Explicitly Modelling the Type/Instance Relation

Yentl Van Tendeloo[1] and Hans Vangheluwe[1,2,3]

[1] University of Antwerp, Belgium
Yentl.VanTendeloo,Hans.Vangheluwe@uantwerpen.be
[2] Flanders Make vzw, Belgium
[3] McGill University, Montréal, Canada
hv@cs.mcgill.ca

Abstract. The various meta-modelling tools in existence today all have differences in their conformance relations, either intentional or accidental. This results in incompatibilities between tools, where a model cannot simply be exchanged as-is: the meta-meta-model and semantics likely differ. Current tools are inflexible in this regard, making their models grafted on the tool implementation. In this paper, we distinguish between syntactical and semantical differences between tools, both resulting in non-exchangable models. We propose to explicitly model the meta-meta-model (addressing syntactical differences) and its semantics (addressing semantical differences). This allows meta-meta-models and semantics to be added and manipulated at runtime, making our approach flexible for new tools as well. Models and languages from different tools can then be meaningfully stored in a single tool, retaining syntax and semantics. We provide a prototype implementation: the Modelverse.

1 Introduction

A plethora of meta-modelling tools currently exist, many with their own meta-circular level and associated conformance semantics. Due to these intentional or accidental differences, inconsistencies between tools arise: all models, including languages, become grafted on the tool's implementation. With the growing importance of tool interoperability, for example in collaboration and model repositories, this might turn into a problem: while the model can be exchanged, the inherent tool semantics cannot.

These limitations are caused by variations between tools, in which we distinguish two types: syntactical and semantical. Syntactical variations result in non-exchangable models, and semantical variations cause unexpected behaviour with successfully exchanged models. A simple example is multiple inheritance. Syntactically, some tools do not support this, making them unable to receive models from tools which do support multiple inheritance. Semantically, some tools handle the resolution order of multiple inheritance differently, thereby altering the set of allowed instances.

In this paper, we address this problem by explicitly modelling the syntax and semantics, usually built into the tool. Apart from documentation, new syntax

and semantics can be loaded on-demand, without altering the tool itself. As such, a single tool is able to store and operate *semantically meaningful* models of different tools, given that explicit models are present. Models now become grafted on another model, which can just as well be exchanged, instead of being grafted on the tool implementation. Additionally, the created models can be used as documentation of the syntax and semantics of the tool.

The remainder of this paper is organized as follows. Section 2 specifies the two types of variations and provides examples. Section 3 presents our explicitly modelled approach. Section 4 describes our implementation. Section 5 presents related work. Section 6 concludes the paper and presents future work.

2 Types of Variation

Semantically meaningful model exchange is hindered by tool incompatibilities. Even similar tools, by the same authors, are often incompatible: AToMPM [19] and AToM^3 [12]; WebGME [14] and GME [13]; DPF [8] and WebDPF [15]. Even different versions of the same tool can be incompatible, either intentionally [3], or accidentally due to minor implementation changes or bugs.

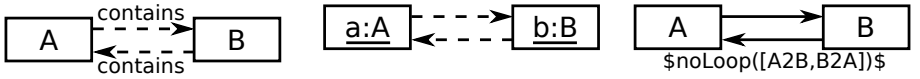
To concretize the problem, we focus on two tools throughout the remainder of the paper: AToMPM [19] and Metadepth [9]. For both tools, we describe some (hardcoded) differences to illustrate the problem. We present minimal example languages and models for both syntactical and semantical differences.

We consider two types of variations: syntactical and semantical variations.

2.1 Syntactical Variations

First are syntactical variations, caused by a different abstract syntax of the meta-language. Such changes can automatically be detected, as a model would rely on unknown constructs. We show two examples: one with a feature of AToMPM that Metadepth does not support, and one that is the other way around.

The first example language, in Figure 1a, uses a specific kind of association: the containment relation, as supported by AToMPM. It resembles an ordinary association, but indicates that the source element is a container for the target element. Its primary use is for visual representation, though it is also used as implicit constraint: containment cycles are not allowed. Instances of class A can contain instances of the class B, and the other way around. Figure 1b shows an example instance of this language, where a contains b, but also the other way around. Conceptually, this does not make any sense. With a containment relation, this is automatically flagged as an error and the model does not conform. In Metadepth, which does not support a containment relation, this same language cannot be loaded; the association type "containment" is unknown. As such, the model cannot be exchanged either, as it depends on the language. It is possible to mimic the containment relation in Metadepth by defining the containment relation as a normal association, which has an additional constraint that does not allow loops. A semantically equivalent meta-model is shown in Figure 1c.



