An Axiom Based Metamodel for Software Process Formalisation: An Ontology Approach

Edward Kabaale[2(✉)], Lian Wen[1,2], Zhe Wang[1,2], and Terry Rout[1]

[1] Institute for Integrated and Intelligent Systems, Griffith University, 170 Kessels Road, Brisbane, QLD 4111, Australia
{l.wen,z.wang,t.rout}@griffith.edu.au

[2] School of Information and Communication Technology, Griffith University, 170 Kessels Rd, Brisbane, QLD 4111, Australia
edward.kabaale@griffithuni.edu.au

Abstract. Software development usually follows well known process models and standards for development processes. However, these are usually diverse and described in natural language which complicates their automation, adaptivity and verification. The need for process formalisation has long been highlighted, and we have provided a formalisation and translation algorithm to that effect in earlier work. However, to systematically and faithfully formalise heterogeneous processes from different standards and process models, there is a need to utilise uniform concepts to underpin the formalisation process. Metamodels and ontologies have been explored recently to lay a foundation for structuring and expressing additional rigour to process formalisation. In this study, we develop an axiom based metamodel utilising powertype patterns as a conceptual framework to underpin homogeneous process formalisation. The advantage of an axiomatic and powertype based metamodel approach lies in its potential to determine the metamodel basic constituents and formalism as well as its extensibility and adaptability. We formalise the metamodel using ontologies while adopting use cases from ISO/IEC 29110 and ISO/IEC 24744 standards for metamodel illustrations. Ontology based process descriptions enable process automated verification and adaptivity capability through the use of ontology reasoning support engines.

Keywords: Software process · Metamodel · Powertype · Axiom · Ontology

1 Introduction

Software Engineering (SE) focuses on sound processes and methods for quality software development within budget and time frame. Over the recent decades, the process dimension of SE has received increased attention from both researchers and practitioners [1]. One of the main objectives of this dimension is to enhance the software product quality through formal definition and improvement of the process by which software is developed and maintained. Consequently, it boosts a wide spectrum of approaches to process definition such as

A. Mas et al. (Eds.): SPICE 2017, CCIS 770, pp. 226–240, 2017.
DOI: 10.1007/978-3-319-67383-7_17

ISO/IEC 12207 [2], ISO/IEC 29110 [3], process assessment and improvement such as ISO/IEC 33061 [4] and CMMI [5], and process metamodels such as OOSPICE [6], SPEM [7] and SEMDM [8] that specify the conceptual foundations for process modeling. However, these are mainly described in natural language and published in manuals and booklets [9–11], this presents several challenges such as difficulty in finding or updating information in different versions of the same document, lack of automated auditing and verification that limits adherence and monitoring (e.g. constraint checking of required relations among activities, work products and roles). Even though these process modeling techniques enhance software development activities, they still need formal enhancement to enable process automation and verification [10–13].

A formal software process specification is a specification expressed in a language whose vocabulary, syntax and semantics are formally defined and well understood. Its a precise and concise specification that supports formal software process definition and management, automated analysis, verification and validation, understanding, evolution management, classification, improvement and aiding in choosing the appropriate process for a given project [11]. Formal methods such as Petri nets, algebra, ontologies, bayesian networks and composition trees have been used in modeling and formalising software process before. For example composition trees have been used to formally model and compare software processes in [14,15]. Ontology based approaches [10,16–18] have also been proposed to model,validate, constrain and query software process descriptions. Ontologies are being deployed in industry to formalise information models and standards that would otherwise be costly to develop, integrate and share, and monitor through automated queries and verification [16]. Even though, some practitioners especially in small entities use informal process descriptions in industry [19], there is a great need and usage of formal process descriptions in practice as well [11]. However, due to the great diversity and complexity of software processes from different standards and process models coupled with varying situational contexts [20], software process formalisation still lacks a standardized, consistent and faithful way making it error prone, time consuming and thus expensive [21].

Metamodels have been proposed as a way of increasing process modeling rigour and formality for automation [9]. These metamodels lay a foundation in terms of concepts, rules and conceptual relationships among concepts used in process modeling [22]. So processes grounded in metamodels offer a higher degree of formalisation and better support for consistent extensions and modification [9]. Moreover, they impose well formedness rules on process models and process instances instantiated from them. These help in maintaining process consistency and completeness [23]. However, the current process metamodels such as SPEM and SEMDM are too generic and complex to guide software process formalisation [22]. For example, SEMDM was intended to be general methodology metamodel and cover not only process modeling but also other areas like computer supported collaborative work (CSCW)[9]. Indeed such genericity and complexity hinders their comprehension that would be of great help to software

process formalisation. Therefore, there is a need for a customised and simple metamodel tailored for homogeneous software process formalisation. To develop this metamodel, we utilise the powertype pattern as introduced in process metamodeling by [24]. The powertype pattern enables tailoring of software process formalisation to specific project requirements and contexts.

