes and intervention patterns is presented. Finally, the paper is concluded in Section 7.

2 Related Work

Much work has been done on verification of workflow and data specifications, e.g. [14]. However, workflow verification as presented in [14] assumes that the analysis is performed in the context of a single party, who has full access to the processes and data involved. However, many business processes are designed in a distributed manner, and their execution is spread among several parties. Often, such business processes are supported by, for instance, various autonomous Web services environments [9]. Work-

flow and data verification, as described by e.g. [14], cannot be applied to interference resolution due to absence of a single ownership over the business processes involved in the execution. The parts of the business process owned by other partners are not specified in detail, as the implementation details are typically only known to the external party.

[18] propose an approach that deals with failing processes as the point of failure for the analysis of data dependencies and process interference. Similarly, compensation activities have been introduced to restore consistency in e.g. aborted long-lived transactions [7]. In practice, however, interfering processes do not necessarily fail. More often, they execute without any noticeable problems from inside the organization, as no error message is signaled. In contrast, the problem is primarily noticed by the external stakeholders (mostly customers) as the final result is undesirable to them [16].

The handling of the process interference at run-time is desirable by identifying and resolving erroneous situations. This may, for example, be realized by a run-time reconfiguration of the business process to repair it and bring it back to a consistent state. Such a concept has been proposed in [15], but the solution emphasizes on resolving *design-time* modeling contradictions in business process specifications rather than run-time ones. Unfortunately the interference issues at *run-time* cannot be identified as a contradicting specification, as each business process runs without any interference in isolation. As a result, abnormal run-time interference situations are not covered by this technique.

In general, the idea to deal with run-time business process reconfiguration is not new. The most notable examples of handling run-time reconfiguration are the ADEPT project [6], the DECLARE framework [10], and more general techniques like ad-hoc sub-processes [5] and run-time case handling [2]. A more detailed overview of various dynamic business process reconfiguration techniques can be found in [11], while detailed requirements for such systems are overviewed in [4].

The ADEPT project is designed to support the synchronization between several running instances of the same process. The idea is to catch any changes made by the user and incorporate those changes into all of the running instances without interrupting their execution. The DECLARE framework utilizes the idea of a declarative process specification [13] in order to attain flexible process execution. As a result, the process defined inside this framework is not a strictly written sequence of actions, but rather a set of rules, which guide the user through the process execution in an interactive manner.

The primary aim of both case handling and ad-hoc sub-processes is to enhance the specification of a business process with conditions, which are left undecided until run-time. At run-time, users can choose whether a part of the process should be included or not, based on their own experience. In the same way, parts of business processes may be left undefined in order to be specified at run-time, using the so-called late binding technique. This technique is supported by the YAWL workflow language [3] and also adopted in a form of reconfigurable sub-processes by [12].

The frameworks and techniques listed above focus on supporting run-time process specification, which provides a design-time defined execution flexibility to the user.

However, ad-hoc compensation to prevent inconsistent process execution requires an automated mechanism to determine which activities need to be executed. In this paper, design-time modeling concepts are developed to manage run-time discovered interference and provide automatic execution of compensation activities to restore consistency.

3 A Case Study: Business Process for the Dutch WMO Law

To demonstrate the proposed solution, and also to evaluate and to show feasibility of the approach, a business process supporting the Dutch Law for Societal Support (known as the WMO[†] law) is examined. The purpose of this law is to enable people with a chronic disease, a disability or physical decline due to aging to live in their own homes and take part in society for as long as possible. Facilities are provided to offer support to such citizens by means of, for example, domestic care, transportation, a wheelchair or a home modification. Both the responsibility and the execution of the business process are located at the municipalities. The business process under investigation concerns the handling of the requests from citizens at one of the 430 municipalities in the Netherlands.

Fig. 1 shows a business process representing the request for a facility taken from one of the municipalities in the Netherlands. The business process consists of activities, which are provided by various services. First, the citizen contacts the municipality and submits an application for a provision. Such a provision may concern, for example, a wheelchair, domestic help, or a home modification. Based on that application, it is decided whether a home visit is necessary, to have a better understanding of the situation. Then, it might be necessary to ask for additional information, by means of a medical advice. This information will finally result in an indication and decision made by the municipality. If the decision is unsatisfactory for the citizen, there is a possibility for appeal.

