An Ontology-Aided Process Constraint Modeling Framework for Workflow Systems

Shasha Liu, Manuel Correa, Krys J. Kochut
Department of Computer Science
The University of Georgia
Athens, GA, USA
{shasha, correa, kochut}@cs.uga.edu

***Abstract* – Specification of non-functional and domain-specific constraints in workflow processes and incorporating them within workflow applications have posed persistent problems for workflow designers. In order to address these problems, we propose a constraint handling framework consisting of a Process Constraint Ontology and a Process Constraint Language. The extensible ontology allows workflow designers to specify constraint knowledge and vocabulary specific to their domain of interest. Subsequently, process constraints are formulated in the constraint language by utilizing the constraint concepts from the ontology. The constraints are connected to the affected process elements (activities, data, and performers), deployed along with the process definition, and enforced and handled at runtime by the workflow enactment system. Based on the proposed framework, we have implemented a prototype of a constraint-enabled workflow management system and used it to incorporate and enforce geospatial constraints for an emergency management workflow process.**

***Keywords* – process modeling, constraints, non-functional requirements, ontology, process constraint language**

I. INTRODUCTION

As defined by the Workflow Management Coalition (WfMC), a workflow is "the computerized facilitation or automation of a business process, in whole or part" [1]. Over the past few decades, workflow systems have been successfully applied in numerous areas of industries, including banking, manufacturing and scientific research. A workflow is often specified by a process definition language. However, one common disadvantage of current process definition languages is the lack of the capability of describing additional process constraints and non-functional requirements (NFRs). For example, Business Process Model and Notation (BPMN) does not include support for NFRs [2]. Nevertheless, functional and non-functional constraints are vital to process definitions of virtually all workflow applications during their design and development [3].

Constraints and NFRs have been a focus in workflow research since the introduction of workflow management systems due to their high impact on the overall success of workflow applications. Quality of service (QoS) is an important subset of NFRs [4]. Other process constraints may involve such factors as geographic or network locations and properties of system resources. For example, an emergency handling workflow process may require some of its tasks to be executed in close geographic proximity. Similarly, a scientific workflow may prohibit transfers and analysis of the generated experimental data by external workflows due to privacy or security concerns. In order to express such application-specific constraints and other NFRs with a process definition, workflow designers need an intuitive and clear method to specify the workflow constraints and NFRs, and these workflow constraints and NFRs should be enforced at runtime by the workflow engine to meet users' needs. Mapping these high-level requirement specifications to the low-level workflow execution remains a big challenge for the researchers and developers of workflow systems.

We summarize the high-level objectives of a constraint-enabled workflow system as follows: (i) constraint specifications should be expressed using a commonly agreed-upon vocabulary; (ii) constraint specifications should be reusable, extensible and intuitive to create; (iii) constraint specifications should be attached to any process elements, and their validation and enforcement should be supported by workflow runtime.

The main contribution of our work is twofold: (i) we have created a process constraint ontology, named ProContO, which enables process designers to express and share their knowledge of process NFRs and domain-specific constraints; (ii) we have developed a process constraint language, named PCL, which can be used to specify process constraints and NFRs in terms of process elements (such as BPMN tasks and data objects) and the constraint vocabulary defined in the ontology. Using our approach, a workflow process designer can define a workflow process and clearly specify a variety of constraints and NFRs that go beyond the expressiveness of typical process definition languages. An important aspect of PCL is that the constraint expressions can be deployed as part of the workflow application, and then evaluated and handled at runtime under a constraint-enabled workflow management system.

The rest of the paper is organized as follows. Section II presents three motivating workflow examples, while Section III contains a review of the related work. A process constraint ontology for constraints modeling is introduced in Section IV. Section V introduces process constraint language (PCL). Section VI proposes a general architecture for a constraint-enabled workflow enactment system and presents our prototype implementation, which is capable of handling geospatial constraints in workflow processes. Conclusions and future work are discussed in Section VII.

 ISBN: 978-1-61208-254-7

II. MOTIVATING EXAMPLES

In this section, three motivating examples are discussed to illustrate the importance of specifying constraints and non-functional requirements in workflow processes.