Even though, metamodels provide a rigous underpinning and consistent terminology to various aspects of SE, they only deal with conceptual definitions, standardisations and syntax of process models necessitating the need for formal semantics and reasoning of such conceptual definitions [25,26]. We therefore formalise the proposed metamodel using OWL ontologies. Ontologies consititute formal models that define formal semantics and inference services for a shared conceptualisation. These can be used to draw interesting logical conclusions through for example (meta)model checking, model enrichment, dynamic classification, information retrieval and querying of software process models across the metamodel hierarchy thus improving software engineering processes [27].

In this study, we propose an axiom-based metamodel to underpin a homogeneous software processes formalisation. We design this metamodel through rising the abstraction levels of common process elements which define the structure of the process models. The metamodel is grounded in an ontology, reusable, and adaptable. In Sect. 2, we discuss background information on software process meta (modeling) and process instantiation, powertypes and ontologies. In Sect. 3, we abstract the axiom metamodel design from existing process constructs, and illustrate a general overview of the metamodel with processes from ISO/IEC 29110. In Sect. 4, we discuss the formalisation of the metamodel through OWL DL that equips it with formal semantics and reasoning capabilities. Finally, we conclude the paper and identify some future research directions.

2 Background

2.1 Process Metamodeling

Process metamodels are a feasible approach to reduce process modeling complexities through rising the abstraction levels, reuse, and formalisation [22]. According to [28] process metamodels describe a *conceptual framework for expressing and composing software process models*. They describe the relevant software process sub-models, basic concepts, rules and relationships among concepts with notations for expressing process models. They allow capturing informal, behaviour, functional, and strategic views of software processes [22]. Such information can then be used to reason on software process modeling for changes, formalisation, improvements and updates [29]. Software process models are the key result of the process modeling activity and instances of process metamodels. They serve as abstract representations of software processes. These prescribe a software process in terms of the activities to be carried out, the roles and work product types involved. Software processes are then instantiated in a specific project endeavour to develop the desired software product, which in itself is seen as an instantiation of the software process.

Process metamodeling is an important conceptual tool in underpinning the definitions of formal software process models. This has largely been popularised by the OMG standards where the UML is now the de facto standard formalism for software modeling. The modeling layers in UML metamodel are defined based on *strict* metamodeling architecture that only allows *instanceOf* relations between adjacent layers, i.e., variables defined at level M_n can only be realised at M_{n-1}. This is termed as *shallow* instantiation where attributes and constraints are defined at the class level and only realised at the instance level [33]. Where as the OMG standard approach works well for modeling languages defined at OMG level M_2 and used at level M_1, its insufficient for process standardisation that requires *deep* instantiation [24]. That is attributes and constraints defined at metamodel M_2 are realised (enacted) on real world projects at level M_0 spanning multiple modeling levels. When attempt to use UML shallow instantiation for process metamodeling, it results into modeling challenges such as *accidental complexity* [24]. To overcome these challenges, Gonzalez-Perez and Henderson-Sellers [24] introduced the use of powertypes as a way of deep instantiation for process metamodeling.

Powertype Based Metamodeling Framework. The Powertype pattern is a flexible and scalable modeling technique that combines instantiation and generalisation semantics in process metamodeling [9]. Mainly introduced as an alternative solution to the inconsistencies, ambiguities and accidental complexities that result from the use of strict metamodeling technique in process standardization [24]. The powertype pattern has been extensively applied in process modeling [24] and underpins the development of an international standard for metamodeling, i.e., ISO/IEC 24744 Software Engineering - Metamodel for Development Methodologies (SEMDM)[8], therefore, in this study we utilise it to underpin modeling and tailoring of software process formalisation to specific software project contexts through the developed metamodel.

Essentially a powertype is a class whose instances are subclasses of another class called a partitioned class [24]. In this regard a powertype is more like a metaconcept with an extra twist that, its instances can also be subclasses of another class. The powertype and partitioned class are closely related and together with the relation between them form a *powertype pattern*. The powertype class represents groups of instances that are used to classify the partitioned class according to a partitioning discriminator (powertype attribute value, e.g., name). For example, *birds* can be classified according to *birdSpecies* such as *eagle* and *penguin*. The powertype pattern enables the combination of generalisation and instantiation across metamodeling layers through *dual* representation of concepts (also known as *clabject*) [33] where the instance of the powertype and a subtype of a partitioned class are the same thing. From our example *eagle* as an instance of bird species is the same thing as *Eagle*, the subtype of bird. However, metamodels only deal with the syntactical structures but not the formal semantics of process models and therefore, the need to ground the metamodel in an ontology.

