MMC-BPM: A Domain-Specific Language for Business Processes Analysis

Oscar González[1,2,⋆], Rubby Casallas[1], and Dirk Deridder[2,⋆⋆]

[1] Universidad de los Andes, TICSw Group, Cra. 1 N○ 18A 10, Bogotá, Colombia
{o-gonza1,rcasalla}@uniandes.edu.co
[2] Vrije Universiteit Brussel, SSEL, Pleinlaan 2, 1050 Brussel, Belgium
dirk.deridder@vub.ac.be

Abstract. Business Process Management approaches incorporate an analysis phase as an essential activity to improve business processes. Although business processes are defined at a high-level of abstraction, the actual analysis concerns are specified at the workflow implementation level resulting in a technology-dependent solution, increasing the complexity to evolve them. In this paper we present a language for high-level monitoring, measurement data collection, and control of business processes and an approach to translate these specifications into executable implementations. The approach we present offers process analysts the opportunity to evolve analysis concerns independently of the process implementation.

Keywords: Business Process Analysis, Domain Specific Language, Monitoring, Measurement, Control.

1 Introduction

Nowadays business process management (BPM) technologies are frequently used by companies to model, automate, execute and analyze executable business processes [1]. Business processes facilitate the integration of human and technological resources in an organization, according to a set of activities that fulfill certain policy goals. BPM has evolved from traditional workflow management systems (WFMS) including a Business Process Analysis (BPA) phase in its lifecycle. BPA is crucial to recover detailed feedback about how business processes are being executed to facilitate their effective improvement.

BPM evidences the difficulty of analyzing the execution of processes from a business perspective [2]. Firstly, while the process models are automated from a business perspective in a workflow implementation, the analysis specifications are realized from an implementation perspective. Although there are many tools and techniques to analyze business processes (*e.g.,* business activity monitoring) [3] [4],

⋆ Funded by the Flemish Interuniversity Council (VLIR) and COLCIENCIAS "Instituto Colombiano para el Desarrollo de la Ciencia y la Tecnología".
⋆⋆ Funded by the Interuniversity Attraction Poles Programme - Belgian State Belgian Science Policy.

W. Abramowicz (Ed.): BIS 2009, LNBIP 21, pp. 157–168, 2009.

typically these solutions use their own languages to encode analysis concerns in the implementation of the workflow system (*e.g.,* BPEL or java). Secondly, typically these analysis specifications result in tangled and scattered knowledge in the process code. These low-level mechanisms decrease the maintainability and reusability capabilities and increase the complexity because analysis concerns have to be repeated or adapted every time the process change. On the other hand, current BPA solutions typically only perform analysis in terms of process execution information (*e.g.,* time running, current state) but not in terms of the inner definition of basic activities (data). However, it is necessary to be aware immediately when a critical attribute value is assigned in order to take complementary decisions (*e.g.,* allow data modification) (Section 2).

In this paper we present our approach to realize analysis concerns from a business perspective and to automatically add them to an existing process implementation. Our work focuses on supporting the explicit description of the monitoring of business processes, the related measurement data collection, and the definition of control actions over this data. We realize this in such a way that our approach is independent of a particular process implementation language. This is done by working directly on BPMN, which is a high-level standard notation that can be used by process analysts to generalize the analysis specifications independent of specific implementation details. We define a domain-specific language to specify analysis concerns in terms of domain concepts (*i.e.,* BPMN) and a mechanism to express the data used by the process model to improve analysis capabilities. (Section 3). We provide the execution semantics of our language with a suitable aspect-oriented workflow language (Section 4).

The processes analyzed *a posteriori* and other kind of analysis problems such as verification or validation of processes, automatic improvements to the process model, and predictions are out of the scope of this work (Section 5). We conclude by discussing promising pathways for improving our work (Section 6).

2 Business Process Analysis Scenario

BPM supports processes modeling and derivation of executable definitions using multiple languages. We adopted the Business Process Modeling Notation (BPMN) [6] for the business perspective and the Business Process Execution Language (BPEL) [7] for the technical perspective since they are currently the de-facto standard in these areas. BPMN can be interpreted as a domain specific language (DSL) to describe business processes at a conceptual level. It bridges the communication gap that frequently occurs between the design of a business process and its corresponding implementation. However, BPMN suffers from a lack of support to specify data-flow and the implementation of activities required to automate processes [8]. BPMN models can be translated into an executable model such as BPEL, in which process instances are enacted by workflow engines.

