Send Orders for Reprints to reprints@benthamscience.ae

Recent Patents on Computer Science, **Year**, *Volume, Page Enation*

RESEARCH ARTICLE

Defining Theoretical Foundations to Unified Metamodel For Model Reusability

Jagadeeswaran Thangaraj[*a] and Senthilkumaran Ulaganathan[b]

[a]*Department of Computer Science, Maynooth University, Maynooth, Ireland;*
[b]*School of Information Technology & Engineering, VIT University, Vellore, TamilNadu, India*

ARTICLE HISTORY

Received:
Revised:
Accepted:

DOI:

Abstract:

Background: In model driven development, model transformation transforms one model to another model between different phases of software engineering. In model transformation, metamodel plays vital role which defines abstract syntax of models and interrelationship between their elements. Unified metamodel defines abstract syntax for both source and target models when they share core elements. Theoretical approaches define language and platform independent representation to models in software engineering. This paper investigates the theoretical foundation to this unified meta-modelling for their consistent transformation.

Objective: This paper aims to define formal foundations to the Unified metamodel for generating implementation from design specifications and model reusability.

Method: In this paper, the study considers transformation from design to implementation and vice versa using theoretical foundations to build verified software systems.

Results: The related tools provide formal representation to the design phase or verification purpose. Our approach provides set theoretical foundation to the Unified metamodel for model transformation from USE (UML/OCL) to Spec#. In other words, our approach defines formal foundation to define a model which consists of all the required properties for verification at design and implementation phase.

Conclusion: This paper introduced a new set theoretical framework which to be act as an interface between design and implementation to generate verified software systems.

Keywords: Formal framework, Theoretical Foundation, Unified Metamodel, Model Transformation, USE, Spec#.

1. INTRODUCTION

In the software engineering life cycle, a model plays a systematic use in Model Driven Engineering (MDE) [1]. A model can refer full description about a system and can be defined by a metamodel which specifies the abstract syntax of models and the interrelationships between their elements. Therefore, a model must conform to this protocol of the metamodel similarly to how a grammar conforms for a programming language. Model Transformation is a mechanism for transforming one model into another model, based on some transformation rules or programs. Using different modelling languages, a model can be transformed into other types of software artefacts. A transformation engine performs the transformation using corresponding program which reads one source model writes a target model. Here source model conforms to source metamodel and target model conforms to a target metamodel. Despite the work of standardization committees, it is still easy for meaning to become lost or blurred in a model-oriented milieu. One relatively simple example of a model transformation is reverse-engineering a Unified Modelling Language (UML) class diagram from a piece of Java code. Yet this has proved to be a non-trivial problem, not least because of the imprecise semantics of both code and model [2]. The modelling approaches are used to describe the client's specification and translate to implementation using appropriate metamodels.

*Department of Computer Science, Maynooth University, Maynooth, Ireland; E-mails: Jagadeeswaran.Thangaraj@mu.ie

1.1. Motivation

We have studied the comparison of approaches towards the transformation from design to implementation to generate verified software system [3]. As a conclusion of this study, class artefacts, such as UML and Object Constraint Language (OCL) specifications, are a major component of object-oriented design and development. These artefacts