The first example is a simplified purchase approval process: a manager requests a new computer purchase, and this request is approved by another manager. In order to avoid fraudulent activities, managers who request and approve should be different. Although BPMN can specify an actor or a role, such a constraint cannot be specified.

GlycoQuant IDAWG™ workflow is used by scientists at the Complex Carbohydrate Research Center at the University of Georgia to perform quantitative glycomics analysis. One part of the workflow can be represented as four sequentially connected tasks shown in Fig. 1. The raw data are produced by the mass spectrometer experiment task. Transferring raw data directly over the open internet may not be feasible due to the large data size (e.g., in gigabytes) and security concerns. Instead, it is preferable to transfer the data pre-processing task to the computer storing the raw data, and then only transfer the segmented and encoded data back to more powerful servers for further computational analysis.

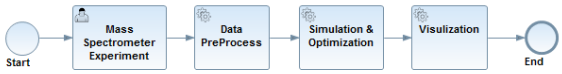

Figure 1: GlycoQuant IDAWG Workflow

The final example illustrates domain-specific geospatial constraints in an emergency management workflow that go beyond those simple constraints like task performers or the input/output data. Fig. 2 shows a fragement of the workflow process dealing with tornado emergencies. Once a tornado warning has been issued, schools within the tornado path need to be evacuated while shelters and hospitals that will not be affected but are near the area need to get prepared. It is apparent that geospatial constraints, such as the distance between a school and tornado path, are impacting the task execution.

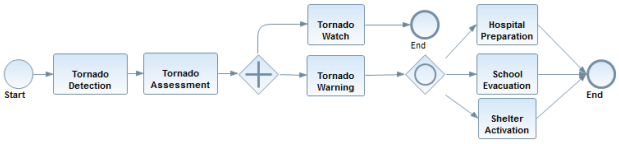

Figure 2: Tornado Emergency Workflow

III. RELATED WORK

The work on representing constraints and NFRs within the general software engineering models has received a lot of consideration. Formalized language and graphical representation have been applied to define the constraints. NoFun [5] is a formalized language aiming to facilitate quantitative analysis of NFRs. A framework consisting of two languages, the process-NFL and the product-NFL, was proposed in [6] for building non-functional software architecture both in software developing phase and for the final software products. In [7], two additional artifacts, named Operating Condition and Control Case, have been introduced to BPMN to better discover and represent NFRs at an early phase of the business development life-cycle. In [8], flow model is applied to connect conceptual activity diagrams in UML and technical activity diagrams in BPMN for the purpose of design process continuity.

In summary, the formalized language representations help developers to easily express and document constraints, while the graphical notations provide an intuitive way for eliciting and visualizing them. A given constraint or an NFR can be stated by different vocabulary, which may lead to imprecise and ambiguous specifications [5]. This is a difficult problem for the reported modeling approaches. In [9], the authors addressed this problem by embedding specific keywords in their modeling framework to control the concepts and vocabulary used by developers. However, it is next to impossible to reach a consensus on a good set of constraint concepts expressive enough to cover a wide variety of application domains.

Recently, a lot of work has been focused on building ontologies for QoS, NFRs and domain-specific requirements in Web services, business and scientific workflows to promote such consensus regarding the constraints concepts and relationship among them. In [10], Dobson, et al. developed an ontology to model non-functional aspects in service-centric systems, based on which, he and his colleagues later presented a domain-independent ontology for NFRs and illustrated its application in a business trip service [11]. DAML-QoS [12] is another example of using ontology to model QoS for web services. However, based on a rapidly developing interest in scientific workflows, there is an increasing need for a general-purpose constraint specification framework that can model both non-functional and other domain-specific requirements. Our work addresses these issues by introducing an extensible process constraint ontology and a process constraint language. We also propose a software framework suitable for the development and execution of workflows incorporating constraints and NFRs.

IV. PROCESS CONSTRAINT ONTOLOGY

Specification and handling of process constraints should be an integral part of a well-designed workflow application. Our motivating examples presented a few types of constraints that may be found in many other processes with similar types of requirements. We believe that ontologies offer the requisite expressive power to define the knowledge about process constraints, their classification and relationships, as well as suitable relationships connecting them to process components.