2.2 Ontologies

New technologies such as semantic web that provide reasoning support services like consistency checking, information retrieval and querying of software process models are beneficial to improving software engineering processes [34]. OWL[1] is a Semantic Web language designed and standardised by W3C to represent rich and complex knowledge about things, groups of things, and relations between things [35]. Its a knowledge representation language underpinned by Description Logic (DL) that enables expressed knowledge to be reasoned on by human and artificial agents for consistency and inferring implicit knowledge from the explicit one. While there are different OWL flavours such as OWL Full, OWL Lite and OWL DL. Here we are interested in utilising a newer version of OWL DL, i.e., OWL DL 2 [35].

OWL representations commonly referred to as ontologies, can be published and stored in the World Wide Web. An OWL DL ontology is mainly composed of two main components; The Terminological knowledge represented in the TBox (Class Level) and the Assertional knowledge forming the ABox (Instance Level). The TBox defines the intensional knowledge by which a concrete world can be described. This knowledge is represented by axioms in the form of logical sentences. The ABox on the other hand, represents assertional knowledge that complies with the intensional knowledge in the TBox.

In order to use the modeling capabilities of OWL and the potential of DL reasoning in a layered architecture, the OWL based modeling language has to provide essential features to support metamodeling and reasoning across layers; To this effect, the current W3C standard Web Ontology Language (OWL) supports metamodeling mainly in two flavours; OWL Full that implements full metamodeling like in RDF is suitable for formalising the powertype metamodeling approach but is undecidable even for basic inference problems [36]. OWL 2[2] supports a decidable fragment of metamodeling based on *contextual semantics* [36] through the *Punning* technique where the same identifier is used to denote both the ontology class and an individual. It is common place in conceptual and ontology engineering to classify entities either as classes or instances depending on their context [36], given the fact that there is no clear cut borderline between classes and instance classification. In fact some entities are classified as classes in higher abstraction levels and instances in lower abstraction levels making the borderline blurred. Therefore, we use OWL 2 *punning* to model powertypes and clabjects concepts into OWL 2 ontologies for automated verification using various off the self ontology reasoning engines such as FACT++ and HERMIT[3].

3 Axiom Metamodel Design

Process metamodel approaches should identify the most appropriate concepts not only to represent process models but also process assessment [22,29].

[1] https://www.w3.org/OWL/.
[2] https://www.w3.org/TR/owl2-overview/.
[3] http://owl.cs.manchester.ac.uk/tools/list-of-reasoners/.

The process is the main concept of any software process metamodel [29]. Therefore, every process has some common basic elements such as activity, work products and roles [29,30]. However, different metamodels and standards use them differently in terms of their granularity, formality and abstraction. These form the tangible internal structure of the process and are important in developing a process model of such processes [17,30–32].

To enhance homogeneous software process formalisation, understandability and reduce model maintenance efforts and costs, process modeling should be supported by a very high level of abstraction [17,31]. To this end, we rise the abstraction levels of these common elements and generalise them to design new abstract concepts for the metamodel. Key abstract concepts for the metamodel are stated as axioms. Axioms are statements which are accepted as true [37]. In this regard, their accuracy doesn't need to be proven [37] and can be used as a basis for argument or inference. Axioms have been used widely in providing foundations for theoretical works [38,39] in SE. In here, we use these axioms to provide a theoretical foundation for the essential metamodel concepts for process formalisation. Moreover, through these axioms new metamodel concepts are accepted as true and valid [37]. The main metaconcepts for the metamodel are **Achievable**, **Doable**, **Tangible** and **Assessable**. The relationship between these axioms is visualised in Fig. 1. To demonstrate these concepts we use the software implementation process from ISO/IEC 29110. ISO/IEC 29110: Systems and Software Life Cycle Profiles and Guidelines for Very Small Entities (VSEs) [3] is an international standard (IS) for VSEs employing not more than 25 people on small software projects (less than six people month). It has two main processes, i.e., software implementation(SI) and project management (PM). These have been mainly drawn from other major standards such as ISO/IEC 12207 and ISO/IEC 15289.

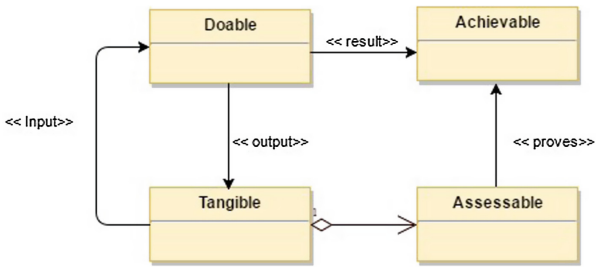

Fig. 1. Relationship between metamodel concepts

Achievable Axiom: *For every process performed, there is something to be achieved.* The purpose of the process can be achieved through the process objectives and the demonstration of the outcomes. We collectively term these as *Achievable*. See Fig. 2 for details. The outcomes are examined for work products and the execution of the base practices to achieve the work products.

