

Represent Software Process Engineering Metamodel in Description Logic

Shengjun Wang, Longfei Jin, and Chengzhi Jin

Abstract—An approach of representing OMG's Software Process Engineering Metamodel (SPEM) in Description Logic (DL) is provided in this paper, and an example of using it to represent and reason on a concrete software process model -- a portion of DMR Macroscopic is presented, too. This approach is built on two mappings. One is a mapping from Meta-Object Facility (MOF) to DL and the other is a mapping from Object Constraint Language (OCL) to DL. Using these two mappings, OMG's SPEM consisted of a metamodel that is an MOF extension and constraints represented in OCL can be mapped to a representation in DL. Thus, software process models based on SPEM can be represented in DL, and process analysis and manipulation techniques, such as reasoning, consistency checks can benefit from DL techniques.

Keywords—Software Process Engineering Metamodel, Description Logics, Meta-Object Facility, Object Constraint Language

I. INTRODUCTION

THE development of software is a complex process consisting of many interdependent activities, including technical development, project management, quality assurance, and customer support activities. There have been many attempts to capture the necessary abstractions in order to make the development of software an easier task. All of these attempts, ranging from the waterfall model to the spiral model of development, have contained an attempt to capture the general process of software design. Precisely specifying the process by which a software engineering activity takes place is a challenging task that involves many factors. Striking a balance between expressiveness and succinctness is only one example of one of the many considerations that must be taken into account. The effort of specifying the exact model of a process, however, rewards developers with significant advantages. Once the model of a process is precisely defined in a formal manner, process analysis techniques can be applied to such a model to identify problematic and erroneous steps, or to leverage efficiency improvements. A common formal basis will enable the easy exchange of specific process models between interested parties.

OMG's Software Process Engineering Metamodel (SPEM) [1] is a standard metamodel that used to describe a concrete software development process or a family of related software

development processes. The SPEM specification version 1.1 published in January 2005 provides a complete Meta-Object Facility (MOF) [2] based metamodel, which facilitates exchange with both UML [3] tools and MOF-based tools/repositories. However, the specification is compliant to MOF 1.4 (not the newest MOF 2.0 [4]) and the metamodel lacks precise semantics. This makes SPEM not as good as OMG has expected. Reasoning on models built on SPEM directly will be difficult and it is hard to keep these models consistent. How to apply process analysis techniques to such models is another problem. Therefore, to represent SPEM with more precise semantics will be a very useful task.

Ontologies are specifications of conceptualizations. Ontology contains knowledge about a domain in a precise and unambiguous manner. Description Logic (DL) [5] is a popular formalism of ontologies, and it is regarded as the foundation of some ontology languages. To represent SPEM in DL will provide significant advantages for software processes.

This paper is intended to represent OMG's Software Process Engineering Metamodel in Description Logic. In section 2, we provide rules for mapping from MOF to DL, and mapping from Object Constraint Language (OCL) [6] to DL. A representation of SPEM in DL through these mapping is presented in Section 3. Section 4 is the case study of a concrete software process model -- a portion of DMR Macroscopic. A detailed discussion of related work can be found in section 5. In section 6, we discuss the results and mention some aspects for future work.

II. MAPPING FROM MOF TO DL

Nowadays, we take an object-oriented approach to modeling a family of related software processes and use the UML as a notation. Fig.1 shows the four-layered architecture of modeling as defined by the OMG. A performing process -- that is, the real-world production process { as it is enacted, is at level M0. The definition of the corresponding process is at level M1. For example, the Rational Unified Process 2001 (RUP2001) [7], DMR Macroscopic, the IBM Global Services Method [8] and Fujitsu SDEM [9] are defined at level M1. Both a generic process like RUP and a specific customization of this process used by a given project are at level M1. Metamodel stands at level M2 and serves as a template for level M1.