(a) AToMPM meta-model. (b) Invalid instance. (c) Metadepth meta-model.

Fig. 1: First example language: use of containment links.

The second example language, in Figure 2a, uses multiplicities on a class, as supported by Metadepth. The lower and upper cardinality is defined as an integer attribute on the class. Its primary use is to restrict the number of instances of this specific type: the number of instances must be within this range. The class A requires that there are exactly two instances of A in every model conforming to it. Figure 2b shows an example instance, where only one instance of A is present. With the class multiplicities, this is automatically flagged as an error and the model does not conform. In AToMPM, which does not support class multiplicities, this same language cannot be loaded: the attribute "multiplicity" is unknown. It is possible to mimic multiplicities in AToMPM by defining a global constraint, which checks the number of instances of A. A semantically equivalent meta-model is shown in Figure 2c.

(a) Metadepth meta-model. (b) Invalid instance. (c) AToMPM meta-model.

Fig. 2: Second example language: use of node multiplicities.

In both example languages, the tools are equivalent in their expressiveness (i.e., they can be used to express the same language), but the language must be represented differently. As such, languages, and therefore models, cannot be easily exchanged without a conversion at the abstract syntax level.

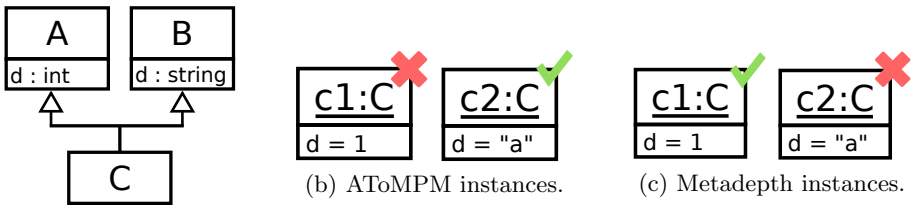
2.2 Semantical Variations

Second are semantical variations, caused by a difference in the implementation of the conformance check, which provides the semantics for the abstract syntax of the meta-language. Clearly, just calling an association "containment" or an attribute "multiplicity" does not automatically give it the correct semantics: it needs to be defined somewhere. Semantic differences are indetectable when models are exchanged, as they structurally conform.

Note that we consider the semantics of the meta-modelling language (i.e., what does a given meta-model mean), and not the semantics of the modelling language (i.e., what does a given model mean). The former is mostly hardcoded

in the tool, whereas the latter is domain-specific and implemented using, for example, model transformations.

The third and final example language, in Figure 3a, uses multiple inheritance, as supported by both AToMPM and Metadepth. An example of such a language is shown in Figure 3a, where the class C inherits from both A and B. Both A and B define the same attribute but with different types. It is unclear which of the two is selected for C, which inherits from both. The semantics attached to multiple inheritance, responsible for the choice, is hardcoded in both tools and left undocumented. Only experimentation is therefore possible to figure out what it means, resulting in Figure 3b for AToMPM and Figure 3c for Metadepth. As the set of conforming instances differs, for the same language, both tools attach different semantics to the language. AToMPM seems to resolve the earliest created inheritance link, whereas Metadepth seems to lexicographically sort the class names and picks the first match.



(a) Metamodel.

(b) AToMPM instances.

(c) Metadepth instances.

Fig. 3: Third example language: use of multiple inheritance.

While the example difference here is likely intentional, many other differences exist that are likely accidental (e.g., bugs or ommissions). For example, AToMPM does not check the type of attributes, and Metadepth cannot connect edges when inheritance is involved, nor can it have attributes with specific keyworded names (e.g., "id"). Notwithstanding the source of the difference, the semantic differences make model exchange meaningless. And when the tool semantics is altered (e.g., an intentional change, a bugfix, or a newly introduced bug), it is possible that a previously conforming model suddenly becomes invalid, or that a previously invalid model suddenly becomes valid.