This section introduces a Loan Approval Process as a running example [7]. In addition, we present a concrete scenario describing a number of analysis requirements, and the problems that users face to realize such requirements.

2.1 Case Study: A Loan Approval Process

The Loan Approval process starts with a loan request, where the client uses the bank′s webpage to introduce personal information and the amount requested. After this, the process executes three activities to decide if the request is approved or rejected. The first activity in the series is a task that invokes a Web service, which evaluates the information provided by the client. This activity is based on a set of business rules defined by the bank. For example, if the client income is less than 10% of the requested amount then the loan request is rejected; if the amount is less than $5000 then the loan request is approved. The second activity, which is executed in parallel with the previous one, is a subprocess that validates the loan viability by evaluating the risk associated with the requester. This subprocess seeks advice from a credit entity and validates the authenticity of the data provided by the client. The third activity merges the outputs of the previous two activities and consults a loan expert who defines the approval decision, which is stored in a variable named *requestState*. The process is depicted in Figure 1 using BPMN notation. This example introduces process elements such as a split, a merge, a subprocess, a set of activities, and two decision gateways.

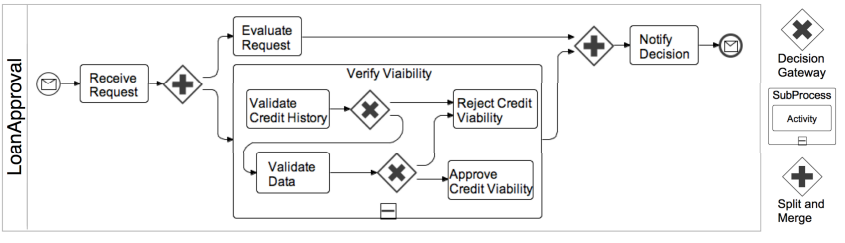


Fig. 1. Loan Request and Approval Process

This BPMN model can be translated into an executable process implementation (*e.g.,* BPEL), thus the process can be automated [11].

2.2 Business Process Analysis Requirements

Once the loan process is automated, the process analysts require feedback during process execution to enable its improvement. For example, based on feedback they could maximize the number of approved loan requests by adapting the business process constraints used by the *EvaluateRequest* activity. In order to support analysis activities, the following requirements are applicable:

1. *Monitoring events during process execution.* It is for example necessary to obtain the state of the loan request variable (*requestState*) for each loan process instance when the request is approved or rejected. This behavior occurs when the loan request variable is updated in the *NotifyDecision* activity.

2. *Creating new measurement data associated with process execution concepts and domain-specific concepts.* Measurement data is constructed by using process execution data as well as existing measurement data. For example, the process analysts require constructing new domain-specific metrics such as accepted/rejected requests, total requests, accepted/rejected requests ratio.
3. *Defining control rules using measurement data to understand process execution.* For example, the process analysts define a control rule to compute the defined measurement data and to verify when the rejected requests rate is higher than 50%, which means that the causes for rejection need to be verified (*e.g.,* the requested amount is too high). As soon as this behavior is present in the process, a control action can be taken. For example by notifying this situation to the quality assurance manager to allow for manually taking the corrective actions to improve the process (*e.g.,* adapt a business to define that the requested amount to automatically approve a request is $4500, instead of $5000) .

3 The MMC-BPM Language

MMC-BPM is a Domain Specific Language to complement business process models with monitoring, measurement and control (MMC) concerns. The language offers a declarative specification in which analysts describe what the analysis concern does instead of describing how it is implemented.

An MMC model defines the analysis concern associated with a target process from a specific observation point defined by the analysts. For example, an administrative stakeholder can be interested in analyzing the process using metrics such as rejected loan rates, while a quality control stakeholder could be interested in analyzing the process using time-related metrics such as average processing time. An MMC specification contains three main blocks to group the language constructs: the *Data* block, the *Event* block, and the *MMCRule* block. The code in Figure 2 illustrates how to specify the requirements presented in the case study using our MMC-BPM language. This specification corresponds to a specific observation point of a loan expert (line 1). We refer to this analysis specification throughout the section to illustrate our language elements.