The high-level classes of our Process Constraint Ontology, (ProContO), are shown in Fig. 3. The *ProcessElement* class represents components in process definitions (activities, data objects, and performers). A *ProcessElement* may have a number of *Constraint-Attributes*, which represent simple constraint properties, such as the execution time of a task, its geospatial position, the size of an input data or a host's network location. However, many constraint attributes may have to be

 ISBN: 978-1-61208-254-7

computed by suitable operations (for example the distance between locations of two tasks). Such operations are represented by the *Constraint-Operation* class. A *ConstraintOperation* takes *Constraint-Attributes* as its parameters and produces a *Constraint-Attribute* as its output. The three classes and relationships among them serve as the backbone of our process constraint ontology, which will be explained in greater detail in the rest of this section. An important aspect of our approach is that the ontology is meant to be extensible and the three classes are regarded as the roots of their respective hierarchies.

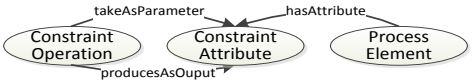

Figure 3: Backbone of the Process Constraint Ontology

A. Process Elements

Existing process definition languages, such as BPMN and XPDL, use different names for the components in a workflow process, but their functions and relationships are similar. Following the BPMN specification, ProContO includes the *Activity* class, used to represent a task within a workflow process, as shown Fig. 4. An *Activity* may input and output *Data*. Each *Activity* is executed by a processing entity, represented by the *Performer* class. *Process-Elements* can be described by attributes, such as the execution time of an *Activity* or the size of a *Data* object. Such properties can later be used in defining process constraints. For example, *SizeAttribute* and *Temporal-Attribute* are types of *ConstraintAttributes* in Fig. 4.

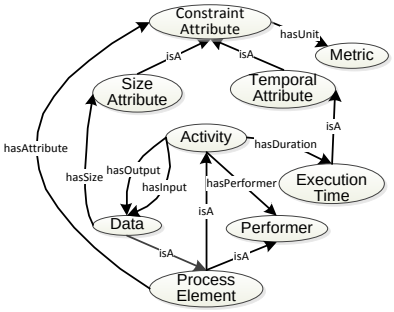

Figure 4: *ProcessElement* and *ProcessConstraint*

B. Constraint Attributes

The *ExecutionTime* is a subclass of the *Temporal-Attribute* and can be used to describe the properties of an *Activity*. Similarly, the *SizeAttribute* can be used to describe properties of *Data* elements. As shown in Fig. 4, *Activity* within the *ProcessElement* module is related to *ExecutionTime*, a subclass of *ConstraintAttribute*, by the object property *hasDuration*, and *Data* to *SizeAttribute* by *hasSize*. Both *hasDuration* and *hasSize* are defined as sub-properties of the *hasAttribute* relationship in the ontology (not depicted in the figure). Process designers may extend the ontology by adding additional *Constraint-Attributes* suitable for their application domain.

Another important part of the constraint ontology is the *Metric* module. It is meaningless to define a numerical constraint without giving its unit. For example, the *ExecutionTime* of an *Activity* may be specified in milliseconds or hours. Defining various units of measure is important but goes beyond the scope of this paper. An example of a metric ontology called QoSOnto in [10].

C. Constraint Operations

Some process constraint attributes cannot be expressed as simple properties attached to process elements and must be calculated via particular operations. The *Constraint-Operation* class is introduced to handle such constraints.