Depending on the requested service, the citizen can choose between a Personal Budget and Care in Kind. Very often, this choice is already made in the initial application. In case of a Personal Budget, the citizen periodically receives a certain amount of money for the granted provision(s). If Care in Kind is chosen, suppliers are contacted, which will eventually supply the citizen with the granted provisions. The process ends with the payment of the invoices received from the suppliers.

Concurrently executed processes may constitute a potential risk of erroneous workflow execution. That is, although the process does finish regularly without any system errors, the final result may be undesirable from a business perspective. Activity branches with a rather long execution time that strongly depend on the consistency of certain process variables are the most vulnerable to inconsistencies caused by concurrent processes.

In Fig. 1, the four distinct activity branches following “Designation and report” (Domestic help, Transportation, Wheelchair and Major home modification) may

[†] http://nl.wikipedia.org/wiki/Wet_maatschappelijke_ondersteuning

potentially suffer from process interference. The throughput time for both the wheelchair provisioning and major home modification may take up to 6 weeks, whereas they strongly depend on the address of the citizen and the content of the designation. This implies that a change in either of these process variables (e.g. address) may have strong consequences for the consistency of the business process.

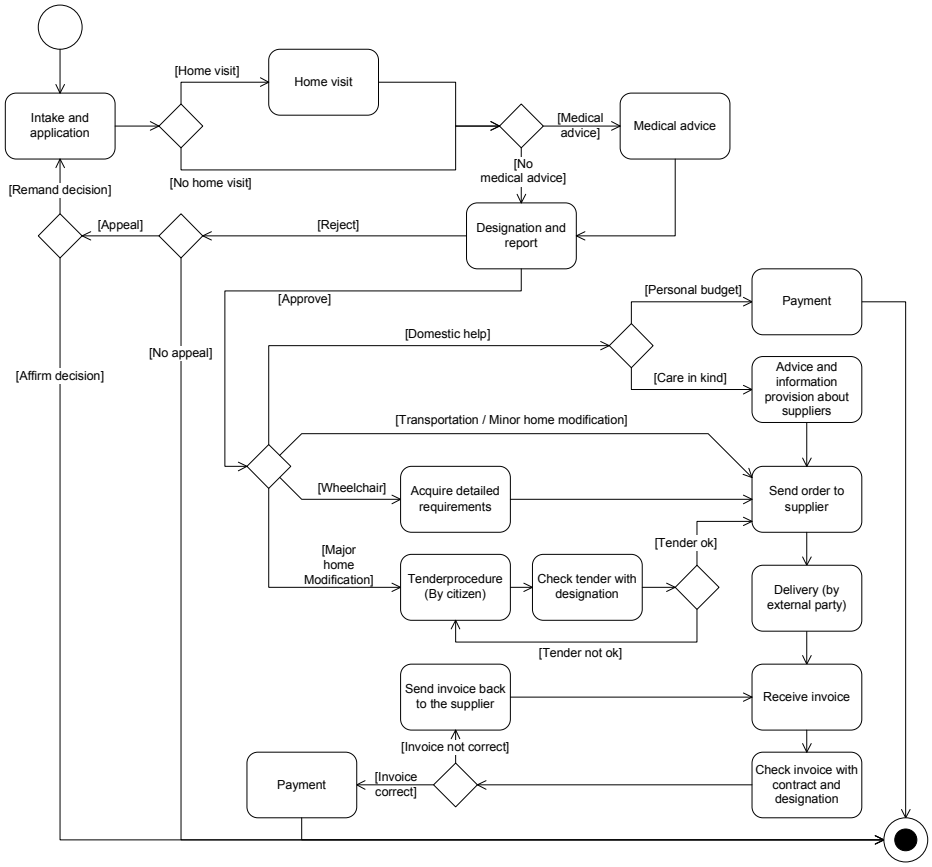

Fig. 1. Process model of the request for a provision at a municipality in the Netherlands.