3.1 Data Block

A process analyst uses the data block to define a) the data manipulated in the process and their types, and b) new measurement data required to analyze the process (Figure 3).

The MMC-BPM language provides a set of constructs to describe the data associated with the process model and with the elements in it (*e.g.,* Activities). This data model facilitates the use of process information within the analysis specification. For example, an analyst can describe that the *requestState* variable is associated to the *NotifyDecision* activity. Note that the data model is specified externally to the MMC model. As a consequence it can be reused in

```
1  MMCspec QualityView process LoanApproval
2   //Data
3   import dataTypes LoanProcess.xsd;
4   include data LoanProcess.pdata;
5
6   int AR; int RR; int TR; double RRA; double RRR;
7   //Events
8   event requestStateChange parameters boolean request;
9
10  onChange data.NotifyDecision.requestState
11   generates requestStateChange using data.NotifyDecision.requestState;
12  //Rules
13  mmcrule UpdateRequestState onEvent requestStateChange do
14   TR = TR + 1;
15   if !(event.request) then
16    RR = RR + 1; RRR = RR / TR;
17   else
18    AR = AR + 1; RRA = AR / TR;
19   endif
20   if RRR*100 > 50 then
21    notify 'quality@bank.com' 'RRR is too high' 'review the log.';
22   endif
23  endRule
24 endMMC
```

Fig. 2. Monitoring, Measurement and Control Specification for the Loan Process

```
Data            ::= ProcessDataTypes ProcessData (MeasureConcept)*;
ProcessDataTypes::= "import" "dataTypes" dataTypesFile=ID ".xsd" ";";
ProcessData     ::= "include" "data" dataFile=ID ".pdata" ";" ;
MeasureConcept  ::= type=DataType measureName=ID ";";
DataType        ::= (SimpleType | ComplexType | CollectionType);
SimpleType      ::= ("string" | "boolean" | "int" | "float" | "double" );
ComplexType     ::= complexDataType=ID;
CollectionType  ::= "<" (simpleType=SimpleType | complexType=ComplexType )">";
```

Fig. 3. EBNF specification of the Data Block

multiple analysis specifications. The MMC-BPM language references the data model using the `include data <dataFile>` clause (Line 4 in Figure 2).

In addition to the data model there is also a data type model, which determines the data types associated with the process data. We assume that the process data type model is defined using an XML Schema language. For example, the schema representing process data types can contain a complex data definition named *Client* containing a sequence of elements such as name, identification, and gender with their associated primitive data types. The data type model is referenced by using the `import dataTypes <dataTypesFile>` clause (Line 3 in Figure 2).

Measurement data defined in this specification is general to all the process instances. A measurement concept (*MeasureConcept*) is defined describing the name of the measurement data (*measureName*) and its data type (*DataType*). The MMC-BPM language provides a set of primitive data types (*SimpleType*) such as: string, int, float, boolean and double. For example, line 6 in Figure 2 illustrates the definition of the *accepted request (AR)* measurement concept using an *int* data type. Since business processes contain more complex data structures (*e.g.,* Loan, Client), the MMC language facilitates associating complex data types (*ComplexType*) and collections (*CollectionType*) to the measurement data.

A complex data type is related to one of the structures defined in the process data types model. Our DSL supports the definition of generic metrics (*e.g.,* duration, count, and resource) that are useful for any kind of process analysis and domain-specific metrics (*e.g.,* total number of rejected loans) that are only useful in a particular business domain.

3.2 Event Block

The Event block represents elements of the process domain used in the definition of analysis rules (for the purpose of monitoring) and the specific points where these rules are connected to the process model (Figure 4). A *ProcessEvent* is used to define the moment when a rule must be triggered as a result of an expected behavior (*e.g.,* an activity started) in the process execution. The MMC-BPM language allows grouping multiple process events in a concept named *LogicEvent*. This defines the information required to execute a rule.