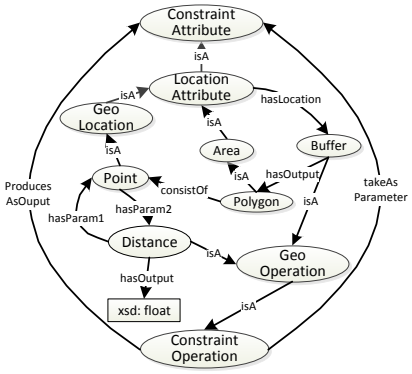

Figure 5: *ProcessConstraint* and *ConstraintOperation*

Considering the domain specific constraints in the Tornado Emergency Workflow depicted in Fig. 2 as an example, locations of hospitals and schools in the emergency system are viewed as geographic locations specified by their longitude and latitude. The distance between them can be calculated at runtime. The *Distance* operation, as shown in Fig. 5, takes two *Points* as parameters. The computed distance to the projected tornado path can later help to determine which nearby hospitals should be alerted to the tornado. For example, the hospitals closer than 30 miles to the tornado path should be prepared for evacuation, while those 30 to 50 miles away should be prepared to accept injured patients.

The *Buffer* operation is an example of an operation that produces a *ConstraintAttribute* as the output rather than a numeric value. In the context of the Tornado Emergency Workflow, the *Buffer* operation can be used to calculate the buffer area (a zone around a map feature), which is a *Polygon* area outlining the tornado path. As explained later, the operations defined here will have corresponding executable functions available for the runtime system.

Our process constraint ontology can facilitate the modeling of process constraints since it (i) serves as a concept vocabulary enabling process designers to use common language when specifying constraints, (ii) allows the specification and correctness validation of process constraints, (iii) can be easily extended by process designer to represent a variety of application domains.

V. PROCESS CONSTRAINT LANGUAGE

During process design of control and data flows, additional constraints are elicited and added as classes and instances in the constraint ontology. The next step involves

 ISBN: 978-1-61208-254-7

specifying constraint expressions and connecting them to suitable elements in the process definition. To enable this, we have created an ontology-aided process constraint language (PCL). It serves as a declarative specification language for formulating and documenting additional process requirements. PCL constraint expressions are attached to the designed process and ultimately deployed to the enactment service for execution. Syntax of PCL is similar to that of the Object Constraint Language (OCL). Although the constraints discussed in motivating examples are difficult to specify via current process definition languages (e.g., BPMN), they can be defined in PCL constraints as described in the following subsections.

A. Expressions

A PCL expression is a logical assertion of a constraint, which evaluates to a Boolean value (true or false**)**. Table I shows the outline of the syntax of PCL expressions (defined using the Extended Backus-Naur Form). A literal is the smallest expression in PCL, which can be a string, a number, or a name. For brevity, we don't precisely define strings and numbers. A name is an identifier referring to a concept in the constraint ontology or a name of an activity in the process definition. It is also used to identify a constraint. Larger expressions are formed with the use of unary and binary operators (Table II), which include arithmetic, logical operators and navigation operators. The two navigation operators are used to traverse relationships in the ontology. The difference between "." and "→" is that the "." operator navigates the ontology by class names on the other side of associations, while the "→"operator uses the name of a specific relationship.

TABLE I. PCL EXPRESSIONS

expression	::=	logical_expr
logical_expr	::=	relational_expr {logical_op relational_expr }
relational_expr	::=	arithmetic_expr [relational_op arithmetic_expr]
arithmetic_expr	::=	unary_expr {arithmetic_op unary_expr }
unary_expr	::=	[unary_op] navigation _expr
navigation_expr	::=	primary_expr [navigation_op name]
primary_expr	::=	"(" expression ")" | if_expr | constraint_call | literal
if_expr	::=	"**if**" expression "**then**" expression "**else**" expression "**end if**"
constraint_call	::=	name "(" [constraint_parameters] ")"
constraint_params	::=	expression {"," expression }
literal	::=	string | number | name | "**true**" | "**false**"

A constraint call is an invocation of a constraint operation defined in the constraint ontology (an instance of a class in the *ConstraintOperation* hierarchy). The name should always starts with a lower case letter. As an example, consider a call *buffer(tornado.geoLocation)*, where *Buffer* is the name of a *ConstraintOperation*, while *tornado* is an alias of the Tornado Assessment activity in the process shown in Fig. 2. The *LocationAttribute* of a tornado is accessed through the "." navigation operator. An alias of an activity can be declared in the context declaration of a process constraint, which will be explained in the next subsection. Due to space limitations, additional PCL elements, such as quantifiers to deal with constraints on sets of process elements or their attributes, are not discussed here.