For instance, assume that the tender for major home modification will be approved and the order will be sent to the supplier. If the citizen moves from a regular house to a nursing home during or right after “Check tender with designation”, the home modification will not be necessary anymore. However, in the process above, the supplier

or the municipality are not notified of the address change and the request for home modification is not cancelled.

A similar situation occurs in case of a request for a wheelchair. The requirements (and constraints) of a wheelchair may depend on certain architectural characteristics of the citizen's home. Therefore, an address change between "acquire detailed requirements" and delivery might result in a wheelchair that does not fit the actual requirements.

4 Dependency scopes and intervention patterns

The situations described in Section 3 may occur when several concurrent processes, that are wrongly assumed to be independent, use the same data at a particular point in time or within a certain timeframe. As a result, this may affect the data used in subsequent activities or even the sub-processes that follow an evaluation of a condition.

In order to analyze this situation, let us formalize the concept of a business process.

Definition 1: A *business process* is a set of linked activities, constructors and process variables that collectively realize a business objective or a policy goal, where:

- Each activity is an atomic piece of work representing an interaction with some service.
- Constructors represent the flow of execution, e.g. sequence, choice, parallelism, join synchronization. These constructors have well-defined semantics, e.g. defined in [1].
- A process variable is a variable over arbitrary domain, and is typically mapped into input/output parameters of activities (services).

In this paper, we do not distinguish between the business process and the business process model, assuming that the business process engine is able to execute the provided business process model. Consequently, the difference between a business process and its modeling representation is irrelevant in the context of this paper.

Definition 2: A *sub-process* is a business process that is enacted or called from another (initiating) business process (or sub-process), and which forms part of the overall (initiating) business process [17].

The process definitions presented above are not new. They have been implemented in different workflow and business process management systems, e.g. using BPMN, or BPEL notation.

In Fig. 2, two processes are presented. The decision made in Process 1 is based on the value of process variable D. That specific decision determines whether activities A1 and A2 are executed or rather A3 and A4. If D is changed by another process (e.g. Process 2) during execution of A2, this may have consequences for the decision made. That is, as a

result of the data change, currently the wrong branch of activities is being executed. In such a situation, the execution of A2 needs to be cancelled and followed by compensating activities to compensate A2 and A1. Subsequently, the process should continue at A3. Therefore, it is desirable to know what activities are implicitly relying on that process variable (D). Furthermore, these activities should be notified if that data has changed, even if those changes happened externally to the process being currently executed.

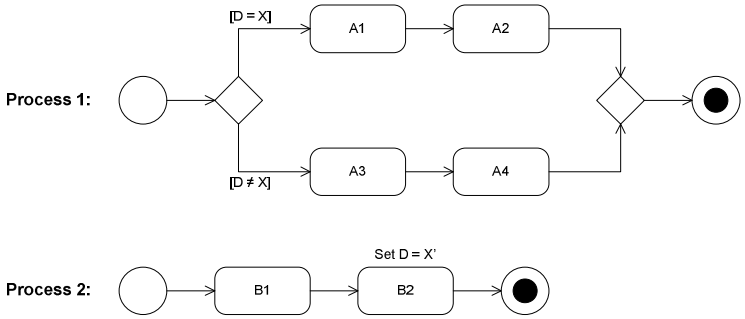

Fig. 2. Two business processes with concurrent data modification.

To identify the specific part of the process that depends on certain process variable, we introduce a notion of dependency-scope. A *dependency-scope* (DS) is defined as a structurally correct subset of the business process, in which the activities are implicitly or explicitly relying on the accuracy of a volatile process variable accessed in the first activity of that set. This implies that the process variable is assumed to remain unchanged (or within a certain range of values) by an external process during the execution of the entire DS. Note that this definition implies that an update of this process variable by the process *will* end the DS of that particular process variable, whereas it *may* start a new DS. In Fig. 3, Process 1 is represented with a corresponding dependency scope.

At an instance level, a DS can be active or inactive. It is activated when the first activity is started, which is part of the set that defines the DS. It is active as long as an activity is executed that belongs to the DS. If the last activity of the set of consecutive activities is finished, the DS switches to be inactive.

Definition 3: A *volatile process variable* is a process variable that can be changed externally during execution of the process.

A change of a volatile process variable can have various origins. That is, it can be changed by another process in the same organization or it can be changed externally to the organization that is executing the business process.