Logic events detach rules from events, which fosters reuse. Thus, multiple process events can be associated with the same rule. For example, a process analyst can define two process events to retrieve the start and end time of all process activities. Thus, creating a rule for each process event, the code for computing the total processing time will be scattered in the analysis specification. The logic event makes it possible to modularize the rule code, which avoids putting crosscutting code in the analysis specification. This feature of the language separates what users want to do from how to do it.

```
Event                 ::= (logicEvent=LogicEvent)+ (processEvent=ProcessEvent)+;
LogicEvent            ::= "event" eventName=ID "parameters" ParameterSet ";";
ParameterSet          ::= parameter=Parameter("," parameter=Parameter)*;
Parameter             ::= type=DataType parameterName=ID;
ProcessEvent          ::= executionEvent = (ControlFlowEvent|DataFlowEvent)
                          "generates" eventName=ID "using" ParameterValueSet ";";
ControlFlowEvent      ::= executionEvent=ControlExecutionEvent BPMNModelIdentifier;
ControlExecutionEvent::= ("onStart" | "onFinish")
BPMNModelIdentifier   ::= "process"."bpmnElementName=ID;
DataFlowEvent         ::= executionEvent=DataExecutionEvent DataModelIdentifier;
DataExecutionEvent    ::= ("onChange" | "onMessage" | "onAlarm");
DataModelIdentifier   ::= "data"["."bpmnElementName=ID]"."attributeName=ID;
ParameterValueSet     ::= ParameterInvocation(","ParameterInvocation)*;
ParameterInvocation   ::= (DataModelIdentifier | MonitorInvocation | Literal |
                           SystemInvocation | EventInvocation);
MonitorInvocation     ::= "MonitorModel"."measureName=ID [InvocationType
                           parameterValueSet=ParameterValueSet];
InvocationType        ::= (invocationType="setValue"| invocationType="getValue");
SystemInvocation      ::= (service="TimeSystem");
```

Fig. 4. EBNF specification of the Event Block

Event blocks must contain at least one logic event and at least one process event. A logic event is defined using the `event <eventName> parameters <ParameterSet>` clause. The *eventName* identifies the event that will be used by a rule. The *ParameterSet* holds the information the logic event requires. Each *Parameter* contains a name and its corresponding *DataType*. Line 8 in Figure 2 illustrates a logic event named requestStateChange, which has a boolean parameter called request.

A process event describes a specific point in the process execution where it is necessary to a) capture process information, and b) generate a logic event assigning the retrieved information to the logic event parameters. This is done in the MMC-BPM language using the `<executionEvent> generates <eventName> using <ParameterValueSet>` clause. Figure 2 (lines 10-11) illustrates a process event that generates the logic event mentioned above when the variable *requestState* (associated with the activity *NotifyDecision*) is updated. The parameter of the logic event takes the value of this variable.

a) Capture Process Information. The MMC-BPM language offers the concept of an *executionEvent* to describe a relevant process execution behavior that we want to be aware of (*i.e.,* onStart, onFinish, onChange, onMessage, onAlarm). These execution events are defined in terms of process domain concepts without any specific implementation dependency. Our MMC-BPM language uses a subset of the vocabulary provided by the process domain model (*BPMNModelIdentifier*) or the process data model (*DataModelIdentifier*) to specify elements that the execution events monitor (*i.e.,* Activity, Data, IntermediateMessage). Process elements can be referenced using the `process.<bpmnElementName>` clause, where the *bpmnElementName* element can reference a unique process element (*e.g.,* NotifyDecision). Process data elements can be referenced using the `data.[<bpmnElementName>].<attributeName>` clause, where the *attributeName* references an attribute associated with a process element. Although we describe only some of the possible process elements and execution events available, the monitoring specification can be easily extended.

b) Generate a Logic Event. Once the process information is captured, a process event indicates which logic event must be generated (*eventName*). The parameter values (*ParameterValueSet*) of the logic event are assigned indicating a set of parameter invocations (*ParameterInvocation*). These parameter invocations can assign a value directly (*e.g.,* number, string), retrieve information from the process data model (*DataModelIdentifier*) or the monitoring data model (*MonitorInvocation*), or capture information from a system service (*SystemInvocation*). A monitor invocation facilitates accessing measurement data stored in the MMC data model. This is done using the `MonitorModel.<measureName>` clause with an optional set of parameter invocations (*ParameterValueSet*).

3.3 Rule Block

The rule block enables analysts to define actions to measure the process and to control its execution.