These collectively can help to determine the capability of the performed process. The achievable axiom represents what needs to be achieved when performing a selected process. The process purpose is a high level objective of performing a process whose achievement is demonstrated through the outcomes. The ISO/IEC 29110 process objectives are provided by one or more outcomes from ISO/IEC 12207 standard. Therefore the objectives are the specific goals through which the process purpose is accomplished [30]. Because of this relationship, we collectively term this as achievable, after all process objectives and outcomes are meant to ensure successfully accomplishment of a process purpose.

We utilise the xKind metaconcepts from the powertype pattern to tailor the approach to different specific project contexts [9,31] by aggregating process characteristics that match a given project context. For example, in Fig. 2, we have subtypes of achievable as purpose, objectives and outcomes but we also have the same subtypes as instances of the achievablekind where we can assert their characteristics such as capability levels. Through enactment of the purpose subtype on a specific project we are able to specify the individual purpose for such a project for example *developing a website for project A*. Collectively such instances aggregate the characteristics of the achievablekind instances.

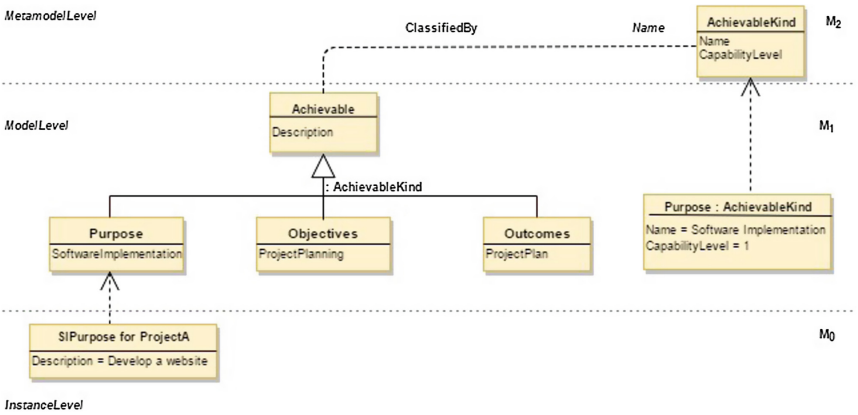

Fig. 2. Achievable Axiom showing process purpose, objectives and outcomes

Doable Axiom: *In order to fulfil the achievables, there is something to be performed.* The basic concepts in performing a process can be generalised into a common abstract concept named *Doable*, see Fig. 3, which represents anything that is performed to fulfill the achievables. This includes processes, activities, tasks and steps. The difference between these is the level of granularity at which they are performed. Whereas the process is a more general concept in the process hierarchy, the task and step on the other hand are more atomic for enactment [30]. The activity hierarchy shows that activities are described at different granularity levels with varying attributes. For example, the activity hierarchy is represented

as *domain concept* in OOSPICE, *WorkUnits* in SEMDM and *activity* in [30]. Makinen proposes the use of abstract class *activity* so that uniform attributes can be inherited by all other concrete classes within the activity hierarchy, but then activity itself is also part of the activity hierarchy. Our *doable* class is a generalisation of all things performed to fulfil the process purpose including the process, activity, task and steps. From Fig. 3, we have the main process as *software implementation* with various activities like *requirements analysis* and these are supported by tasks such as *elicit requirements*. The duration of the doable class will be the aggregation of all these activities sharing the same attributes whose values are taken at the project level. On the other hand, the doablekind class provides attributes for the process model level describing the characteristics of all the kinds of processes, activities and tasks.

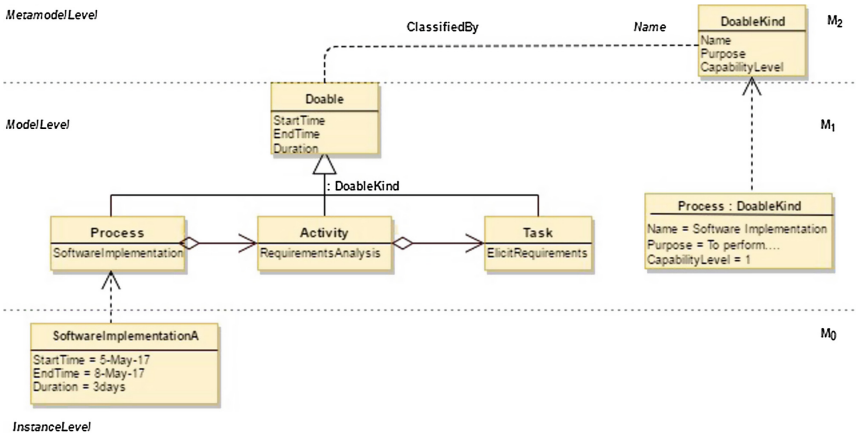

Fig. 3. Doable Axiom showing software implementation process example

Tangible Axiom: *In order to perform the doable, it requires something as input to produce the desired output.* The main purpose of performing a process is to produce outputs. *Tangible* are the inputs and outputs of performing the *doable* see Fig. 4. The tangible within ISO/IEC 29110 at the abstract level are input, internal or output products. When they are outputs, they are always associated with a *destination*. This destination can either be another process for example *project plan* from project management process to software implementation process [3] or just going outside of the process, for example *acceptance document* goes outside of the implementation process to the customer. The tangible can also be inputs to the *doable* for example a *project plan* is an input to the software implementation process. Other times the tangible can be internal work products such as *ChangeRequest* from within the process [3].