Definition 4: A *dependency-scope* (DS) is a part of the business process with a set of volatile process variables D, where:

- The activities of the dependency scope are relying on the correct value of D.
- There exists only one start activity (first activity in the dependency scope).
- The dependency scope is activated when its first activity starts.

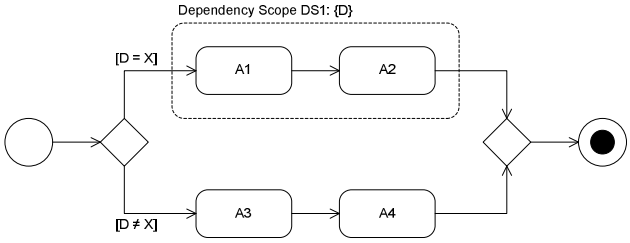

Fig. 3. Business process with a dependency scope definition.

Consider an example shown in Fig. 2: even if dependencies are identified, as shown in Fig. 3, an intervention may happen in any part of the dependency scope, both after A1 and A2. More specifically, if A2 is notified that B2 of Process 2 has just changed the variable D, A1 has already executed. In order to resolve the conflict, the process should restart again from the decision point prior to activity A1, which requires both A1 and A2 to be rolled back to the initial state.

This rollback may in some cases be prevented by a number of alternative activities to be executed before starting A3, in order to regain consistency. For example, the state of process 1 is undesirable from a business perspective, due to the incorrect decisions made as a result of the data mutation. A repairing activity should be interposed between A2 and A3, to recover process 1 to a consistent state that corresponds to reality again.

Definition 5: An *intervention pattern* is a sub-process that is linked to a DS, comprising a set of compensation activities, which together restore the consistent state of a business process. An intervention pattern has the following properties:

- A condition over the set D of the DS determines when the set of compensation activities needs to be executed.
- If the condition is true then the currently executed activity in the DS is stopped and the compensation process is executed.
- The last activity provides a re-entry point in the business process.

Note that intervention patterns are not expected to contain dependency scopes by themselves. They are considered to be relatively simple repairing activities. Fig. 4 shows a sequence of compensation activities, which is defined as an intervention pattern.

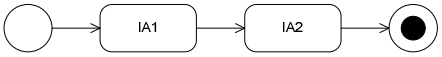

Fig. 4. Specification of intervention activities.

The activities required to restore consistency may vary, even concerning the same volatile process variable. A DS can comprise several intervention patterns. Depending on the condition, the corresponding intervention pattern is executed. However, if more intervention patterns are connected to one DS, then the conditions should be mutually exclusive. In addition, the activities required to restore consistency may vary between processes. In most cases, it may be sufficient to update the process variables in the currently executed activity and proceed, whereas in a more severe case the activity needs to be cancelled and the process should be resumed with another activity.

An example of the process including both a DS and an inserted intervention pattern is shown in Fig. 5. This solution allows for execution without manual process reconfiguration. As a result of the firing of the trigger on DS1, the activity currently being executed (A2) is halted. Next, the process is continued at the 'Continue' mark, where the execution of the intervention activities starts. After the intervention activities have been finished, the process proceeds with the correct activity in the regular process flow (A3).

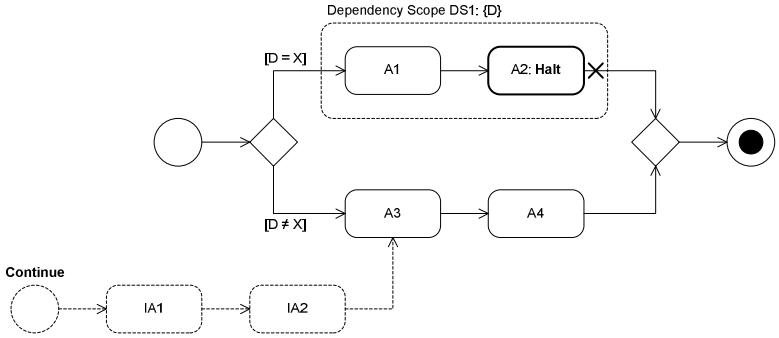

Fig. 5. Business process with dependency scope and connected intervention activities.