TABLE II. PCL OPERATORS

Operator			Associativity
unary_op	::=	"**not**"	right-to-left
logical_op	::=	"**and**" | "**or**" | "**xor**"	left-to-right
relational_op	::=	"="|">"|"<"|">=" |"<=" | "<>"	
arithmetic_op	::=	"+" | "-" | "*" | "/"	
navigation_op	::=	"." | "→"	

B. Constraint Declarations

Connections between constraints and processes are specified in the context definition part, which is used to list the process activities involved in the constraint. As the name of an activity can be long, activity aliases can be introduced at the same time. A constraint is identified by its name and includes one or more conditions, which are either invariants, pre-, or post-conditions. The syntax of constraint definitions is shown in Table III.

TABLE III. PCL CONSTRAINT DECLARATION

constraint_declaration	::=	"**constraint**" name context_definition condition { condition }
context_definition	::=	"**context**" [alias ":"] name {"," [alias ":"] name}
condition	::=	constraint_type [name] expression {"," expression}
constraint_type	::=	"**inv**" | "**pre**" | "**post**"

An invariant condition (**inv**) must hold during a workflow instance execution. More specifically, it is checked *before* and *after* the execution of all activities listed in the **context** definition. In case not all of the constraint attributes used in the expression are available (have already been established) due to the relative ordering of activities determined by the process control flow definition, the assertion is considered true. Consider an example shown in Table IV. Before and after the execution of the PurchaseRequest activity, the constraint is true, as the Performer of PurchaseApproval is not available yet. The actual verification of such a constraint can only be performed once the activity of PurchaseApproval starts and its *Performer* has been determined.

Pre-conditions (**pre**) define the required status of process elements *before* they start to execute. If more than one activity is declared in the context definition, the constraint expression specified within the **pre** clause needs to be verified at the starting point of each activity instance. Again, the constraint is trivially asserted as true if some attributes are not available yet (task has not executed yet).

 ISBN: 978-1-61208-254-7

Examples of pre-condition definitions are shown in Table V. Similarly, post-conditions (**post**), are evaluated *after* the execution of each involved activity.

The name given to a constraint not only facilitates the documentation and serialization of constraints along with the process definition, but also makes it easier to connect constraints with suitable exception handling methods within the process. If a constraint fails during a process instance execution, an exception is thrown and made available to the workflow engine and handled in a proper way, according to the defined exception handler. PCL is a declarative language used to specify constraints, while the process definition language, such as BPMN, is the proper place to provide detailed logic for handling of failed constraints. This issue will be further discussed next.

TABLE IV. CONSTRAINT DEFINITION OF THE PURCHASING WORKFLOW

constraint	ApprovalPermission
context	t_1: PurchaseRequest, t_2: PurchaseApproval
inv	t_1→hasPerformer <> t_2→hasPerformer

TABLE V. CONSTRAINT DEFINITION OF GIS AND IDAWG™ WORKFLOW

constraint	GeoLocationProximity
context	tornado: TornadoDetection, shelter: ShelterActivation
pre	**not** withIn(shelter.geoLocation, buffer(tornado.geoLocation)) **and** distance(tornado.geoLocaion, shelter.geoLocation)) < 50 mile
constraint	DataTaskCoLocation
context	t_1: MassSpectrometerExperiment, t_2: DataPreProcess
pre	**if** t_1→hasOutput > 1GB **then** t_2.networkLocation = t_1→hasOutput.networkLocation **else true endif**

VI. PROTOTYPE IMPLEMENTATION

In this section, we introduce a general architecture for a constraint-enabled workflow system and describe our prototype implementation based on the jBPM 5 workflow management system. jBPM 5's process definition is based on BPMN 2.0 and our prototype implementation adds the constraint specification and handling to a BPMN process definition by linking the PCL constraints to BPMN process elements and enforcing them during execution.