2213-2759/19 $58.00+.00

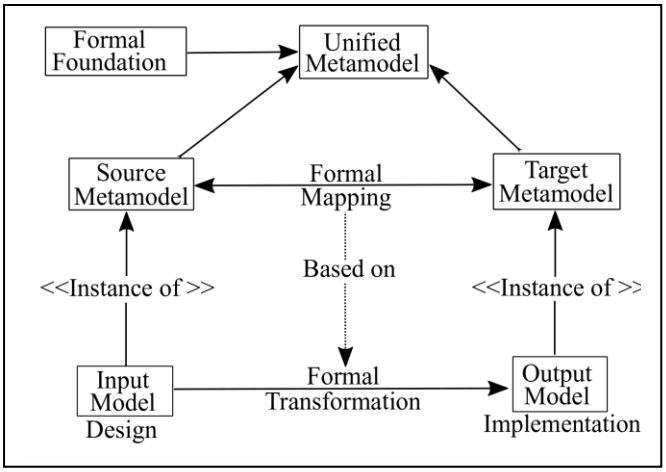

Fig. 1. Our approach in Model Transformation

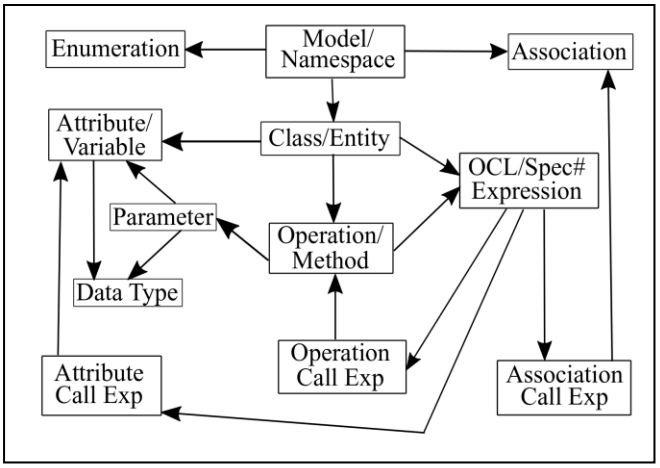

Fig. 2. Unified Metamodel of USE & Spec#

contain static class structure with specification constraints. Yet there are many barriers to describing and using them correctly across different models in a model-oriented environment. Finally, it needs to investigate the class artefacts at the intra- and inter-model level and investigate the portability of these artefacts between different phases. Therefore, if a system has a suitable methodology which supports re-usability of class artefacts, then it will be much efficient system in the transformation. Therefore, this paper establishing a new framework which to be act as an interface between design and implementation.

1.2. Formal Modelling

Modelling a software is common approach in different abstraction of formal software development, starting from high abstract specification to concrete implementation [4]. In formal software engineering, it can be a verifiable way of mapping these abstractions through formal transformation process. Formal transformation allows us to build up the system gradually and it offers a means by which proofs carried out in an abstract model of the system can be reused to prove properties about the concrete model.

In a formal system, a given theory is a collection of sets, categories and functors, satisfying certain conditions, that acts as a description of a logic. More precisely, a formal theory defines the vocabulary, syntactic constructs, and meaning of validity within a particular logic. Moreover, these provide a framework within which it is possible to formalise a given logic whilst providing access to an array of generic modularisation constructs and a framework for defining interoperability between them. This provides a means for formalising its semantics, modularisation constructs and interoperability between different formalisms in this framework. Therefore, this paper aims to establish a formal framework within which these artefacts can be shared between programs and models in a model-oriented environment in order to avoid loss of constructs. It will help in producing verified system in the target language according to the design specification and to support the re-usability of the model.

2. BACKGROUND

In Software development life cycle, model plays wider range of automation covering all phases and aspects. Due to the importance of meta-modelling, growing research efforts on meta-modelling have been made in the past few years. Recently, many researchers are working towards developing a unified representation to generate code from design when sharing core elements in related formalisms [5][6][7]. The Unified view is an emerging approach in formal methods, model-based software engineering, and programming languages when they share the core properties [8]. The modelling approaches are used to describe the client's specification and translate to implementation using appropriate metamodels as shown in Fig.1. We can use a unified metamodel which is the intersection of both source and target metamodels when translating between related formalism. We have introduced a unified metamodel which has unified properties of source and target metamodels of design to implementation. It will help to produce a common model of both source and target. Consistency and interoperability are major complexity in the model transformation activities. This paper aims to propose a formal approach to this unified metamodel in order to support reusability and interoperability of models, consistent transformation.

In this paper, we used the USE (The UML-based Specification Environment) specification to describe the program's specification at the design specification phase and Spec# at the implementation phase of software development process. The reasons behind these tools' selection have been explained in [9]. Also, we have implemented the ownership addition to UML in USE tool which can act as unified representation of Spec#. The unified properties of USE and Spec# has summarised in Table.1 based on our paper [10]. Here, Static structure and constraints of USE is mapped with Spec#' static properties and its specifications. USE/Spec# expression is used to construct USE constraints and Spec# specifications. The unified metamodel of USE and Spec# has constructed based on these unified properties as shown in Fig.2.