In our language, the `mmcrule <ruleName> onEvent <eventName> do` clause allows specifying the name of the rule and the logic event that will trigger the rule. The body of a rule can contain a) a set of actions, and b) a set of condition-action statements (Figure 5). Figure 2 (lines 13-23) illustrates the rule code for the case study.

```
MMCRule          ::= "mmcrule" ruleName=ID "onEvent" eventName=ID "do"
                       actionSet=ActionSet (conditionActions=ConditionAction)*
                     "endRule";
ActionSet        ::= (actionStatement=ActionStatement)*;
ActionStatement  ::= Assignment | ManagementAction;
Assignment       ::= measureName=ID "=" domainExpression=DomainExpression ";";
DomainExpression::= invocation+=Invocation(Oper invocation+=Invocation)*;
Invocation       ::= (ParameterInvocation | EventInvocation);
EventInvocation  ::= "event"'.' parameterName=ID;
ManagementAction::= NotifyAction | StoreAction | TraceAction ;
NotifyAction     ::= "notify" destinatary=STRING subject=STRING content=STRING";";
StoreAction      ::= "store" monitorInvocation=MonitorInvocation ";";
TraceAction      ::= "trace" log=STRING path=STRING ";";
ConditionAction  ::= "if" ifStatement=ConditionSet "then" thenStatement=ActionSet
                       ["else" thenStatement=ActionSet] "endif";
```

Fig. 5. EBNF Specification of the Rule Block

a) Actions. The *ActionSet* denotes a combination of action statements that can invoke an assignment (*Assignment*) to compute measurement data or take a management action (*ManagementAction*) to improve the process.

The measurement data is built using the `<measureName> = <Domain Expression>` clause. The *measureName* corresponds to a measurement data previously defined in the data block. The *DomainExpression* represents the arithmetical relation between a *ParameterInvocation* set or an *EventInvocation* set. The `event.<parameterName>` clause defines an event invocation. The event extracts the information required by the rule from a logic event definition. This measurement data facilitates process analysis in the particular business domain since these specifications are data-centric. The process execution data and measurement data are made persistent in the MMC data model for further analysis. Figure 2 illustrates the assignments to the measurement data using metrics stored in the MMC data model and other domain-specific metrics previously defined.

Process analysts can define management actions such as 1) storing monitoring concepts in the MMC model for visualization or query purposes (*store*), 2) the creation of a Log with the information retrieved and constructed in the application of the rule (*trace*), and 3) the notification of certain events to other external entities as for example the notification via mail (*notify*). Line 21 in Figure 2 illustrates a management action code for the case study, which notifies the quality assurance area when the rejected requests rate is higher than 50%.

b) Condition-Action statements. Process analysts can indicate the special behavior and the corresponding action to be taken in the Rule block to analyze the process. This is done with Condition-Action statements.

A control specification describes the *ConditionSet* that must be satisfied to execute some *ActionSet*. A set of conditions denotes a logic combination of expressions (*DomainExpression*). These represent an arithmetical relation between *ParameterInvocation* set or an *EventInvocation* set. The set of expressions involve a boolean expression, which triggers the set of management actions accordingly.

The language supports process analysis using evaluation statements over 1) the measurement data, 2) the process execution information, and 3) the process data. For example, Figure 2 (lines 15, 17, 20) illustrates the evaluation statements

to verify the state of the loan request and the rejected requests rate percentage to apply some actions. Other examples, for analyzing the process, involve the classification of generic Service Level Agreement (SLA) metrics [9]. A business SLA can state that loan requests with an amount over a certain threshold must be rejected, and an IT SLA can state that 95% of the loan request must complete within 72 hours.

4 Integrating MMC Concerns with the Business Process

This section briefly presents our approach to transform the MMC model into an executable implementation. We transform the MMC model into (BPEL) aspects, which are later merged with the BPEL base process [10]. We base our approach on Model-Driven Engineering (MDE) technology [12].

4.1 Padus: An Aspect-Oriented Workflow Language

We use Padus, an aspect-oriented extension to BPEL, which allows introducing crosscutting behaviour to an existing BPEL process in a modularized way [13].