The concepts described above prevent the process designer from being forced to check the value of the condition after every activity within the DS. That is, in order to predefine the error-handling in case of process interference without the presented concepts, for every activity the values of volatile process variables have to be tested. A (simplified) example of such an undesirable situation is represented in Fig. 6. In more complex business processes, this would require a high amount of checks predefined in the business process. It is to be expected that this way of overcoming interdependency issues will strongly

increase the complexity of each process model and, accordingly, result in a cascading change after the model is to be updated.

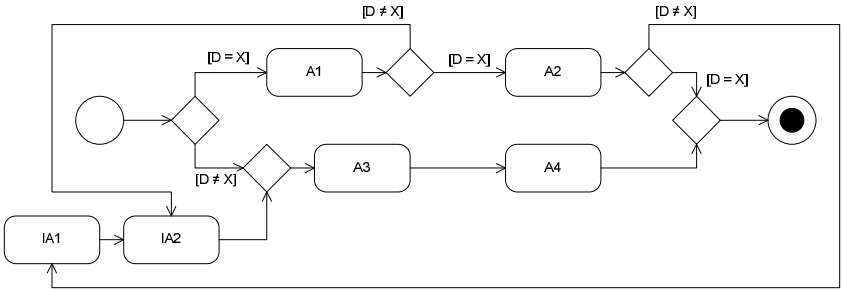

Fig. 6. Alternate solution to resolve dependencies.

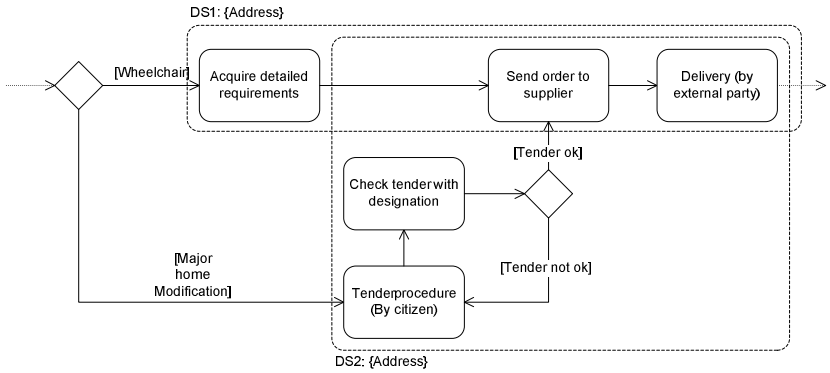

Fig. 7. Dependency scope for the case study.

5 Case Study: Repairing the WMO Process

In Fig. 1, the four distinct activity branches following “Designation and report” (Domestic help, Transportation, Wheelchair and Major home modification) may potentially suffer from the process interference. In this section, we enrich two of them ([Wheelchair] and [Major home modification]) with dependency scopes that are referring to the variable {Address}. Each dependency scope has a trigger on the corresponding process variable {Address}. As shown in Fig. 7, both DS1 and DS2 start at the first activity in the branch and end after the final delivery, when the process variable {Address} is no longer required to remain unchanged.

If the trigger of one of the dependency scopes fires, the main process is halted, and corresponding compensation activities are executed. These activities are defined within the intervention patterns. In Fig. 8, the intervention pattern on [Wheelchair] (DS1) is shown. The intervention pattern on [Major home modification] (DS2) is shown in Fig. 9.

The decision concerning which intervention activity is to be executed first for each intervention pattern depends on the cancelled activity in the main process. For instance, if the order for home modification is already sent to the supplier, it must be cancelled first. If the order is not sent yet, it will suffice to start with a home visit.