Unified Properties	**USE**	**Spec#**
Model Namespace	Model	Namespace
Class / Entity	Class	class
Enumeration	Enumeration	enum
Attribute/ variable	Attribute	variable
Operation/ Method	Operation	method
Association	Association	Association
Collections	Bag	List
Ownership Types	Ownership Types Rep Peer	Ownership Types Rep Peer
OCL/Spec# Expression	OCL Expression (Constraints)	Spec# Expression (Specifications)

Table. 1. Unified Properties of USE & Spec#

In the literature, we studied well about formal theory and the recent works based on these theories in transformation of specifications in different phases. Cengarle *et al.* [11] has developed a specification of UML class diagrams and OCL constraints using category theoretic Institutions which defines a semantics. Achouri [12] developed a work to possible transformation from UML activity diagrams to EVENT-B model using Institutional theory. Alexander Knapp [13] has defined the institutional framework for UML state machines in order to develop relationships with other UML sublanguages and diagram types. Fu *et al.* [14] has developed a semantic approach to check inconsistencies of OCL invariants by translating them into Ontology axioms. Farrell et al. [15] has introduced Institutions to Event-B for formal refinements. Bettaz *et al.* [16] has projected the concepts of intention to modelling for consistent transformation using Institutional theory. Falah *et al* [17] has developed a set theoretic approach to define formal semantics for UML class diagram. These formal theory-based approaches give a logic independent framework to support reusability and interoperability in different formalisms. We investigate the formal approach in model transformation for interoperability and reusability between different phases as shown in Fig.1.

2.1. Our proposal

This paper develops a formal framework to unified metamodel to solve the question which we raised in our previous work [2]. The framework allows software developers to reliably re-use models between different software representations in a model-oriented environment. We propose a formal approach based on these related works to coupe the relationship between USE and Spec#. Our proposed approach aims at first to give a semantic of ownership added USE via its representation as a formal ontology. Thus, with the defined semantic the transformation of USE model to a Spec# model can be semantically proven so that after the transformation, we will have the two models semantically equivalent. Our proposed approach doesn't tackle the problematic of heterogeneous specification environment like USE and Spec#. But we think that our work is a first attempt to support the use of a formal method like Spec# as target specification unlike other related works concentrate in establishing formal definition to either different phases [12][15][17] or transformation [13][14] as explained in section.2. However, even though recent work using formal semantics has included a specification of UML class diagrams and OCL constraints [11][16], neither approach provides a means whereby proofs and can be directly manipulated as models within the system. The use of sets assumes that each modelling representation has a "natural semantics", which forms the basis for the construction of models. However, our approach, a more theory-oriented development, provides a better context for the type of structures we intend to transform.

3. PROVIDING FORMAL SEMANTICS TO USE

We first aim to give formal presentation of USE specification with ownership support. We can view this formal presentation as a formal semantics of this USE specification using the algebraic presentation. This can act as the source formalism of a meta-level to possibly transform to Spec# models. Those notions are important for the specification, composition specifications, and heterogeneous specification approaches. In addition, we define a set of rules to generate Spec# model from the defined USE Specification semantics. With this formalised USE specifications, it makes easier translation into Spec# with translational correctness. In this section, we formalise the USE specifications such as USE class model, invariant specification with ownership types.

3.1 Semantics of USE Specification

3.1.1 USE model Syntax

USE model provides elements for the basic building blocks such as enumerations, classes, along with associations and constraints. This can be defined as follows using formal notation.

Definition 1 (*USE model*)

A USE model is defined as 3-tuple of the form

$USEMod = \langle ENUM, CLAS, CONSTR \rangle$

Here USE vocabulary *ENUM* is a 2-tuple of the form,

$ENUM = \langle Enum, ELit \rangle$

where,

Enum is a set of Enumerations, and

ELit is a set of enumeration literals.

As referring to the 'CarSystem' model in the example in section.5.1,

Enum = {Color, Type}

ELit = {silver, gold, type1, type2}.