A. System Architecture

As shown in Fig. 6, our architecture has three layers: (i) the User Interface Layer, (ii) the Process Constraint Engine Layer, and (iii) the Context Awareness Layer. The User Interface Layer manages the interactions between a process designer and the underlying process-constraint engine. It not only provides intuitive graphical notation to define processes, but also provides a way of uploading the related constraint ontologies, as well as defining constraints using PCL. The Context Monitor module in the Context Awareness Layer is responsible for runtime context information, such as the network status, workload balancing statistics of the system and the geospatial information of a newly formed tornado in an emergency reaction workflow during the execution. The information is further processed by the Context Handler and fed to the Constraint Validator.

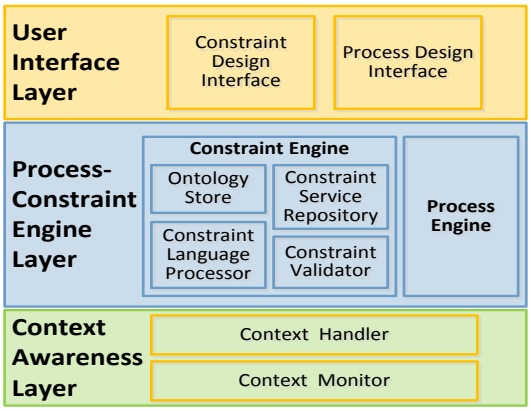

Figure 6: Prototype Architecture

The Process-Constraint Engine Layer is the core component of the constraint-enabled workflow system. It consists of the Constraint Engine and Process Engine.

Within the Constraint Engine module, there are four sub-components. The Ontology Store contains the created ProContO ontology, including process elements, constraint attributes and constraint operations, while the Constraint Service Repository serves as a registry to manage the services related to the constraint operations and provide references to the Constraint Language Processor and Constraint Validator about how to interact with these services. The Constraint Language Processor parses the constraints specified in PCL and validates the syntax and semantics using the constraint ontology. It translates the constraints into executable constraint objects stored in the Constraint Service Repository. The translation keeps track of the mapping between the constraint operations defined in the ontology and the actual implementation of such operations (e.g., the code for calculating the distance between two points). This mapping enables process designers to focus on the constraint design without worrying about the underlying implementation.

The Constraint Validator is responsible for validation of constraints against the context information. It determines whether the constraint is satisfied and provides the validation results to the process engine.

One part of the Process Engine's functionality is to accept a process definition from the user interface layer, deploy it within the engine and execute process instances. In addition, since it is constraint-enriched, the Process Engine also receives input from the Constraint Engine concerning the constraints validation, and adapts its behavior accordingly, if needed. To be more specific, if the Constraint Validator does not detect any failed constraints, the Process Engine continues the normal sequence flow defined in the process. However, if one of the constraints fails, the Constraint Validator throws an

 ISBN: 978-1-61208-254-7

exception corresponding to the name of the constraint. The Process Engine catches the exception and invokes the corresponding exception handler, if one has been defined in the process. If no suitable exception handler has been defined for the exception thrown by the failed constraint, a default action, such as suspend or terminate the execution of the current process instance is triggered.

B. A Prototype Implementation

Based on the general architecture, we have implemented a constraint-enabled workflow system prototype, focusing on enforcing geospatial constraints for emergency response processes [13]. It utilizes the existing jBPM 5 (from Redhat's jBoss) as the Process Engine. Domain specific constraint operations are implemented as Web services coded in Python. User defined geospatial constraints defined in PCL, are translated into JSON strings and mapped to the corresponding constraint operations. Every time a process instance is created by the process engine, the Constraint Validator generates a validator instance based on the associated constraint specification and evaluates them based the given runtime context information. The whole procedure is illustrated in Fig. 7. BPMN exception handling mechanism to is used to signal and handle runtime exceptions.

