Doctoral School MASTIC Mathématiques et STIC	**Specialty:** Software **Laboratory:** LS2N **Team:** NaoMod,Velo

Towards Model-based Performance Evaluation of Manufacturing Systems

Hiba AJABRI
2nd year PhD student, thesis started on January 2023
hiba.ajabri@ls2n.fr

Abstract: Early performance evaluation is essential when designing systems in order to enable decision-making. This requires both a way to simulate the system in an early state of design and a set of relevant Key Performance Indicators (KPIs). Model-Driven Engineering and Domain-Specific Languages (DSLs) are well suited for this endeavor. However, KPIs are commonly tailored to a particular system, and therefore need to be redefined for each of its variation. In light of these problems, our research work deals with enabling performance evaluation facilities for models conforming to an executable DSL. To deal with early performance evaluation, we built an executable domain-specific language called *Simple Manufacturing System (SMS xDSL)*, meant to define and simulate different simple manufacturing systems. This article exhibits the construction of the SMS xDSL and a solution that has been elaborated. We outline a future projection and the strategy for establishing more generic solutions by capitalizing on specific ones, thus performance evaluation facilities can be applied to a wide range of executable DSLs.
Keywords: *Domain-Specific Language, Model Execution, Key Performance Indicators, Reconfigurable Manufacturing System.*

1 Introduction

Systems have a great effectiveness and efficiency when they undergo a performance evaluation earlier in the design phase. This requires modeling the behavior of the systems and simulating them. Model-driven engineering (MDE) and Domain-Specific Languages (DSLs) are well suited for this endeavor.

My thesis is undertaken in the RODIC [1] project grounded in the domain of industrial engineering and manufacturing systems [2, 4]. It is a prominent domain that intensively necessitates early performance appraisals, as evaluating the efficiency of systems after their implementation through a trial-and-error strategy is costly and time-consuming. The necessity of a quick and an early performance feedback is even more crucial when dealing with specific industrial systems as Reconfigurable Manufacturing Systems (RMS) [5]. RMS are manufacturing systems (e.g., production lines) designed for rapid change in order to quickly respond to sudden market changes: when an RMS must be reconfigured, it is indispensable to evaluate and compare the performance of the multiple configurations of the same RMS to arbitrate the choice between them.

In order to assess the performance of production lines, several questions need to be answered (e.g., how much a production line can produce over a specified time period?, are machines always in use, or are some machines underused?). Key Performance Indicators (KPIs) are introduced as a set of metrics that are used to answer those questions [3].

There are a number of issues that need to be addressed in order to evaluate the performance of systems. We need an appropriate model of the system (i.e., with related KPIs), but one major problem pertains to the nature of KPIs, which are typically tailored to a particular domain. Given this nature, KPIs need to be redefined for each new considered context. However, defining new KPIs or adapting existing KPIs can quickly become a cumbersome and error-prone task, especially in contingency situations when decisions have to be made rapidly and efficiently. Our research targets enabling performance evaluation for models not only for RMS but also for more general industrial systems. In the present paper, Section 1 examines how executable DSL techniques have been used to model and simulate industrial systems. Section 2 outlines a solution that has been implemented around these techniques, and we demonstrate how concerns, related to the customization aspect of KPIs, have

[1] Rapid recOnfiguration of manufacturing systems: a model-baseD software engineering and human Interaction Coupled approach (RODIC) is a multidisciplinary ANR project involving skills in industrial engineering, cognitive psychology, and software engineering.

been tackled and, therefore, KPIs have been computed for industrial systems. Section 3 discusses the challenges related to permit performance evaluation for a wide range of domains.

2 xDSL to Model Simple Manufacturing System

Early performance assessment demands both the modelling of system behavior and the ability to simulate it. Executable Domain-Specific Languages (xDSLs) are introduced for this purpose. An executable DSL is a language that provides a set of concepts suitable for a specific domain. We consider it to be composed of two parts. First, the abstract syntax defines the concepts of the language and their relationships. An abstract syntax can be defined as an object-oriented model called a metamodel. Second, the operational semantics of an xDSL, which consists of two parts: the definition of the possible runtime states of a model under execution, and a set of execution rules defining how such a runtime state changes over time. We introduce an executable DSL called Simple Manufacturing System (SMS xDSL), which has been implemented in order to enable modeling and simulating simple manufacturing systems.

2.1 Abstract syntax

SMS xDSL does not cover exhaustively all industrial features, but it is meant to include the basic concepts of the manufacturing industry.