Here USE vocabulary *CLAS* is a 6-tuple of the form,

$CLAS = \langle CLAA, AttrType, Atr, Opr, Assoc, OwnType \rangle$

where,

CLAA is a set of USE classes,
AttrType is a set of basic types of Attributes,
Assoc is a set of associations,
Atr is a set of USE attributes,
Opr is a set of operation names, and
OwnType is a set of Ownership types

As referred to the 'CarSystem' example in Section.5.1,

CLAA = {Car, Part, Radio, Sound,…}
AttrType = {Real, Integer, String, Boolean}
Assoc = {carPart, Spares, carRadio, radio, carSeat, spareSeat,…}
Atr = {speed, pid, pname, Frequency, …}
Opr = {addPart, usePart, volume, tune, move,}
OwnType = {Rep, Peer}

CONSTR is the set of OCL constraints as shown in the example in section 5.1. that provides some restrictions over the USE model using some OCL constraints. In our approach, we use only USE invariants to the classes which can express using the following expressions as mentioned in Unified metamodel in Fig.2.

exp ::= AttrCalExp | OperCalExp | AssocCalExp

AttrCalExp::= NavigationalPath | (NavigationalPath, ".", attributes) | (NavigationalPath, ("->", collectionOperation)*)| RelationalExp

OperCalExp:= NavigationalPath | (NavigationalPath, ".", operations) | RelationalExp

RelationalExp ::= AttrCalExp | AttrCalExp, relationOperator, AttrValue

relationOperator ::= ">" | "<" | ">=" | "<=" | "<>" | "="

NavigationalPath ::= (Variable) | NavigationalPath, ".",AssociationEnd

operations::="forAll"|"exists"|"select"| "reject"|"isEmpty"| "notEmpty"|"size"

Therefore, the USE invariant can be formed using these expressions which use the NavigationalPath which use at least one step in *exp*:

$$exp = ae_n.exp^j, \quad exp^j = ae_{n-1}.ae_{n-2}. \ldots ae_2.ae_1$$

For example,

context Car inv: self.speed <= 180

Here context refers USE keyword, follows by class name which is holding the invariant. Car is the class name which navigate to its attribute speed with condition. This can be expressed using AttrCalExp as following path.

NavigationPath->Attribute->relationOperator->Attrvalue

In our approach, we used three main expression to express the constraints over the USE specification such as AttrCalExp, OperCalExp and AssocCallExp.

3.1.2 Interpretation of USE vocabulary

Definition 2 (*Interpretation for USE Enumerations*)
The following structure is given for interpreting USE Enumerations vocabulary in formal semantics.

$$\mathbb{E} = < \mathcal{S}, .\mathbb{ENUM}, .\mathbb{ENUMLIT} >$$

Here $\mathcal{S}$ is a non-empty Interpretation set which consisting of all Enumeration instances $\mathcal{SE}$ and Enum Literal values $\mathcal{SEL}$ for that model. $\mathbb{E}$ is defined by group of functions, where,

Enumeration interpretation function maps to each USE Enumerations $e \in \mathbb{ENUM}$, $el \in \mathbb{ENUMLIT}$
a subset $(e)^{\mathbb{ENUM}} \subseteq \mathcal{SE}$,
e.g. $(Color)^{\mathbb{ENUM}} = \{silver, gold\}$,
$(Type)^{\mathbb{ENUM}} = \{type1, type2\}$

Definition 3 (*Interpretation for USE Classes*)
The following structure is given for interpreting USE class vocabulary in formal semantics.

$$\mathbb{C} = < \mathcal{S}, .\mathbb{CLAA}, .\mathbb{ATTR}, .\mathbb{ATTRTYPE}\ .\mathbb{OPER}, .\mathbb{ASSOC}, .\mathbb{OWNT} >$$

Here $\mathcal{S}$ is a non-empty Interpretation set which consisting of all instances $\mathcal{SC}$ and data values $\mathcal{SD}$ for that model. $\mathbb{C}$ is defined by group of functions, where,

Class interpretation function maps to each USE class further,
$\mathbb{CLAA}$ is the class interpretation function that maps to each UML class

$$c \in \mathbb{CLAA} \text{ a subset } (c)^{\mathbb{CLAA}} \subseteq \mathcal{SC}$$

$\mathbb{ATTRTYPE}$ is the datatype interpretation function that maps to each UML basic datatype

$$at \in \mathbb{ATTRTYPE} \text{ a subset } (at)^{\mathbb{ATTRTYPE}} \subseteq \mathcal{SD}.$$

$\mathbb{ATTR}$ is the attribute interpretation function that maps to each UML attribute

$attr \in \mathbb{ATTR}$
a subset $(attr)^{\mathbb{ATTR}} \subseteq \mathcal{SC} \times \mathcal{SD}$,
e.g. $(pid)^{\mathbb{ATTR}} = \{<part, Int>\}$.