Description Logics is a family of logic-based knowledge representation formalisms well-suited for the representation of and reasoning about terminological knowledge, configurations, ontologies, database schemata, etc. Architecture of a standard DL system is shown in Fig.2. A DL knowledge base is analogously typically comprised by two components -- a TBox

Shengjun Wang is with the Computer College, Jilin University, P.R.China (86-10-89177192, wsjcc @vip.163.com)

Longfei Jin is with the Computer College, Jilin University, P.R.China

Chengzhi Jin is with the Computer College, Jilin University, P.R.China

and an ABox. The TBox contains intentional knowledge in the form of a terminology and is built through declarations that describe general properties of concepts. The ABox contains extensional knowledge -- also called assertional knowledge -- knowledge that is specific to the individuals of the domain of discourse.

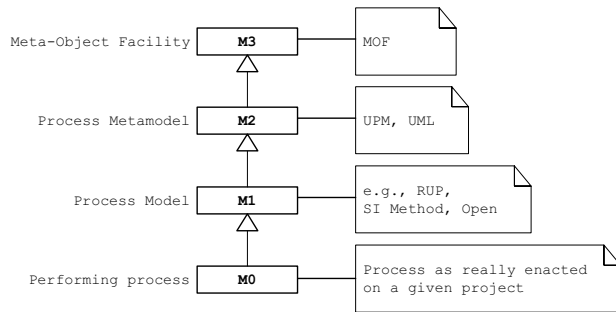


Fig. 1 Levels of modeling

OMG's architecture of modeling has the same relations that exist between concepts and individuals in DL. If elements of MOF layer are looked as concepts, then elements of process metamodel layer can be looked as individuals. If elements of process metamodel layer are looked as concepts, then process model layer can be looked as individuals. The relation of process model layer and performing process layer is the same. In this paper, we map OMG's SPEM to a DL TBox, and then models based on SPEM can be looked as an ABox. Therefore, reasoning using DL techniques on these models is feasible.

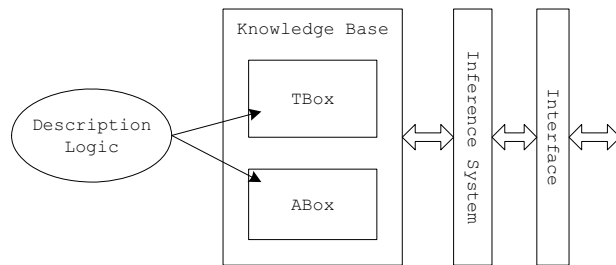


Fig. 2 Architecture of a standard DL system

It is very difficult to provide a complete mapping from MOF to DL because they do not overlap exactly. However fortunately, SPEM only use a subset of MOF 1.4, many contents that are hard to map are not used in SPEM, and a partial mapping is sufficient.

Rules of this mapping are followed.

- Class => Concept (1)
- Association => (Role, Role) (2)
- Primitive Type => Primitive_Concept (3)
- Structure Type => Concept (4)
- Enumeration Type => Concept (5)
- Alias Type => Concept (6)
- Structure Field => Role (7)
- Collection Type => Concept (8)

Attribute => Role (9)

Reference => Role (10)

Association End=> Role (11)

Constraint => Assertion (12)

Constant => Individual (13)

III. MAPPING FROM OCL TO DL

There are many well-formed rules in SPEM that act as constraints. Those constraints are written in OMG's OCL 1.4. In order to give an integrate representation of SPEM in DL, we also present a mapping from OCL to DL. This mapping is not a complete one, too.

Rules of this mapping are followed.

/*concept assertion*/
 context Kind inv => Kind(x) (14)
 elem.ocIsKindOf(Kind) => Kind(elem) (15)

/*instance of context*/
 self => x (16)

/*role assertion*/
 elem.attr => attr(elem, y) (17)
 elem.attr1.attr2 => attr1(elem, y) ^ attr2(y, z) (18)

/*cardinality assertion*/
 Association_Ends->isEmpty() => 0(Association_Ends) (19)
 Association_Ends->nonEmpty() => ≥1(Association_Ends) (20)
 Association_Ends->size = n => = n(Association_Ends) (21)
 Association_Ends->size <n => < n(Association_Ends) (22)

...
 /*others*/
 forall => ∀ (23)
 exists => ∃ (24)
 not => ¬ (25)
 and => ∧ (26)
 or => ∨ (27)
 ...