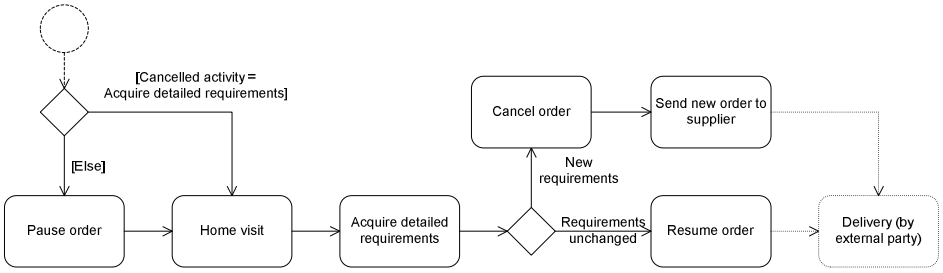

Fig. 8. Intervention pattern on [Wheelchair]

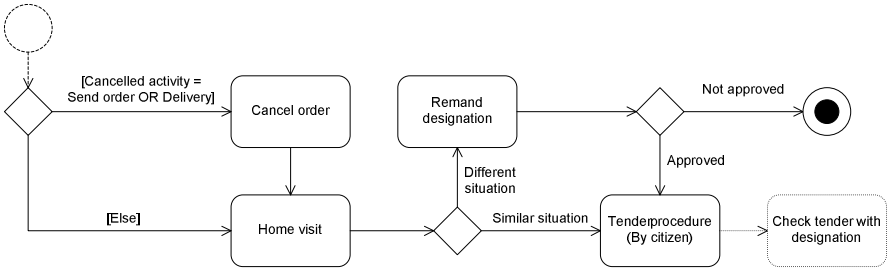

Fig. 9. Intervention pattern on [Major home modification]

6 Implementation

To show the feasibility of the approach, a prototype has been implemented on top of the Cordys Business Operations Platform[‡]. Cordys is a business process management platform, which also adheres to modern variability management techniques, such as case handling and process inheritance, thus providing run-time reconfiguration abilities. This

[‡] http://www.cordys.com

prototype adds dependency scopes over existing business process models, and maps each of the defined dependency scopes to an appropriate trigger.

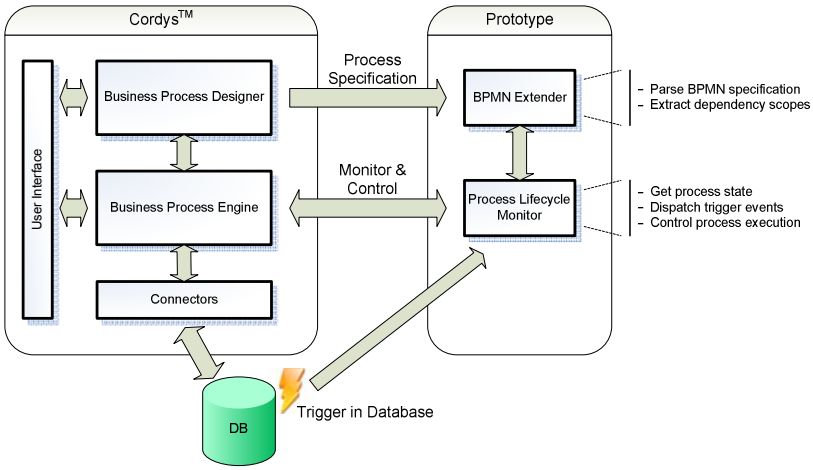

Fig. 10. Architectural overview of the prototype.

Fig. 10 depicts the architectural overview, where the left box represents the simplified Cordys architecture, while the right box represents the structure of the prototype itself. In the Cordys box, the major parts are Business Process Designer and Business Process Engine. The former provides visual process design facilities, whereas the latter provides the business process lifecycle support (i.e. process execution and monitoring). Connectors at the bottom of the Cordys box provide communication with external data, e.g. databases, or data provided by external services. Finally, the user interface provides means for interacting with users, and is usually represented by a web-based application.

In the prototype box, the BPMN Extender parses the BPMN process specification in order to extract dependency scopes. Such dependency scopes can be designed using standard Cordys process design facilities, and the information about dependency scopes is saved along with a process's BPMN specification in the internal process repository. After that, the process specification can be retrieved via the Cordys public API.

When the data modification occurs, a trigger is fired, which passes the corresponding information to the Process Lifecycle Monitor (PLM). This monitor has access to the information about existing business processes and their dependency scopes, which is provided by BPMN Extender. Based on the information about the processes being currently executed by the Cordys platform, PLM makes the decision whether or not to stop process(es) and fire appropriate intervention pattern(s). In order to support decision making, additional information must be associated with every dependency scope, such as

a table in the database, the criteria to find a row in the table, and the criteria to identify which changes in the data are significant.