$\mathbf{ASSOC}$ is the association end interpretation function that maps to each UML association end

$$as \in \mathbf{ASSOC} \text{ a subset } (as)^{\mathbf{ASSOC}} \subseteq \mathcal{SC} \times \mathcal{SC}$$

$$\text{e.g. } (Spare)^{\mathbf{ASSOC}} = \{<Car, Part>\}.$$

$\mathbf{OPR}$ is the operation interpretation function that maps to each UML operation

$$opr \in \mathbf{OPR} \text{ a subset } (opr)^{\mathbf{OPR}} \subseteq \mathcal{SC} \times \mathcal{SD}$$

$$\text{e.g. } (move)^{\mathbf{OPR}} = \{<Seat, Bool>\}.$$

$\mathbf{OWNT}$ is the ownership type interpretation function that maps to each ownership type literal

$$ot \in \mathbf{OWNLIT}.$$

3.1.3 Interpretation of USE constraints

A USE constraint is denoted using OCL constraints as follows

$$Inv\ (Contx, exp(v))$$

where $exp(v)$ is an OCL expression with the variable v of contextual classifier *Contx*. In transformation, the transformation function must transform corresponding expressions for each expression which form the invariant.

4. DEFINITION OF THE TRANSLATION

Definition 4 (*Definition of the translation approach trsl*)
Let *U* be a USE class model with USE constraints and S be a Spec# ontology. A translation approach from *U* to S is the function

$$trsl : U \rightarrow S \quad \text{or} \quad U \xrightarrow{trsl} S$$

trsl is sensible only if its correctness is established. *trsl* is correct in the sense that the translation from USE class model with constraints to Spec# preserves the meanings of *U* and *S*. So, we first have to provide the formal semantics of both *U* and *S* in order to define the correctness of *trsl*. It is analogous to the homomorphism in graph theory [18].

Definition 5 (*Correctness of trsl*)

Let U be the interpretation of USE vocabulary and S be the interpretation of Spec# vocabulary using unified metamodel. Correctness of *trsl i*s established according to the two criteria:

1. $U \rightarrow S$, i.e., interpretation of USE vocabulary is equal to the interpretation of Spec# vocabulary.

2. The interpretation of USE invariant is equal to interpretation of Spec# invariant. Otherwise, *trsl* is translated the USE invariant which is satisfied by the interpretation Spec#.

```
model CarSystem
enum Color{silver, gold}        enum Type{type1, type2}

class Car                        class Radio
attributes                       attributes
speed: Integer;                   Frequency: Integer;
operations                       operations
addPart();                        end
usePart();
end

class Part                      class Seat
attributes                       attributes
pid:Integer;                      --
pname:String                    operations
operations                       move():Boolean;
end                               end

association Spare between
Car[0..1] role carPart --Ownership [Rep]
Part[1..*] role spares --Ownership [Rep]
end
association Music between
Car[0..1] role carRadio --Ownership [Rep]
Radio[0..1] role radio --Ownership [Rep]
end
association seats between
Car[0..1] role carSeat --Ownership [Rep]
Seat[1..*] role spareseat --Ownership [Peer]
end

constraints
context Car
inv: self.speed <= 180
```

Fig.3. CarSystem Model in USE

5. RESULT

5.1. An example of USE Specification

We apply our approach with reference to a practical example of USE specification to a USE Class model in Fig.3. We propose this USE class model to illustrate the transformation of USE model to Spec# while this is not adequate to represent a real world. This *CarSystem* model contains different classes and they have their own attributes and operations as shown in Fig.3. Each class connected through associations with role names, multiplicities and ownership types. It may contain some constraints to depict over the classes which must be translated to Spec#. We took the constraint supported by three major expressions as shown in Unified metamodel in Fig.2.