IV. A REPRESENTATION OF SPEM IN DL

A. The metamodel

OMG's SPEM specification version 1.1 contains two parallel approaches to represent SPEM -- a metamodel as an instance of MOF and a UML profile as an extension of UML. In this paper, we investigate the MOF-based metamodel. The SPEM metamodel includes packages -- Foundation, Basic Elements, Dependencies, Process Structure, Process Components, and Process Lifecycle, etc. The complete metamodel is big and cannot be shown here. We will use a portion of SPEM specification -- Process Structure package as an example. The metamodel in this example is shown in Fig.3.

B. Constraints

Well-formed rules represented in OCL associated with this portion of SPEM are followed.

Rule 1. *Each Activity is imported by exactly one Discipline.*
context Activity inv:

```
self.supplierDependency.select (d |
d.ocIsKindOf(Import)).client.select (c
c.ocIsKindOf(Discipline))->size = 1
```

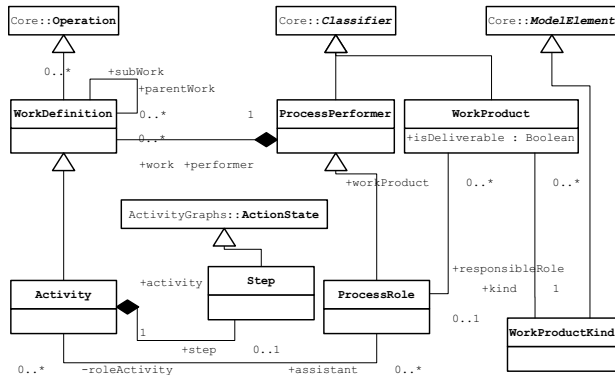


Fig. 3 A portion of software process engineering metamodel

Rule 2. *Every Activity is owned by a ProcessRole.*
context Activity inv:

```
self.performer.ocIsKindOf(ProcessRole)
```

Rule 3. *Every work must be a kind of Activity.*

context ProcessRole inv:

```
self.work->forall(f | f.ocIsKindOf(Activity))
```

Rule 4. *A Step has no associated Action.*

context Step inv:

```
self.entry->isEmpty()
```

Rule 5. *Every StateMachine (but not ActivityGraph) has a WorkProduct as its context.*

context StateMachine inv:

```
self ocIsTypeOf(StateMachine) implies
self.context->notEmpty() and
self.context.ocIsKindOf(WorkProduct)
etc.
```

C. Results of Mapping

Using those two mappings presented in Section 2, We can get a formal metamodel represented in DL and a set of axioms as constraints.

A metamodel represented in DL:

$$\begin{aligned}
 \text{WorkDefinition} &= \text{Operation} \\
 &\wedge (\exists(\text{parentWork. WorkDefinition})) \\
 &\wedge (=1(\text{performer. ProcessPerformer})) \\
 &\wedge (\exists(\text{subWork. WorkDefinition})) \quad (28)
 \end{aligned}$$

$$\begin{aligned}
 \text{Activity} &= \text{WorkDefinition} \\
 &\wedge (\exists(\text{assistant. ProcessRole})) \\
 &\wedge (\exists(\text{step. Step})) \quad (29)
 \end{aligned}$$

$$\text{Step} = \text{ActionState} \wedge (=1(\text{stepActivity. Activity})) \quad (30)$$

$$\text{ProcessPerformer} = \text{Classifier} \wedge (\exists \text{work. WorkDefinition}) \quad (31)$$

$$\begin{aligned}
 \text{ProcessRole} &= \text{ProcessPerformer} \wedge (\exists(\text{activity. Activity})) \\
 &\wedge (\exists(\text{workProduct. WorkProduct})) \quad (32)
 \end{aligned}$$