2.3 Problem Scope

The previously introduced problems are often irrelevant for users of a single tool. As such, we do not propose to alter existing (meta-)modelling tools, though it might be useful to have sufficient documentation. The true problem lies elsewhere, with tools that explicitly have to communicate with many different tools: model repositories.

The primary purpose of model repositories is model storage and exchange. It makes sense that they want to maximize the set of supported tools, thereby maximizing the available models. Model repositories, however, have to understand

the models they are managing, as otherwise they would be reduced to a mere file server. When working with models from different tools, and possibly exchanging them between these tools, it becomes important to take these syntactical and semantical variations into account.

3 Explicit Type/Instance Relations

Varying abstract syntax and semantics at the meta-language level were identified as the root of the problem. Current tools acknowledge that an explicit meta-model is required to create a flexible modelling tool, in which the language can be altered. They do not, however, take this one level up the meta-modelling hierarchy: the language used to create new languages, the meta-language, is hardcoded in the tool. Being hardcoded, the problem becomes even worse: they are not flexible either, as the hardcoded aspects cannot be altered in any way. For this reason, we propose to explicitly model both the meta-language's abstract syntax and semantics, and make this fully flexible at runtime.

Some aspects of current tools are already modelled explicitly. We distinguish three layers, as commonly agreed upon: *M1* (model level), *M2* (language level, or meta-model), and *M3* (meta-language level, or meta-meta-model). For Metadepth, both *M1* (M_1^{AS}) and *M2* (M_2^{AS}, M_2^{SEM}) are explicitly modelled. In AToMPM, the syntax of *M3* (M_3^{AS}) is additionally explicitly modelled. Both tools hardcode the semantics of *M3* (M_3^{SEM}). It is this aspect of the tool that we will also model explicitly in our approach.

As both M_3^{AS} and M_3^{SEM} are explicitly modelled in our approach, it becomes possible to (1) alter them at runtime (e.g., optimizations, refactorings); (2) create and use new ones at runtime (e.g., support a new tool, bugfixes); and (3) have multiple of them simultaneously (e.g., models from different tools loaded in a single tool).

3.1 Meta-meta-model (M_3^{AS})

The meta-meta-model M_3^{AS} defines the concepts that can be used when defining a new language, or meta-model. It has two primary purposes. First, it can serve as documentation for language engineers: what is the name of attributes, what constraints can be added, whether multiple inheritance is supported, and so on. Second, it is required for several operations that need an explicit meta-model. For example RAMification [7], used to create a new language to express model transformation rules by Relaxing, Augmenting, and Modifying the existing language. Differences in M_3^{AS} lead to syntactical differences between tools, which can be automatically detected by comparing two M_3^{AS} models.

The M_3^{AS} of AToMPM is shown in Figure 4, modelled explicitly using Entity-Relation Diagrams. It shows the various attributes that can be set on a class, such as "attributes" to define new attributes, and "name". Perhaps surprisingly, attributes have no dedicated entity, in contrast to other approaches, such as EMF.

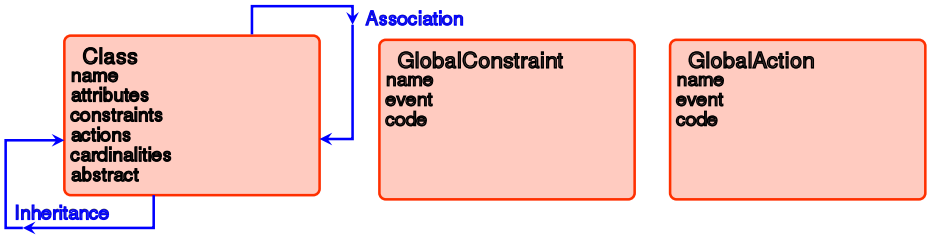


Fig. 4: AToMPM's M_3^{AS}, taken directly from AToMPM itself.

3.2 Meta-Language Semantics (M_3^{SEM})

The meta-language semantics M_3^{SEM} defines the semantics of concepts defined in M_3^{AS}. It takes a model and its meta-model as input, and determines whether the model conforms to the meta-model. Its primary purpose is in determining whether a model is valid with respect to a given language specification (i.e., check conformance). Differences in M_3^{SEM} lead to semantical differences between the tools, which are difficult to detect automatically. Indeed, we would need to verify if two models for M_3^{SEM} behave exactly the same in every possible context.