Assessable Axiom: *The tangible can be objectively observed and measured to prove if the achievable are fulfilled.* The assessment of the quality of execution and outputs produced is normally done through process assessment.

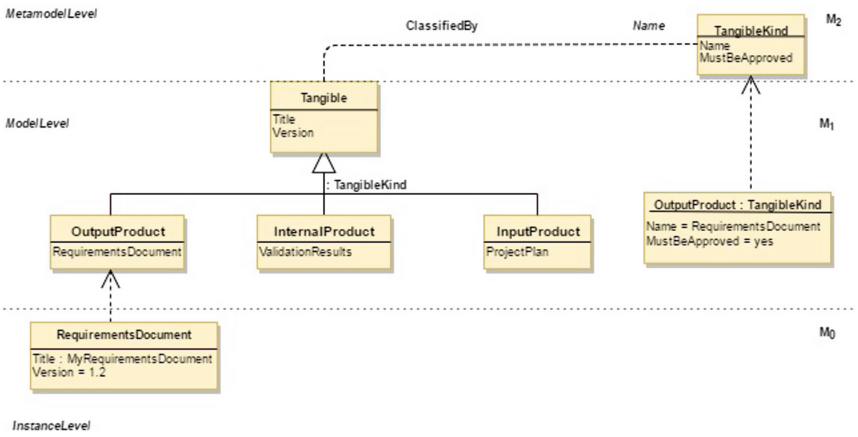

Fig. 4. Tangible Axiom showing work products

Such assessments are normally carried out using prescribed assessment models such as ISO/IEC 33061 [4] and CMMI [5]. These models outline the abstract properties for the process assessment such as process purpose and outcomes [29]. We refer to this collectively as *Assessable*. Process assessment indicators are grouped into two categories as process performance and capability assessment. Its the process performance category that assesses the accomplishment of a process purpose through the process outcomes. Moreover, ISO/IEC 29110 doesn't state any process capability beyond process performance [40]. We therefore limit our assessment to only process performance, i.e., process capability level one. The main assessment indicators for process performance are the *work products* and *base practices* [4] and have been used to develop a process assessment model (PAM) and method for VSEs operating at process capability level one [40]. The work products have been already discussed in the tangible axiom, see Fig. 4. The base practise according to [4] is *an activity that addresses the purpose of a particular process. A coherent set of base practices is associated with each process in the process dimension.* And importantly base practices are described at an abstract level and have been linked to taskkind by [31]. The taskind classes that represent base practices have also been discussed in the doable axiom see Fig. 3.

According to [29] a metamodel that directly supports the concept of capability levels enhances the definitions of process models that are dynamically tailorable along their capability levels. It should be noted, however, that from the *Doable* its only the process that is assessable because its associated with a purpose and outcomes while others are not [29]. An *assessable process metamodel* is one that incorporates the necessary formal properties for assessment so that no external process reference model is needed [29]. A good metamodel should address the different aspects of a process model such as the process hierarchy (*Doable*), work products (*Tangible*), objectives (*Achievable*) as well as the

assessment (*Assessable*) of the entire process [29]. Its in this regard that we provide our axiom based metamodel for process formalisation as discussed above that addresses the key different aspects of process models.

4 Ontology Based Process Metamodeling

The need to fix formal semantics for metamodels and their instantiated process models has long been highlighted by different authors [25–27]. Metamodel formal semantics are needed for satisafiability checking at the process model layers as well as checking the consistency of the instantiated process models [27]. The current approaches [27] emphasise fixing formal semantics for metamodels and the corresponding process models helping to maintain consistency between metamodels and their instantiated process models [26]. For example in [27], both the metamodel and process models are transformed to ontologies where the metamodel is transformed to the OWL DL TBox and the process model is transformed to the OWL DL ABox. In [26] domain ontologies are used to define formal semantics for process models while meta-ontologies or foundational ontologies are used to define formal semantics for metamodels. A reference ontology for the domain of software engineering standards has been developed by [31] where a common and unambiguous terminology is sought for all (current and future) software engineering standards developed by ISO/IEC JTC1's SC7. Software engineering standard harmonization ontologies [32] have also been developed to harmonise concepts and term usage across different process models and references models in software engineering domain. All these studies lay a theoretical foundation for the work presented in this paper.