$$\begin{aligned}
 \text{WorkProduct} &= \text{Classifier} \wedge (\leq 1(\text{isDeliverable. Boolean})) \\
 &\wedge (\exists(\text{responsibleRole. ProcessRole})) \\
 &\wedge (=1(\text{workProductKind. WorkProductKind})) \quad (33)
 \end{aligned}$$

$$\begin{aligned}
 \text{WorkProduct} &= \text{Classifier} \wedge (\leq 1(\text{isDeliverable. Boolean})) \\
 &\wedge (\exists(\text{responsibleRole. ProcessRole})) \\
 &\wedge (=1(\text{workProductKind. WorkProductKind})) \quad (34)
 \end{aligned}$$

$$\begin{aligned}
 \text{WorkProductKind} &= \{\text{Text_Document, UML_Model,} \\
 &\text{Executable, Code Library, Others}\} \quad (35)
 \end{aligned}$$

Constraints represented in DL axioms:

$$\begin{aligned}
 &/* mapping result of Rule 1 */ \\
 &\text{Categorizes}(x) \wedge \text{client}(x, y) \models \text{Package}(y) \quad (36)
 \end{aligned}$$

$$\begin{aligned}
 &/* mapping result of Rule 2 */ \\
 &\text{Impacts}(x) \wedge \text{client}(x, y) \wedge \text{supplier}(x, z) \models \text{WorkProduct}(y) \\
 &\wedge \text{WorkProduct}(z) \quad (37)
 \end{aligned}$$

$$\begin{aligned}
 &/* mapping result of Rule 3 */ \\
 &\text{Import}(x) \wedge \text{client}(x, y) \wedge \text{supplier}(x, z) \models \text{Package}(y) \\
 &\wedge \text{Package}(z) \quad (38)
 \end{aligned}$$

$$\begin{aligned}
 &/* mapping result of Rule 4 */ \\
 &\text{Precedes}(x) \wedge \text{client}(x, y) \wedge \text{supplier}(x, z) \models \text{WorkProduct}(y) \\
 &\wedge \text{WorkProduct}(z) \quad (39)
 \end{aligned}$$

$$\begin{aligned}
 &/* mapping result of Rule 5 */ \\
 &\text{Activity}(x) \wedge \text{supplierDependency}(x, y) \wedge \text{Import}(y) \wedge \text{Import}(y) \\
 &\wedge \text{client}(y, z) \models (=1(\text{Discipline}(z))) \quad (40)
 \end{aligned}$$

...

V. A CASE STUDY

Following is a Software Process Engineering Model instantiation example. This example only represents a portion of a typical information system delivery process.

A. Source of The Example

Phase : Preliminary Analysis

Process : Information System Delivery Process

Subactivities

Iteration : First Joint Requirements Planning (JRP) Workshop

Subactivities

Activity : Define Owner Requirements

ProcessRole : System Architect

ActivityParameters {kind : input}

WorkProduct : Enterprise Architecture

ActivityParameters {kind : output}

WorkProduct : Assessment of Current System

{state: initial draft}

WorkProduct : Owner Requirements

{state: initial draft}

Steps

Step : Define objectives based on stated needs

Step : Define the key issues

Step : Determine the relevant enterprise principles

...

B. Results of The Example

In order to represent the mapping results clearly, we use acronym of instance's name and we represent the acronym in capital and italic letters.

Preliminary Analysis = PA

Information System Delivery Process = ISDP

First Joint Requirements Planning (JRP) Workshop = FJRPW

Define Owner Requirements = DOR

System Architect = SA

Enterprise Architecture = EA

Assessment of Current System = AOCS

Owner Requirements = OR

Define objectives based on stated needs = DOBOSN

Define the key issues = DTKI

Determine the relevant enterprise principles = DTREP

DL assertions derived from the example are followed.