Due to space restrictions, we only show a snippet of the conformance algorithm of Metadepth, pertaining to the multiplicity checks of classes in Algorithm 1. This pseudo-code can be modelled in an explicitly modelled action language.

Algorithm 1 M_3^{SEM} snippet for the cardinality check.

```
for all class ∈ allInstances(M3, Class) do
    assert allInstances(M2, class) ≥ class.lower_cardinality
    assert allInstances(M2, class) ≤ class.upper_cardinality
end for
```

4 Implementation

As a proof of concept, we implemented this approach in our prototype tool: the Modelverse[1] [21, 23]. We describe the basics of the Modelverse, and present our implementation of this approach in our prototype tool. We show that our approach can indeed handle different types of M_3^{AS} and M_3^{SEM} in the same tool, making it compatible with multiple tools. We do not consider technical challenges, such as how to export a model and load it again in our tool.

The Modelverse, in essence, consists of an explicitly modelled action language interpreter, combined with an action language implementation of all metamodelling behaviour. As such, not only the conformance aspects are modelled

[1] https://msdl.uantwerpen.be/git/yentl/modelverse.git

explicitly, but all other aspects as well, such as model instantiation, model transformation, and model management in general. The action language semantics has been explicitly modelled as well [22]. When defining a new conformance relation, this is therefore a model in the Modelverse and an extension to the Modelverse at the same time.

The interrelations between all models, such as conformance relations, are also modelled explicitly. For each conformance relation, the link is specified by the source model (instance), target model (meta-model), and conformance semantics. As such, a model can conform to the same meta-model through multiple conformance semantics, or to different meta-models using the same conformance semantics, or to different meta-models using different conformance semantics.

Using this explicitly modelled approach, the Modelverse is able to offer the same results for the conformance checks as either AToMPM or Metadepth, or any other tool, for that matter. An overview of how these explicitly modelled interrelations are stored, is shown in Figure 5, which itself represents a model in the Modelverse. While *L1* and *L2* can only be created in AToMPM and Metadepth, respectively, *L3* is a valid language in both. Nonetheless, both tools attach a different semantics to *L3*, as seen in the two different instances: *M3* and *M4*. Both conformance semantics are also explicitly modelled, as shown at the right. These are again typed by something, in this case using a conformance relation defined specifically for the Modelverse, though also explicitly modelled. This could just as well have been any other previously defined semantics.

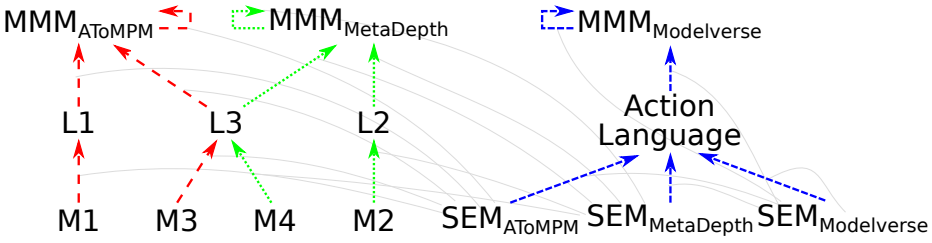


Fig. 5: Overview of languages, models, and relations in the Modelverse.

5 Related Work

The conformance relation plays a crucial role in Model Driven Development (MDE) [20, 1]. But while it has often been studied, it has remained hardcoded in various tools. Research up to now has mostly focussed on defining different levels of conformance [6] and new types of conformance, such as relaxed [17] or partial [18]. Nonetheless, they are hardcoded and partially inflexible as well: tools cannot be extended with additional conformance relations, nor can existing conformance relations be inspected or manipulated.

Updates to the hardcoded syntax and semantics, such as UML, have resulted in (often unnecessary) breakage of conformance [3]. As such, old models cannot

be loaded in newer versions of the same tool, if tools update their implementation of the UML. But whereas a model might still load in a newer version of the UML, nothing guarantees that the same semantics are used in its evaluation.