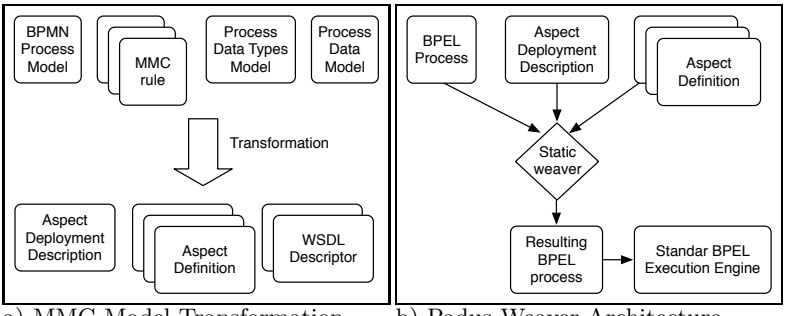
The Padus language facilitates the definition of specific points during the process execution where additional behaviour has to be added. These points can be selected using a logic pointcut language, and the Padus weaver can be used to combine the behaviour of the core process with the specified behaviour. The new behaviour can be introduced by inserting it *before* or *after* certain joinpoints defined by the pointcut, or it can replace existing behaviour by using an *around* advice. The Padus language introduces the concept of *in* advice to add new behaviour to existing process elements and provides an explicit deployment construct to specify aspect instantiation in specific processes. The advice code contains the extra behaviour that should be inserted, which is specified using standard BPEL elements.

4.2 Encapsulating Executable MMC Rules

Figure 6a illustrates the transformation process from the high-level analysis specification to the executable implementation.

The input for the transformation is a) the analysis specification defining the MMC rules, b) the BPMN process model representing the domain elements that we want to refer in our language, c) the XML schema describing the data types model associated with the process, and d) the process data model that describes the data attributes associated with each process element.

The output of the transformation is a) the Padus aspects that represent the MMC crosscutting concerns, b) the aspect deployment specification indicating how the aspects are instantiated and composed to the base process, c) the weaver java class that composes the base process and the aspects, d) the web services for supporting management actions, accessing the monitoring data model, or accessing to system services, and e) the XML schema that describes the measurement data defined in the language.



a) MMC Model Transformation b) Padus Weaver Architecture

Fig. 6. Transformation Process for the High-level Analysis Specification

Table 1. Analysis Specification into an Aspect Implementation

MMC-BPM language construct	*Output : Padus Implementation*
MMC model	- Aspect defining the analysis specification - Reference to the process model (BPEL code)
Data	Reference to data definitions (BPEL code)
LogicEvent	Joinpoint
ProcessEvent	Pointcut
onStart	Before advice
onFinish	After advice
onChange, onMessage	In advice
MMCRule	Advice (BPEL code)

Table 1 details the relevant mappings from our analysis language to the aspect-oriented programming implementation elements.

Figure 6b illustrates the Padus weaver architecture. The resulting artefact, after applying the aspect deployment description, is a BPEL process that can be deployed on a BPEL execution engine. We created an Eclipse plugin that serves as an editor to define the MMC rules and to execute the transformation process. The proposed MMC-BPM language and the model transformations rely on the OpenArchitectureWare [14] environment as metamodeling framework.

5 Related Work

Several commercial BPM products offer solutions for business activity monitoring (BAM) [15] [3] [4]. Typically, BAM solutions extract the information from audit trails, in which process metrics are added to the process architectures for analysis. In contrast, we propose a high-level language to describe, from a business perspective and independently of specific implementation languages, the process activities we want to monitor, the process and domain-specific metrics we want to build, and rules we want to apply over this information.

Other approaches use data mining and data warehouse techniques to capture the process execution information and to discover structural patterns for identifying the characteristics (explanations) that contribute to determine the value of a metric [16]. Our DSL allows end users to define evaluation statements making explicit the behavior that they are interested in and providing an implicit explanation of such situation. Moreover, our DSL allows defining the relations between monitoring and process data to built new domain-specific measures.

Other works [2] [5] [16] [17] introduce the idea to perform business process management using taxonomies and ontologies to capture semantic aspects. The authors of these works, as we do, consider analysis capabilities at the knowledge level, the definition of domain-specific metrics, and subsumption relations between concepts. The main difference is that our approach is focused on the analysis of business process activities instead of using process mining techniques. Process Mining aims at automatically discovering analysis information about processes based on event logs. Our approach facilitates to perform a similar performance analysis (measures) to the ones using a mining approach. Our approach can also be used to support the process mining approaches specializing the management action *trace* to provide the facilities necessary for logging the process and monitoring trail data in a structured way.