However, the semantics and consistency of the instantiated processes in respect to the metamodel are largely undefined in the current approaches. In earlier work [13], we have shown how the semantics of the instantiated process can be used for process reasoning, verification and conformance to the process model. This provides our motivation for the current section by extending the semantic mapping from the metamodel further to the process instances. This can guarantee the semantic consistency right from the metamodel upto the instantiated process. Hence, we map the metamodel at M_2 to a OWL DL TBox, the clabject (dual entity) at M_1 to OWL DL TBox/ABox (OWL punning) and the instantiated process M_0 to OWL DL ABox. See Fig. 5 for details. With such a mapping, we are able to utilise ontology reasoning support across the metamodeling layers such as satisfiability checking between metamodels and process models, process models and process instances as well as process metamodels and process instances. Moreover, we can enforce and check the well formedness constraints that the metamodel imposes on the process models and process instances respectively.

To illustrate our approach, we take the Tangible metaconcept for an example from our approach as shown by Fig. 5. At level M_2, we have the $TangibleKind$ class(Powertype) that classifies or partitions the $Tangible$ class (partitioned class) at M_1 using *attribute name* as a *partitioning discriminator*. The $Tangible$

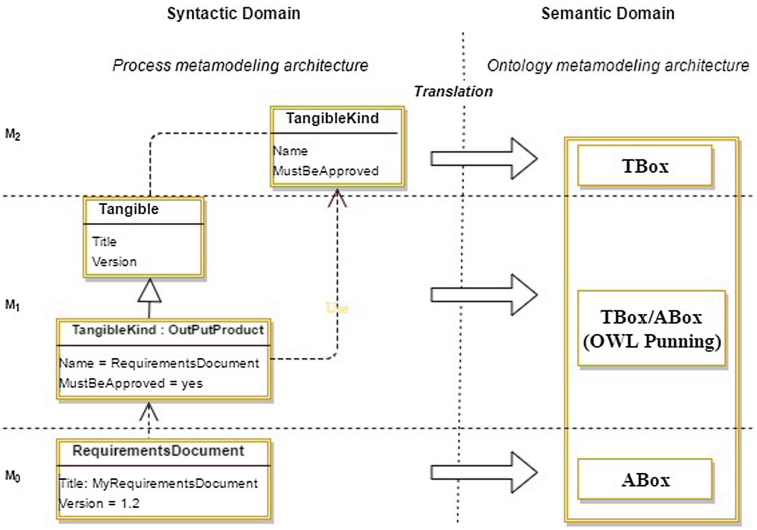

Fig. 5. Ontology based process metamodeling architecture

class at M_1 is subtyped into different kinds such as output, internal and input products (only output products-requirementsdocument is shown in Fig. 5). At the same time the instances of the $TangibleKind$ class are exactly the subtypes of the $Tangible$ class forming a dual concept also known as the *clabject* in our case the *RequirementsDocument* that has both the class facet from the $Tangible$ class and the object facet from the $TangibleKind$. The class facet of the clabject represents the actual output products from enacting the process such as the *RequirementsDocument* which can further be instantiated in a specific project to yield *MyRequirementsDocument* version 1.2. On the other hand, the instance facet of the *RequirementsDocument* is used as a template from which other versions of requirement documents may be later instantiated.

The metamodel is translated to an OWL DL ontology through a translation algorithm developed in our earlier work [13] for consistency checking, verification and query answering. The $TangibleKind$ and $Tangible$ classes at M_2 and M_1 respectively are translated to *OWL DL TBox*. The class facet of the *clabject* from the $Tangible$ class is translated to the $TBox$ while the object facet of the *clabject* from the $TangibleKind$ class is translated to the *OWL DL ABox* at the same level through OWL 2 *Punning* technique. OWL 2 Punning is similar to clabject in process metamodeling because it treats one identifier in this case *RequirementsDocument* as an ontology class and an individual based on the context in which the identifier is used in the ontology. For example, the following axioms state the fact that *RequirementDocument* is a $Tangiblekind$, and that $MyReuirementDocument$ is a $RequirementDocument$:

$$ClassAssertion(RequirementDocument, MyRequirementDocument) \quad (1)$$

$$ClassAssertion(Tangiblekind, RequirementDocument) \quad (2)$$

We can note from these axioms that the symbol (clabject) *Requirements Document* is used in (1) as a class and in (2) as an individual. This serves our purpose earlier stated, for example requirements document as a subtype of *Tangible* class can be treated as a class where *RequirementsDocument* can be further instantiated into $MyRequirementsDocument1.2$, but on the other hand, it can also be treated as an instance of the $TangibleKind$ class and therefore translated to OWL DL TBox where the characteristics of the requirement document *kind* can be asserted such as their approvals. This approach can provide practical support and enhance dynamic software process formalisation, tailoring, assessment and process run time modeling and verification. The formal approach presented in this paper is part of an ongoing work towards software process formalisation and automation where process monitoring, adaptability, and verification can be enabled. Furthermore, ontology reasoning engines can be used to perform automated reasoning and verification on the former model to ensure process (model) well formedness.