Many researches have been developed in translating design specifications to implementation using model transformations. Most of them transformed OCL to others specification, otherwise, UML to programming code. In our work, we planned to transform USE, UML and OCL, to the code implementation with verification support in Spec#. We

Interpretation functions	Properties	Set	Subset
ENUM	Enum	$e \in$ ENUM,	$(e)^{ENUM} \subseteq SE$
ENUMLIT	Enum Literals	$el \in$ ENUMLIT	$(el)^{ENUMLIT} \subseteq SEL$
CLAA,	Class	$c \in CLAA$	$(c)^{CLAA} \subseteq SC$
ATTR	Attribute	$attr \in ATTR$	$(attr)^{ATTR} \subseteq SC \times SD$
ATTRTYPE	Attribute Types	$at \in ATTRTYPE$	$(at)^{ATTRTYPE} \subseteq SD.$
OPER	Operations	$opr \in OPR$	$(opr)^{OPR} \subseteq SC \times SD$
ASSOC	Associations	$as \in ASSOC$	$(as)^{ASSOC} \subseteq SC \times SC$
OWNT	Ownership Types	$ot \in$ OWNT	$(ot)^{OWNT} \subseteq SOL$

Table. 2. Interpretation functions

have investigated an approach to represent a platform independent model for translating USE specifications into Spec# specifications. Using the proposed approach, we can translate a large class of USE design with constraints, using the three expressions: AttrCalExp, OperCalExp and AssocCalExp. We have used the theoretic representation of related works of authors Fu et al. [14] and Falah [17]. Fu et al. translates OCL invariants into OWL 2 DL axioms to check inconsistencies. Falah et al provides clear semantics to UML Class diagrams. We have combined their both approaches to define a unified metamodel for consistent transformation. We have provided the semantics to unified metamodel to define a USE specification model.

5.1 Discussion

A model which conforms the unified metamodel is common for all related meta models. So, we can transform one specification of design to the code of implementation by applying the transformation rules. Also, we can multi directionally transform one language to another language using the corresponding transformation rules [19]. As shown in Table.2., this paper provides the formal semantics to each property of the unified metamodel in order to generate model instances which support reusability and multidirectional transformation. It provides the set of classes, enumerations, datatypes and ownership types and their subset to represent other properties. This formal framework support platform independent designs to the elements in model-oriented environment. It helps to implement a tool in modelling framework. The next step is to develop a tool to implement the objective and verify the consistent transformation.

6. CONCLUSION

This paper has presented the theoretic semantics to Unified metamodel in terms of USE model to define an independent model for consistent transformation. As a conclusion, the theoretic framework provides the possibilities of all required class artefacts in terms of formal semantics. Our next aim is to build a tool using the theory and validate these artefacts by transforming them into different formalisms. Therefore, our next job is to build the tool to implement these artefacts in a model-oriented environment and transform consistently. The new tool should help in producing verified system in the target language according to the design specification and to support the re-usability of the specification. Also, it should be able to produce specifications in different languages enabling the implementation by expertise in that language. Therefore, the new framework will be based on the sustained development of different formalism to support interoperability between these specifications. If the generated system supports the verification with all required properties, code developer needs less e orts to produce correct software system.

REFERENCES

1. Hiroaki Shimba, Kentrao Hanada, Kozo Okano and Shinji Kusumoto: “Bidirectional Translation between OCL and JML for Round-Trip Engineering”, 20th Asia-Pacific Software Engineering Conference (APSEC), Bangkok, pp. 49-54. DOI: 10.1109/APSEC.2013.111 (2013).

2. Jagadeeswaran.Thangaraj, Senthilkumaran.Ulaganathan: “A Formal Framework to Unified Metamodel for Consistent Transformation”, In the PhD Symposium at iFM’18 on Formal Methods: Algorithms, Tools and Applications (PhD-iFM’18) Research report 483, ISBN 978-82-7368-435-6 ISSN 0806-3036 (2018)

3. Jagadeeswaran Thangaraj, Senthilkumaran Ulaganathan: “A Comparative Study on Transformation of UML/OCL to Other Specifications”, Recent Patents on Computer Science ISSN: 2213-2759 12: 1. doi.org/10.2174/2213275912666190129121059 (2019)

4. Marie Farrell: “Event-B in the Institutional Framework: Defining a Semantics, Modularisation Constructs and Interoperability for a Specification Language”. PhD Thesis, Department of Computer Science, Maynooth University. (2017)

5. S. Sepúlveda, C. Cares and C. Cachero: “Towards a unified feature metamodel: A systematic comparison of feature languages”, In 7th Iberian Conference on Information Systems and Technologies (CISTI 2012), Madrid, 2012, pp.1-7, (2012).