Phase(PA) \wedge *subWork(PA, ISDP)* \wedge
Process(ISDP) \wedge *Iteration(FJRPW)* \wedge
parentWork(FJRPW, PA) \wedge *subWork(FJRPW, DOR)* \wedge
Activity(DOR) \wedge *performer(DOR, SA)* \wedge
parameter(DOR, AP1) \wedge *parameter(DOR, AP2)* \wedge
parameter(DOR, AP3) \wedge *parentWork(DOR, FJRPW)* \wedge
step(DOR, DOBOSN) \wedge *step(DOR, DTKI)* \wedge
step(DOR, DTREP) \wedge *ProcessRole(SA)* \wedge
work(SA, DOR) \wedge *ActivityParameters(AP1)* \wedge

behavioralFeature(AP1, DOR) \wedge *kind(AP1, pkd_in)* \wedge
type(AP1, EA) \wedge *ActivityParameters(AP2)* \wedge
behavioralFeature(AP2, DOR) \wedge *kind(AP2, pkd_out)* \wedge
type(AP2, AOCS) \wedge *ActivityParameters(AP3)* \wedge
behavioralFeature(AP3, DOR) \wedge *kind(AP2, pkd_out)* \wedge
type(AP2, OR) \wedge *WorkProduct(EA)* \wedge
typedParameter(AP1) \wedge *WorkProduct(AOCS)* \wedge
typedParameter(AP2) \wedge *WorkProduct(OR)* \wedge
typedParameter(AP3) \wedge *stepActivity(DOBOSN, DOR)* \wedge
stepActivity(DTKI, DOR) \wedge *stepActivity(DOBOSN, DOR)*

Reasoning results are shown in Table I.

Reasoning results of this example imply that the example -- a portion of DMR Macroscopic model mentioned above is not complete.

From this example, we can see that representing SPEM in DL can help reasoning on software process models based on it and some consistency checks can be done directly.

TABLE I
REASONING RESULTS

Something wrong	Reason
Phase(PA) does not have a performer.	Formula (28) denotes that a WorkDefinition must have just a performer. Phase is a subclass of WorkDefinition.
Iteration(FJRPW) does not have a performer.	See above.
WorkProduct(EA)'s kind is not clear.	Formula (33) denotes that a WorkProduct must have just a kind.
WorkProduct(AOCS)'s kind is not clear.	See above.
WorkProduct(OR)'s kind is not clear.	See above.

VI. CONCLUSION

In this paper, we present an approach of representing OMG's Software Process Engineering Metamodel (SPEM) in Description Logic (DL). Two mappings of this approach is provided in a formal manner. Based on this approach, software process models based on SPEM can be represented in DL. Thus, Reasoning and other process analysis techniques can be applied on those models.

Our work is just in its initial state. We have lots of work to do in the future. A new SPEM based on MOF 2.0 will give more precise semantics in OMG's style. A complete mapping from MOF 2.0 to DL will give significant benefits and give tight relations among software engineering and ontological engineering. Replacing OCL with DL or some kind of extension of DL will enable a direct reasoning capability. Some consistency checks based on DL techniques will be investigated in the future.

REFERENCES

- [1] *Software Process Engineering Metamodel Specification (Version 1.1)*, OMG Adopted Specification: formal/05-01-06, 2005.
- [2] *MetaObjectFacility (MOF) Specification (Version 1.4)*, OMG Specification, 2002.
- [3] *OMG Unified Modeling Language Specification (Version 1.4.2)*, OMG Specification: formal/04-07-02, 2004.
- [4] *Meta Object Facility (MOF) 2.0 Core Specification*, OMG Adopted Specification: ptc/03-10-04, 2003.
- [5] F. Baader, *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003, pp. 1–65.
- [6] *UML 1.4 - chapter 6 - Object Constraint Language Specification*, OMG Document: formal/01-09-77, 2001.
- [7] *Rational Unified Process (RUP)*, Rational Software Corporation, 2000.
- [8] IBM Object-Oriented Technology Center, *Developing Object-Oriented Software - An Experience-Based Approach*. Prentice-Hall, 1997, pp. 1-54.
- [9] M. Itakura, "SDEM90: A framework for system development activities and responsibilities" in *Proc. of The 2nd International Conf. on System Integration*, Morristown, N.J, 1992, pp. 359-363.