5 Conclusion and Future Works

This paper presents an axiom based metamodel towards systematic and faithfully software process formalisation as part of an ongoing work for software process automation and verification. The main aim of the metamodel is to provide uniform formal concepts at an abstract level that may be utilised for software process formalisation. Powertype pattern has been utilised to develop the metamodel with the view of enabling process model extensibility and flexibility with ability to model run time processes and their verification. Powertypes also enable tailoring of processes to different specific projects through the use of *xKind* and *clabject* concepts. This helps to create different hierarchical views/contexts for the modeled process, a limitation earlier identified with UML *strict* metamodeling. Finally we formalise the metamodel using OWL DL into formal ontology. Especially, we formalise the powertype and clabject using the OWL DL 2 punning technique. This enables various types of verification at different levels to be carried. For example, we are able to utilise ontology reasoning tool support across the metamodeling layers such as satisfiability checking between process models and metamodels, process models and process instances as well as process metamodels and process instances.

Software development processes are typically too complex to be modeled and maintained without the help of tools [11]. As this work is ongoing, we are yet to evaluate it in a practical setting. However, the work presented in this paper and the algorithm developed in earlier work [13] forms a conceptual foundation for the development of a software process translation tool that we are currently developing. This tool will enable us to evaluate our approach in a practical setting in future work.

References

1. Fuggetta, A.: Software process: a roadmap. In: ICSE 2000: Proceedings of the International Conference on Software Engineering (ICSE)(2000)
2. ISO/IEC FDIS 12207:2017. Systems and software engineering - Software life cycle processes (2017)
3. ISO/IEC TR 29110-5-1-2:2011, Software engineering Lifecycle profiles for VSEs: Management and engineering guide: Generic profile group: Basic profile (2011)
4. ISO, ISO/IEC 33061:2015 Information technology - Process capability assessment model for software life cycle processes (2015)
5. CMMI Product Team, CMMI for Development, Version 1.3, Software Engineering Institute, Carnegie Mellon University (2010)
6. OOSPICE, Software Process Improvement and Capability Determination for Object-Oriented/Component-Based Software Development (2002)
7. Object Management Group: Software and Systems Process Engineering Meta-Model 2.0, formal/2008-04-01. Object Management Group, USA (2008)
8. ISO/IEC, 2007. ISO/IEC 24744. Software Engineering Metamodel for Development Methodologies. ISO, Geneva (2007)
9. Henderson-Sellers, B., Gonzalez-Perez, C.: A comparison of four process metamodels and the creation of a new generic standard Inform. Softw. Technol. (2005)
10. Gallina, B., Szatmári, Z.: Ontology-based identification of commonalities and variabilities among safety processes. In: Abrahamsson, P., Corral, L., Oivo, M., Russo, B. (eds.) PROFES 2015. LNCS, vol. 9459, pp. 182–189. Springer, Cham (2015). doi:10.1007/978-3-319-26844-6_13
11. Diebold, P., Scherr, S.: Software process models vs. descriptions: What do practitioners use and need? J. Softw. Maint. Evol. Res. Pract. (2017)
12. Tarhan, A., Giray, G.: On the use of ontologies in software process assessment: a systematic literature review. In: EASE (2017)
13. Kabaale, E., Wen, L., Wang, Z., Rout, T.: Representing software process in description logics: an ontology approach for software process reasoning and verification. In: Clarke, P.M., O'Connor, R.V., Rout, T., Dorling, A. (eds.) SPICE 2016. CCIS, vol. 609, pp. 362–376. Springer, Cham (2016). doi:10.1007/978-3-319-38980-6_26
14. Wen, L., Tuffley, D., Rout, T.: Using composition trees to model and compare software process. In: O'Connor, R.V., Rout, T., McCaffery, F., Dorling, A. (eds.) SPICE 2011. CCIS, vol. 155, pp. 1–15. Springer, Heidelberg (2011). doi:10.1007/978-3-642-21233-8_1
15. Wen, L., Rout, T.: Using composition trees to validate an entry profile of software engineering lifecycle profiles for very small entities (VSEs). In: Mas, A., Mesquida, A., Rout, T., O'Connor, R.V., Dorling, A. (eds.) SPICE 2012. CCIS, vol. 290, pp. 38–50. Springer, Heidelberg (2012). doi:10.1007/978-3-642-30439-2_4
16. Kharlamov, E., Grau, B.C., Jiménez-Ruiz, E., Lamparter, S., Mehdi, G., Ringsquandl, M., Nenov, Y., Grimm, S., Roshchin, M., Horrocks, I.: Capturing industrial information models with ontologies and constraints. In: Groth, P., Simperl, E., Gray, A., Sabou, M., Krötzsch, M., Lecue, F., Flöck, F., Gil, Y. (eds.) ISWC 2016. LNCS, vol. 9982, pp. 325–343. Springer, Cham (2016). doi:10.1007/978-3-319-46547-0_30
17. Liao, L., Qu, Y., Leung, H.K.N.: A Software Process Ontology and its Application (2005)