In Fig. 11, the business process from Fig. 7 is modeled using Cordys process designer with BPMN Extender on top of it. Two overlapping dependency scopes ([Wheelchair] and [Major home modification]) are designed. Both dependency scopes are assigned to one variable {Address}. Both dependency scopes are associated with the table "Citizens" in the underlying database and, whenever the {Address} is changed, the corresponding intervention pattern is executed.

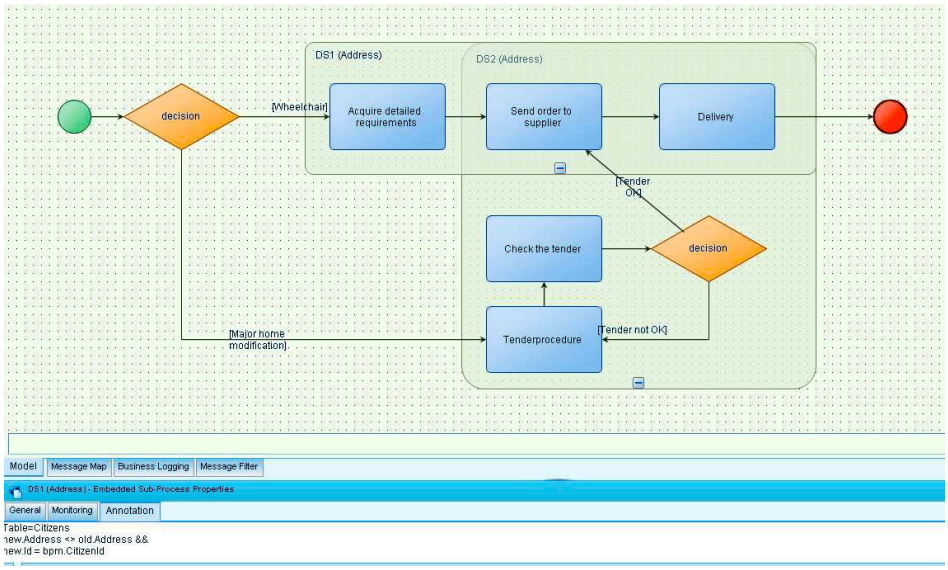

Fig. 11. Screenshot of dependency scope implementation within Cordys platform.

For example, imagine the situation in which some process instance went on the "Wheelchair" path, i.e., after the decision steps "Acquire detailed requirements" and "Send order to supplier" were executed, and the step "Delivery" is about to be executed. Now, suppose that the customer's address is changed due to the moving to another city. Since the process is now within DS1, the intervention actions must be undertaken.
The sequence of actions in this case is the following:

- The Process Lifecycle Monitor is called with the information that a row is modified in the "Citizens" table.
- The dependency scopes associated with the "Citizens" table (which are DS1 and DS2) are identified.

- The currently running process instances that are now inside one of those scopes (there is one running process instance, and activated dependency scope is DS1) are fetched.
- Check if data modification is significant for those process instances (since the address has been changed, then the modification is significant.

When all conditions are met, the intervention pattern is automatically executed as follows:
- The original process is stopped.
- The compensation process assigned to DS1 is executed, as shown in Fig. 8.

7 Conclusion

In this paper, intervention patterns are used for repairing inconsistencies in process execution as a result of process and data interference. The main advantage of intervention patterns is that continuity of the process can be ensured in a predefined way. We have shown that both dependency scopes and intervention patterns can easily be integrated within an existing BPMS platform. Furthermore, the problem of business process interference can be resolved using the concepts introduced in the paper, namely, dependency scopes and intervention patterns. A real case scenario (Dutch WMO law) is implemented and is planned to be integrated into the implementation of the law in one of the Dutch municipalities.

A situation might occur that does require intervention, but the predefined intervention patterns attached to the dependency scope do not apply for this particular situation. In these cases, two possible solutions can be suggested. First, the process can be paused and require a human decision on how to proceed. Second, a rollback can be executed. This is, however, the least desirable solution, especially in processes with a long lead time. Furthermore, it is important that intervention patterns are defined as generic as possible, in order to make sure they cover potential inconsistency cases that might occur within a dependency scope.