6. Amjad Fayoumi, Evangelia Kavakli, and Pericles Loucopoulos: “Towards a Unified Meta-Model for Goal Oriented Modelling”, In European, Mediterranean & Middle Eastern Conference on Information Systems (EMCIS), pp. 1-10. ISBN 9789606897085 (2015).

7. Huseyin Ergin, Eugene Syriani: "A Unified Template for Model Transformation Design Patterns", In Patterns in Model Engineering, co-located with Software Technologies: Applications and Foundations STAF (2015).

8. Manfred Broy, Klaus Havelund and Rahul Kumar: "Towards a Unified View of Modeling and Programming." Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications. ISoLA 2016. Lecture Notes in Computer Science, vol 9953. Springer, Cham doi.org/10.1007/978-3-319-47169-3_17 (2016).

9. Jagadeeswaran.Thangaraj, Senthilkumaran.Ulaganathan: "Introducing Ownership Types to Specification Design". In International Journal of Scientific & Engineering Research Volume 8, Issue 8, August- 2017 pp: 843-848. ISSN:2229- 5518, DOI:https://dx.doi.org/ 10.14299/ijser.2017.08, (2017).

10. Jagadeeswaran.Thangaraj, Senthilkumaran.Ulaganathan: "Mapping USE Specifications into Spec#". In: Software Technologies: Applications and Foundations. STAF 2017. Lecture Notes in Computer Science, vol 10748. Springer, Cham, doi.org/10.1007/978-3-319-74730-9_29 (2018)

11. María Victoria Cengarle, and Alexander Knapp: "An institution for UML 2.0 static structures". Technical Report TUM-I0807, Technische Universität München (2008).

12. Aymen Achouri: "Towards a Semantic for UML Activity Diagram Based on Institution Theory for It's Transformation to Event-B Model", In 4th International conference on Computer Science & Information Technology (CS & IT) – CSCP 4, pp. 123–138, DOI: 10.5121/csit.2014.4211 (2014).

13. Alexander Knapp, María Victoria Cengarle: "Institutions for OCL-Like Expression Languages". In Software, Services, and Systems. Lecture Notes in Computer Science, vol 8950. Springer, Cham (2015)

14. Chunlei Fu, Dan Yang, Xiaohong Zhang, Haibo Hu: "An approach to translating OCL invariants into OWL 2 DL axioms for checking inconsistency". In Automated Software Engineering Journal. Vol 24(2), pp. 295-339, issn:1573-753, DOI:10.1007/s10515-017-0210-9. (2017).

15. Marie Farrell, Rosemary Monahan, and James F. Power: "An Institution for Event-B". in: Recent Trends in Algebraic Development Techniques. WADT 2016. Vol. 10644. LNCS, pp. 104–119 (2017).

16. Nabil Messaoudi, Allaoua Chaoui, Mohamed Bettaz: "An Approach to UML Consistency Checking Based on Compositional Semantics". International Journal of Embedded and Real-Time Communication Systems, Vol.8(2), pp.1-23 (2017)

17. Bouchaib Falah, Mohammed Akour, Issar Arab & Mhanna, Yassine: "An Attempt Towards a Formalizing UML Class Diagram Semantics". In New Trends in Information Technology (NTIT 2017), The University of Jordan, Amman, Jordan, pp. 21-27 (2018).

18. Yilun Shang: "Limit of a nonpreferential attachment multitype network model". International Journal of Modern Physics B. Vol:31(5) id.1750026. DOI:10.1142/S0217979217500266. (2017).

19. Jagadeeswaran.Thangaraj, Senthilkumaran.Ulaganathan, Model Reusability and Multidirectional Transformation using Unified Metamodel'. In 2018 IEEE Distributed Computing, VLSI, Electrical Circuits and Robotics, Mangalore, India (IEEE-DISCOVER 2018).