18. Clarke, P.M., Calafat, A.L.M., Ekert, D., Ekstrom, J.J., Gornostaja, T., Jovanovic, M., Johansen, J., Mas, A., Messnarz, R., Villar, B.N., O'Connor, A., O'Connor, R.V., Reiner, M., Sauberer, G., Schmitz, K.-D., Yilmaz, M.: Refactoring software development process terminology through the use of ontology. In: Kreiner, C., O'Connor, R.V., Poth, A., Messnarz, R. (eds.) EuroSPI 2016. CCIS, vol. 633, pp. 47–57. Springer, Cham (2016). doi:10.1007/978-3-319-44817-6_4
19. Kabaale, E., Nabukenya, J.: A systematic approach to requirements engineering process improvement in small and medium enterprises: an exploratory study. In: Caivano, D., Oivo, M., Baldassarre, M.T., Visaggio, G. (eds.) PROFES 2011. LNCS, vol. 6759, pp. 262–275. Springer, Heidelberg (2011). doi:10.1007/978-3-642-21843-9_21
20. Jeners, S., P. Clarke, P., OConnor, R.V., Buglione, L., Lepmets, M.: Harmonizing software development processes with software development settingsA systematic approach. Commun. Comput. (2013)
21. Garca, F., Serrano, M., Cruz-Lemus, J., Ruiz, F., Piattini, M.: Managing software process measurement: a metamodel-based approach. Inf. Sci. (2007)
22. Martins, P.V., da Silva, A.R.: PIT-ProcessM: a software process improvement meta-model. In: QUATIC, 7th International Conference (2010)
23. Pereira, E., Bastos R., da C. Mra, M., Oliveira T.: Improving the consistency of SPEM based software processes. In: Proceedings of the 13th ICEIS (2011)
24. Gonzalez-Perez, C., Henderson-Sellers, B.: A powertype-based meta- modelling framework. Softw. Syst. Model. (2006)
25. Henderson-Sellers, B.: Bridging metamodels and ontologies in software engineering. J. Syst. Softw. **84** (2011)
26. Saeki, M., Kaiya, H.: On relationships among models, meta models and ontologies. In: Proceedings of Workshop on Domain-Specific Modeling (2007)
27. Staab, S., Walter, T., Gröner, G., Parreiras, F.S.: Model driven engineering with ontology technologies. In: Aßmann, U., Bartho, A., Wende, C. (eds.) Reasoning Web 2010. LNCS, vol. 6325, pp. 62–98. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15543-7_3
28. Lonchamp, J.: A structured conceptual and terminological framework for software process engineering. In: The ICSP 2. IEEE Computer (1993)
29. Gonzalez-Perez, C., McBride, T.M., Henderson-Sellers, B.: A metamodel for assessable software development methodologies. Softw. Qual. J. (2005)
30. Makinen, T., Varkoi, T.: Analyzing a Process Profile for Very Small Software Enterprises. In: SPICE (2008)
31. Gonzalez-Perez, C., Henderson-Sellers, B., McBride, T., Low, G.C., Larrucea, X.: An Ontology for ISO software engineering standards: 2) Proof of concept and application. Comput. Stand. Interfaces (2016)
32. Pardo-Calvache, C.J., Garca-Rubio, F.O., et al.: A reference ontology for harmonizing process-reference models (2014)
33. Atkinson, C., Kühne, T.: The essence of multilevel metamodeling. In: Gogolla, M., Kobryn, C. (eds.) UML 2001. LNCS, vol. 2185, pp. 19–33. Springer, Heidelberg (2001). doi:10.1007/3-540-45441-1_3
34. Jekjantuk, N., Groner, G., Pan, J.Z.: Modeling and reasoning in metamodeling enabled ontologies. Int. J. Softw. Inf. (2010)
35. OWL 2 Web Ontology Language Primer, 2nd edn., https://www.w3.org/TR/2012/REC-owl2-primer-20121211
36. Motik, B.: On the properties of metamodeling in OWL. J. Logic Comput. **17**(4), 617637 (2007)

37. Suh, N.P.: Axiomatic Design: Advances and Applications. Oxford University Press, New York (2001)
38. Kim, S.J., Suh, N.P., Kim, S.: Design of software systems based on AD. Ann. CIRP **40**(1), 16570 (1991)
39. Arsenyan, J., Bykzkan, G.: Modelling collaborative software development using axiomatic design principles. IAENG (2009)
40. Varkoi, T.: Process assessment in very small entities-An ISO/IEC 29110 based method. In: 7th International Conference QUATIC. IEEE (2010)