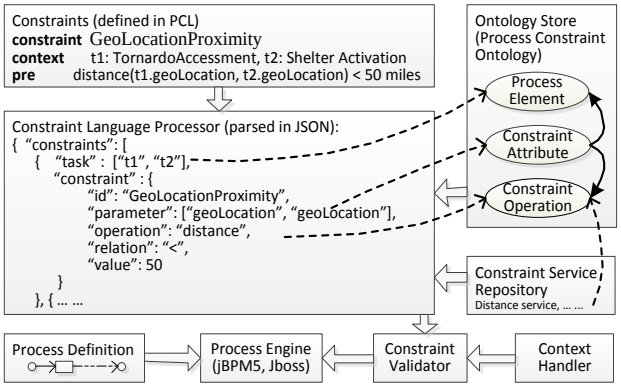

Figure 7: Procedure of constraint-enabled workflow system

VII. CONCLUSION AND FUTURE WORK

In this paper, we have addressed the problem of modeling and specifying domain-specific constraints and NFRs in workflow processes and incorporating them within workflow applications. We introduced a process constraint handling framework consisting of a process constraint language (PCL) and an extensible process constraint ontology (ProContO). ProContO allows workflow designers to specify constraint knowledge and vocabulary specific to their domain of interest, which we illustrated with examples. Process constraints are formulated in PCL, utilizing the constraint concepts defined in ProContO. PCL constraints are connected to process elements and deployed along with the process definition for execution. We have implemented a prototype of a constraint-enabled workflow management system and used it to create an emergency management workflow incorporating to handle geospatial constraints.

In the near future, we plan to integrate user interface of extending the ontology and creating PCL constraints with existing process definition tool. We also intend to integrate the framework with Web service composition methods.

ACKNOWLEDGMENT

This work was funded in part by the National Institutes of Health grant RR018502 for the Integrated Technology Resource for Biomedical Glycomics.

REFERENCES

1 WfMC: 'Terminology & Glossary' (Winchester, 1999, 3rd edn. 1999)

2 Gorton, S., and Reiff-Marganiec, S.: 'Towards a task-oriented, policy-driven business requirements specification for web services', Proc. The 4th International Conference on Business Process Management (BPM 2006), 2006, pp. 465-470

3 Chung, L., and do Prado Leite, J.: 'On Non-Functional Requirements in Software Engineering': 'Conceptual Modeling: Foundations and Applications' (Springer-Verlag, 2009), pp. 363-379

4 Cardoso, J., Sheth, A., Miller, J., Arnold, J., and Kochut, K.: 'Quality of service for workflows and web service processes', Web Semantics: Science, Services and Agents on the World Wide Web, 2004, 1, (3), pp. 281-308

5 Franch, X., and Botella, P.: 'Putting non-functional requirements into software architecture', Proc. The 9th International Workshop on Software Specification And Design, 1998, pp. 60-67

6 Rosa, N.S., Justo, G.R.R., and Cunha, P.R.F.: 'A framework for building non-functional software architectures', Proc. The 2001 ACM Symposium on Applied Computing, 2001, pp. 141-147

7 Pavlovski, C.J., and Zou, J.: 'Non-functional requirements in business process modeling', Proc. The 5th Asia-Pacific Conference on Conceptual Modelling (APCCM 2008), 2008, 79, pp. 103-112

8 Al-Fedaghi, S.: 'BPMN Requirements Specification as Narrative', Proc. 3rd International Conference on Information, Process, and Knowledge Management (eKNOW 2011), 2011, pp. 68-75

9 Cysneiros, L.M., and do Prado Leite, J.C.S.: 'Nonfunctional Requirements: From Elicitation to Conceptual Models', IEEE Trans. Softw. Eng., 2004, 30, (5), pp. 328-350

10 Dobson, G., Lock, R., and Sommerville, I.: 'QoSOnt: a QoS Ontology for Service-Centric Systems', Proc. The 31st EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA 2005), 2005, pp. 80-87

11 Dobson, G., Hall, S., and Kotonya, G.: 'A Domain-Independent Ontology for Non-Functional Requirements', Proc. The IEEE International Conference on e-Business Engineering (ICEBE 2007), 2007, pp. 563-566

12 Zhou, C., Chia, L.T., and Lee, B.S.: 'DAML-QoS Ontology for Web Services', Proc. The IEEE International Conference on Web Services (ICWS 2004), 2004, pp. 472-479

13 Correa, M.: 'Geospatial context awareness in business pocess modeling', The University of Georgia, 2012

 ISBN: 978-1-61208-254-7