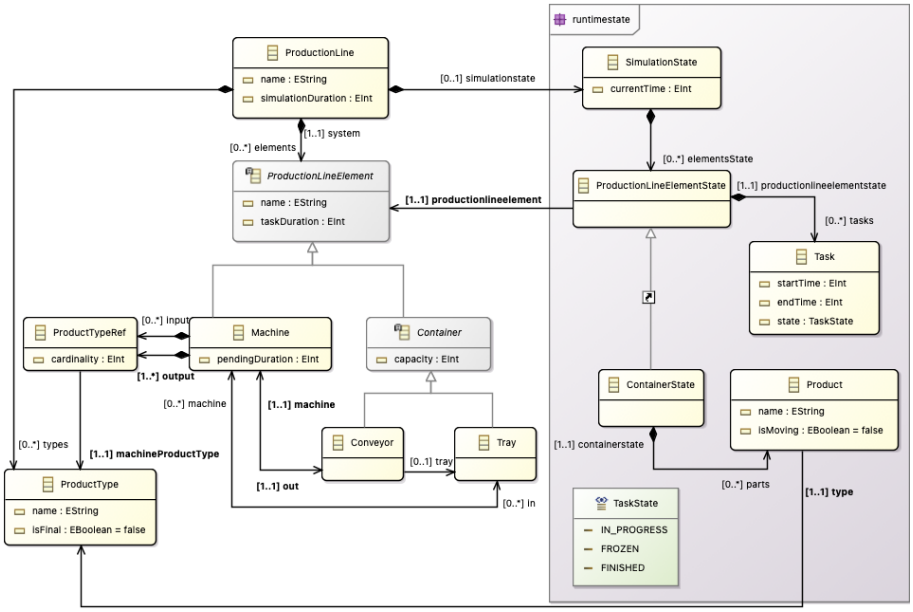

Fig. 1: Abstract syntax (left part) and runtime state definition (right part) of the SMS xDSL

The abstract syntax of the SMS xDSL is depicted on the left part of the Figure 1. Concepts related to production line are defined as a set of meta-classes and their features (i.e., attributes, references). The root element of an SMS model is a *ProductionLine* instance, which contains a set of *ProductionLineElements*. A *ProductionLineElement* is en entity that is able to accomplish a task (e.g., process, move products) and the *taskDuration* value meant to specify the time dedicated to the task execution.

A *Machine* is a *ProductionLineElement* that crafts products. It may consume a set of products as input, and it produces a non-empty set of products as output. These inputs and outputs are represented by *ProductTypeRef* instances, each specifying both a *ProductType* (i.e., what products are required or produced) and a cardinality (i.e., how many products of said types are required or produced).

A *Container* is a *ProductionLineElement* that contains and/or transports products. Two types of containers are possible: a *Conveyor* transports elements, whereas a *Tray* temporarily stores products to be handled. A *Machine* must be connected to one or multiple *Trays* to receive its input products, and must be connected to one *Conveyor* where it drops the output products.

2.2 Operational semantics

We define the operational semantics (OS) of the SMS xDSL, which will then allow the simulation of any SMS model.

2.2.1 SMS runtime state definition

The first part of the operational semantics is the definition of the possible runtime states of a model under execution. In SMS xDSL case, the runtime state definition is depicted at the right part of Figure 1. It is defined in a separate package of the SMS xDSL metamodel. During simulation, an SMS xDSL model contains at each *currentTime* (i.e., an attribute) a *SimulationState*. This *currentTime* is changed each time the runtime state is updated, which occurs when a task of an element is completed and/or it starts. The *SimulationState* encompasses a set of *ProductionLineElementState* for each *ProductionLineElement*. Each *ProductionLineElementState* instance may contain a set of *Task* instances, which inform about what are the ongoing tasks of the module, when they have been started or finished and their state (e.g., IN PROGRESS, FROZEN, or FINISHED). Particularly, the *ContainerState* may contain the instances of *Product* in the runtime.

2.2.2 SMS execution rules

The second part of OS is a set of execution rules that specify how the runtime state of a given SMS model changes over time during a simulation. Hence, we define, for a module declared in the abstract syntax, how it would behave in the execution and under which constraints. At this level, we also implement the time advancement relaying on the principles of discrete-event simulation.

3 Definition and Computation of KPIs for Performance Evaluation

As has been stated before, one major problem in evaluating the performance is that KPIs are typically tailored to a particular domain, and must therefore be redefined for each new considered context. To outstrip this problem, we have proposed to define KPIs directly at the level of the language, thus making them available for domain experts at the model level. This proposal was the contribution that has been recently published [1]. The process of KPI definition and computation depicted in Figure 2 demonstrates this proposal.