In the multi-level modelling community, many aspects of conformance are still being investigated, specifically for newly introduced physical attributes. For example, the "potency" attribute has various meanings, though in all tools it is typed as a natural number [16]. All identified semantics were implemented side by side in Metadepth, and users could toggle between them on a case-by-case basis. While this certainly gives great freedom, it creates yet another specification of the semantics: a union of existing ones. Future definitions might yet again require changes to this specification, potentially breaking conformance.

When models are exchanged, the first thoughts are of the technological problems: how to transfer the data from one tool to another. Various serialization formats were conceived for this problem, such as XMI and JSON. Nonetheless, they limit themselves to transfering data only, not the actual model. Essentially, M_2^{AS} is exchanged, but M_3^{AS} and M_3^{SEM} are not. As such, models can be exchanged, assuming that both tools implement the same M_3^{AS} and M_3^{SEM}. Should they differ, the exchanged data becomes semantically meaningless.

This brings us to model repositories, which, using this approach, often resort to semantically meaningless model exchange. For example, ReMoDD [5], being as general as possible, sacrifices model semantics: uploaded models have only marginally more semantics than arbitrary files. Advanced operations, which rely on the semantics of these models, are not supported. Another solution, as taken by MDEForge [2], is to restrict exchanged models to its own M_3^{AS} and M_3^{SEM}. This allows them to actually use the models, for example for model transformations [4], though they do so by limiting the set of supported tools.

A posteriori typing [11] has been proposed as a way to have multiple types for a single model. But whereas our approach does not consider any conformance relation as special, a posteriori typing starts from a special relation: the constructive type. The constructive type is the type used to instantiate the model, and cannot be changed. All other types, discovered on-the-fly, are completely flexible. Even though multiple meta-models can be found for a single model, all conformance relations must use the same M_3^{AS} and M_3^{SEM}. As such, only M_2^{AS} can change, and not M_3^{AS} nor M_3^{SEM}, though this provides sufficient knowledge to reuse operations between different meta-models. Concepts [10] serve a similar purpose, but also with similar restrictions: there is only freedom at M_2^{AS}.

6 Conclusions and Future Work

We have presented the problem of combining multiple (meta-)modelling tools, each with their own meta-meta-model syntax and semantics. This problem is particularly relevant in the context of model repositories, where the collaboration between multiple tools is the primary requirement.

We proposed to explicitly model all aspects of the conformance relation: its abstract syntax and semantics. Using our approach, we have shown that we can achieve full flexibility. A single tool was able to store, in a semantically meaningful way, models with the same semantics as other tools. Furthermore, a single model can conform using multiple such semantics, potentially to different abstract syntax definitions as well.

A prototype implementation of this approach was presented in the Modelverse, our explicitly modelled meta-modelling environment. The Modelverse was able to handle multiple models, meta-models, and also meta-meta-models, all with their own associated semantics. At runtime, new conformance semantics models could be uploaded, and existing ones could be modified.

In future work, we plan to investigate the influence of the conformance relation on other modelling operations as well, such as the *allInstances* operation. Also, we intend to create a variability model of all differences encountered in various M_3^{AS} and M_3^{SEM} implementations, of which each tool implementation is (theoretically) a specific configuration. Finally, this approach is ideally suited for semantically meaningfull model exchange between two existing tools: the unifying tool, using our approach, would be able to perform the translation automatically.

Acknowledgments. This work was partly funded by a PhD fellowships from the Research Foundation - Flanders (FWO). This research was partially supported by Flanders Make vzw, the strategic research centre for the manufacturing industry.

References

1. Atkinson, C., Kühne, T.: Concepts for comparing modeling tool architectures. In: Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MoDELS). pp. 398–413 (2005)
2. Basciani, F., Di Rocco, J., Di Ruscio, D., Di Salle, A., Iovino, L., Pierantonio, A.: MDEForge: an extensible web-based modeling platform. In: Proceedings of the Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE). pp. 66 – 75 (2014)
3. Degueule, T., Combemale, B., Blouin, A., Barais, O., Jézéquel, J.M.: Safe model polymorphism for flexible modeling. Computer Languages, Systems & Structures 49, 176–195 (2017)
4. Di Rucco, J., Di Ruscio, D., Pierantoini, A., Sánchez Cuadrado, J., de Lara, J., Guerra, E.: Using ATL transformation services in the MDEForge collaborative modeling platform. In: Proceedings of the International Conference on Model Transformation (ICMT) (2016)
5. France, R., Biemand, J., Cheng, B.H.C.: Repository for model driven development. In: Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MoDELS). pp. 311 – 317 (2006)
6. Kennel, B.: A Unified Framework for Multi-Level Modeling. Ph.D. thesis, University of Mannheim (2012)
7. Kühne, T., Mezei, G., Syriani, E., Vangheluwe, H., Wimmer, M.: Explicit transformation modeling. In: Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MoDELS). pp. 240 – 255 (2009)