6 Conclusions and Future Work

Our approach helps in making the measurement variables and the associated rules more transparent to the end user by explicitly modeling them with a domain specific language and within an integrated process data model. In contrast to other approaches, domain-related metrics are easier to support because of the data model that we use to describe data associated with the process elements and the capabilities to define new measurement data. This high-level of abstraction facilitates the transformation of the analysis specifications into multiple workflow implementations. Our aspect-oriented approach in the definition and implementation of the analysis concerns offers process analysts the opportunity to evolve them independently of the process implementation.

Using our MMC-BPM language increases the level of abstraction of the business process analysis specifications at a conceptual level. In the future we plan to advance our work in several areas. Firstly, we require a way to assess how the MMC rules interfere with the process being executed. Secondly, it is necessary to find a mechanism to instrument the process for supporting the analysis specification done in terms of process data events. Third, we require defining a mechanisms to tackle or at least manage co-evolution issues between of MMC rules and the BPMN process model they refer to. Finally, as we consider that multiple users can analyze the same process, we should investigate ways to detect and resolve possible inconsistencies and conflicts that may arise.

References

1. van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M.: Business process management: A survey. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 1–12. Springer, Heidelberg (2003)
2. Hepp, M., Leymann, F., Domingue, J., Wahler, A., Fensel, D.: Semantic business process management: A vision towards using semantic web services for business process management. In: Lau, F.C.M., Lei, H., Meng, X., Wang, M. (eds.) ICEBE, pp. 535–540. IEEE Computer Society, Los Alamitos (2005)
3. Miers, D., Harmon, P., Hall, C.: The 2007 bpm suites report – a detailed analysis of bpm suites version 2.1, Business Process Trends (2007), `http://www.bptrends.com/`
4. Lau, C., Peddle, S., Yang, S.: Gathering monitoring metrics to analyze your business process, IBM (December 2007), `http://www.ibm.com/us/`
5. de Medeiros, A.K.A., Pedrinaci, C., van der Aalst, W.M.P., Domingue, J., Song, M., Rozinat, A., Norton, B., Cabral, L.: An outlook on semantic business process mining and monitoring. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2007, Part II. LNCS, vol. 4806, pp. 1244–1255. Springer, Heidelberg (2007)
6. OMG: Business process modeling notation specification - final adopted specification (February 2006), `http://www.bpmn.org`
7. IBM: Business process execution language for web services (July 2002), `http://www.ibm.com/developerworks/library/specification/ws-bpel/`
8. Dubray, J.J.: The seven fallacies of business process execution, InfoQ (December 2004), `http://www.infoq.com/`
9. Castellanos, M., Casati, F., Dayal, U., Shan, M.C.: Intelligent management of slas for composite web services. In: Bianchi-Berthouze, N. (ed.) DNIS 2003. LNCS, vol. 2822, pp. 158–171. Springer, Heidelberg (2003)
10. González, O., Casallas, R., Deridder, D.: Modularizing monitoring rules in business processes models. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM Workshops. LNCS, vol. 5333, pp. 22–23. Springer, Heidelberg (2008)
11. Giner, P., Torres, V., Pelechano, V.: Bridging the gap between bpmn and ws-bpel. m2m transformations in practice. In: Koch, N., Vallecillo, A., Houben, G.J. (eds.) MDWE. CEUR Workshop Proceedings, vol. 261, CEUR-WS.org (2007)
12. Schmidt, D.C.: Guest editor's introduction: Model-driven engineering. IEEE Computer 39(2), 25–31 (2006)
13. Braem, M., Verlaenen, K., Joncheere, N., Vanderperren, W., Straeten, R.V.D., Truyen, E., Joosen, W., Jonckers, V.: Isolating process-level concerns using padus. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 113–128. Springer, Heidelberg (2006)
14. OpenArchitectureWare (oAW) website, `http://www.openarchitectureware.org`
15. von den Driesch, M., Blickle, T.: Operational, Tool-Supported Corporate Performance Management with the ARIS Process Performance Manager. Aris in practice edn. (2006)
16. Grigori, D., Casati, F., Castellanos, M., Dayal, U., Sayal, M., Shan, M.C.: Business process intelligence. Computers in Industry 16(3), 321–343 (2004)
17. Pedrinaci, C., Domingue, J., Brelage, C., van Lessen, T., Karastoyanova, D., Leymann, F.: Semantic business process management: Scaling up the management of business processes. In: ICSC, pp. 546–553. IEEE Computer Society, Los Alamitos (2008)