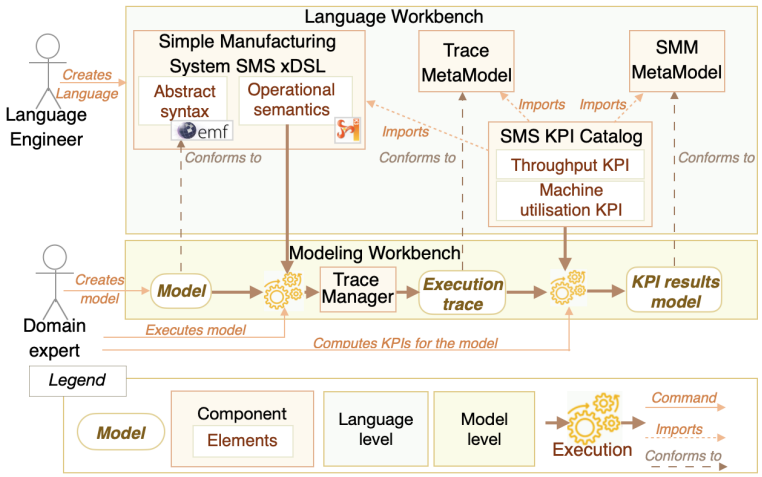

Fig. 2: KPI definition and computation process

The computation of KPIs is based on defining formulas, which was the purpose of introducing the central element, which contains a list of KPIs with their formulas: *SMS KPI Catalog*. The latter is implemented at the level of the language in the language workbench, and it embeds different KPIs that the domain expert may select when assessing the performance of a system. However, the computation of KPI formulas cannot begin without sufficient data. For this reason, simulation information is required. Thus, we assume the existence of a *Trace Manager* mechanism that captures the model state during the simulation, and generates the execution traces [2] at the end. Traces will be analyzed by a *SMS KPI Catalog* software to retrieve required information.

[2] Trace is a sequence of model states captured during the simulation.

According to this process, the domain expert can create the models, simulate them, then select KPIs from the *SMS KPI Catalog* and obtain the results that will be structured in a persisted Structured Metrics Metamodel (SMM[3]) model.

4 Discussions and Ongoing Work

4.1 Generalizing KPI description for xDSLs

At this stage, the KPI computation relies on a software to query the execution trace and retrieve necessary information. However, language-related elements are hard-coded into the software, which leads to rigidity, and the software cannot fit in the context of other languages.

Considering these issues, we should implement the parameterized aspect that allows KPIs in any context to be computed. Therefore, the language-related elements will not be embedded directly into the source code. We should suggest default KPIs; allow the domain expert to select the relevant KPIs to the current context and allow him to provide new KPIs [7].

Responding to these requirements, an executable *Performance Evaluation Language* (PEL) that opens the possibility to edit and compute KPIs may be designed [6]. Models conforming to PEL can accept a model under evaluation (MUE) conforming to a given xDSL as input, and they can specify the desired KPIs. A mechanism to collect necessary data for KPI computation may be explored.

4.2 Extension of the language concepts

At this point, we had assumed the existence of a *Trace Manager* mechanism that captures the runtime values through which the model has gone during the simulation, and thus generates the execution traces. The latter are considered to be the pivot source of the KPI computation process, because they permit to compute KPIs. Nevertheless, it remains one shortcoming: the execution traces may not contain all necessary information for all requested KPIs.

The content of the traces specifically relies on concepts introduced at the level of the language. These language concepts can either be explicitly introduced in the runtime state definition at the level of the metamodel in the form of metaclasses and their attributes, or they can be implicitly embedded in the execution rules. Consequently, the lack of information in the execution traces is caused by the absence of the introduction of necessary concepts at the language level. To grapple with this issue, we investigate a way to extend a given language with required concepts.

References

[1] Hiba Ajabri, Jean-Marie Mottu, and Erwan Bousse. Defining KPIs for Executable DSLs: A Manufacturing System Case Study:. In *Proceedings of the 12th International Conference on Model-Based Software and Systems Engineering*, pages 169–178, Rome, Italy, 2024. SCITEPRESS - Science and Technology Publications.

[2] Kyoungho An, Adam Trewyn, Aniruddha Gokhale, and Shivakumar Sastry. Model-Driven Performance Analysis of Reconfigurable Conveyor Systems Used in Material Handling Applications. In *2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*, pages 141–150, Chicago, IL, USA, April 2011. IEEE.

[3] Borja Ramis Ferrer, Usman Muhammad, Wael Mohammed, and José Martínez Lastra. Implementing and visualizing ISO 22400 key performance indicators for monitoring discrete manufacturing systems. *Machines*, 6(3):39, September 2018.

[4] Benjamin Kaiser, Alexander Reichle, and Alexander Verl. Model-based automatic generation of digital twin models for the simulation of reconfigurable manufacturing systems for timber construction. *Procedia CIRP*, 107:387–392, 2022.

[5] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, and H. Van Brussel. Reconfigurable Manufacturing Systems. *CIRP Annals*, 48(2):527–540, 1999.

[6] Thibault Béziers la Fosse, Massimo Tisi, Jean-Marie Mottu, and Gerson Sunyé. Annotating executable DSLs with energy estimation formulas. In *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*. ACM, November 2020.

[7] Ivan Monahov, Thomas Reschenhofer, and Florian Matthes. Design and Prototypical Implementation of a Language Empowering Business Users to Define Key Performance Indicators for Enterprise Architecture Management. In *2013 17th IEEE International Enterprise Distributed Object Computing Conference Workshops*, pages 337–346, Vancouver, BC, Canada, September 2013. IEEE.

[3] https://www.omg.org/spec/SMM/1.2/