8. Lamo, Y., Wang, X., Mantz, F., MacCaull, W., Rutle, A.: DPF Workbench: A diagrammatic multi-layer domain specific (meta-) modelling environment. Computer and Information Science, Studies in Computational Intelligence 429, 37–52 (2012)
9. de Lara, J., Guerra, E.: Deep meta-modelling with MetaDepth. In: Proceedings of the TOOLS EUROPE Conference. pp. 1 – 20 (2010)
10. de Lara, J., Guerra, E.: Generic meta-modelling with concepts, templates and mixin layers. In: Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MoDELS). pp. 16 – 30 (2010)
11. de Lara, J., Guerra, E., Sánchez Cuadrado, J.: A-posteriori typing for model-driven engineering. In: Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MoDELS). pp. 156 – 165 (2015)
12. de Lara, J., Vangheluwe, H.: AToM3: A tool for multi-formalism and meta-modelling. In: International Conference on Fundamental Approaches to Software Engineering. pp. 174–188 (2002)
13. Ledeczi, A., Maroti, M., Bakay, A., Karsai, G.: The Generic Modeling Environment. In: Proceedings of International Symposium on Intelligent Signal Processing (WISP) (2001)
14. Maróti, M., Kecskés, T., Kereskényi, R., Broll, B., Völgyesi, P., Jurácz, L., Levendovszky, T., Lédeczi, A.: Next generation (meta)modeling: web- and cloud-based collaborative tool infrastructure. In: Proceedings of the Workshop on Multi-Paradigm Modeling (MPM). pp. 41 – 60 (2014)
15. Rabbi, F., Lamo, Y., Yu, I.C., Kristensen, L.M.: WebDPF: A web-based meta-modelling and model transformation environment. In: Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development. pp. 87–98 (2016)
16. Rossini, A., de Lara, J., Guerra, E., Rutle, A., Wolter, U.: A formalisation of deep metamodelling. Formal Aspects of Computing (Springer) 26(6), 1115–1152 (2014)
17. Salay, R., Chechik, M.: Supporting agility in MDE through modeling language relaxation. In: Proceedings of the Workshop on Extreme Modeling. pp. 21 – 30 (2013)
18. Sottet, J.S., Biri, N.: JSMF: a Javascript flexible modelling framework. In: Proceedings of the 2nd Workshop on Flexible Model Driven Engineering. pp. 42–51 (2016)
19. Syriani, E., Vangheluwe, H., Mannadiar, R., Hansen, C., Van Mierlo, S., Ergin, H.: AToMPM: A web-based modeling environment. In: Joint Proceedings of MODELS'13 Invited Talks, Demonstration Session, Poster Session, and ACM Student Research Competition. pp. 21–25 (2013)
20. Theisz, Z., Mezei, G.: An algebraic instantiation technique illustrated by multilevel design patterns. In: Proceedings of the 2nd International Workshop on Multi-Level Modelling. pp. 53–62 (2015)
21. Van Tendeloo, Y.: Foundations of a multi-paradigm modelling tool. In: Proceedings of the ACM Student Research Competition at MODELS 2015 co-located with the ACM/IEEE 18th International Conference MODELS 2015. pp. 52 – 57 (2015)
22. Van Tendeloo, Y., Barroca, B., Van Mierlo, S., Vangheluwe, H.: Modelverse specification. Tech. rep., University of Antwerp (2017)
23. Van Tendeloo, Y., Vangheluwe, H.: The Modelverse: A tool for multi-paradigm modelling and simulation. In: Proceedings of the Winter Simulation Conference (2017), (accepted)