Currently, intervention patterns are built manually by a domain expert and specific to a certain dependency scope. More complex interacting processes require the number of intervention patterns as well. For future work, we plan to generalize the intervention concept, to allow for automatic DS and intervention pattern generation. This way, it can be ensured that all cases of interference are covered, potentially without the need for human intervention.

References

1. Aalst, van der , W.M.P., Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases, Vol. 14 (1), pp. 5-51 (2003).

2. Aalst, van der, W.M.P., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. Data and Knowledge Engineering, Vol. 53 (2), pp 129-162 (2005).
3. Aalst, van der, W.M.P., Hofstede, ter, A.H.M.: YAWL: Yet Another Workflow Language. Information Systems, Vol. 30 (4), pp. 245–275 (2005).
4. Aiello,M., Bulanov, P., Groefsema, H.: Requirements and Tools for Variability Management. In IEEE workshop on Requirements Engineering for Services at IEEE COMPSAC, to appear (2010)
5. OMG: Business Process Model and Notation (BPMN) 2.0, Request For Proposal. OMG Document: BMI/2007-06-05 (2007).
6. Dadam, P., Reichert, M.: The ADEPT project: a decade of research and development for robust and flexible process support. Computer Science - R&D, Vol. 23 (2), pp. 81-97 (2009).
7. Garcia-Molina, H., Salem, K.: Sagas. In Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pp 249 – 259 (1987).
8. Klai, K., Tata, S., Desel, J.: Symbolic Abstraction and Deadlock-Freeness Verification of Inter-enterprise Processes. Proceedings of the 7th International Conference on Business Process Management, Lecture Notes in Computer Science Vol. 5701, pp 294-309 (2009).
9. Li, Q., Zhou, J., Peng, Q.R., Li, C.Q., Wang, C., Wu, J., Shao, B.E.: Business processes oriented heterogeneous systems integration platform for networked enterprises. Computers In Industry, Vol. 61 (2), pp 127-144 (2010).
10. Pesic, M., Schonenberg, M. H., Sidorova, N., Aalst, van der, W.M.P.: Constraint-Based Workflow Models: Change Made Easy. OTM Conferences (1), pp. 77-94 (2007).
11. Rinderle, S., Reichert, M., Dadam, P.: Correctness Criteria for Dynamic Changes in Workflow Systems – A Survey. Data and Knowledge Engineering 50 (1) pp. 9–34 (2004).
12. Sadiq S., Sadiq W., Orlowska M.: Pockets of Flexibility in Workflow Specifications., in proc. ER'01, pp. 513–526 (2001).
13. Schonenberg, M.H., Mans, R., Russell, N., Mulyar, N., Aalst, van der, W.M.P.: Process Flexibility: A Survey of Contemporary Approaches. LNBIP, CIAO! / EOMAS, Springer Vol. 10, pp 16-13 (2008).
14. Trčka, N., Aalst, van der, W.M.P., and Sidorova, N.: Data-Flow Anti-Patterns: Discovering Data-Flow Errors in Workflows. In Proc. of CAiSE'09, LNCS 5565, Springer (2009).
15. Van Beest, N.R.T.P., Szirbik, N.B., Wortmann, J.C.: A Vision for Agile Model-driven Enterprise Information Systems. In Proc. of the 11th Int. Conf. on EIS, vol. "Inf. Syst. Analysis and Specification", pp. 188-94 (2009).
16. Van Beest, N.R.T.P., Szirbik, N.B., Wortmann, J.C.: Assessing The Interference In Concurrent Business Processes. In Proc. of the 12th Int. Conf. on EIS, vol. "Inf. Syst. Analysis and Specification" (2010).
17. WfMC: The Workflow Management Coalition Specification, Terminology & Glossary. Document Number WFMC-TC-1011 (1999).
18. Xiao, Y., Urban, S.D.: Using Data Dependencies to Support the Recovery of Concurrent Processes in a Service Composition Environment. In Proc. of Coop. Inf. Syst., Monterrey, Mexico (2008).