

A Domain Specific Modeling Language Framework (DSL) for Representative Medical Prescription by using Generic Modeling Environment (GME)

Llahm Omar Faraj Ben Dalla¹, Ali Mohammed Ameer El-sseid², Tarik Milod Alarbi Ahmad³, Mohammed Ali Mohammed El-sseid⁴

Software Engineering Department¹, Medical research departement², Computer Engineering Department³, Software Engineering Department⁴

Sebha Technical Uniersity¹, Sebha University², University of Gharyan³, Sebha Technical University⁴

Email: mohmdaesed@gmail.com¹, Nilufer.design@gmail.com², tma_7444@yahoo.com³, mohmdaesed2005@gmail.com⁴

Abstract

This paper introduces and propose a domain specific modeling language for representing medical Prescription. the flow of the Prescription between different users and different parts of a system. Furthermore, this research study will define a meta-modeling for this language by using Generic Modeling Environment (GME) DevOps and introduce domain specific language DSL. In addition, after defining the meta-model for this study language will be automatically generated by using automated tool support of the (GME) DevOps. In addition, the result of this beneficial study is important for several domains such as industrial world, educational world, medical world as well as scientific world in addition researchers who aimed for some investigations outcome based on Generic Modeling Environment (GME) DevOps usage.

Keywords: *metamodel, modeling language , GME, Generic Modeling Environment, Domain Specific language*

I. Introduction

Prescriptions may be entered into an electronic medical record system and transmitted electronically to a pharmacy. Alternatively, a prescription may be handwritten on preprinted prescription forms that are assembled into pads, or printed onto similar forms using a computer printer [1]. In some cases, a prescription may be transmitted from the physician to the pharmacist orally by telephone, although this practice may increase the risk of medical error. The content of a prescription includes the name and address of the prescribing provider Unique for each prescription is the name of the patient DevOps [2]. The patient's name and address must also be recorded [3]. Each prescription is dated. The paper prescription system that was in use before, wasted the time of all parties involved in the prescribing and buying/selling of medications. The scarce resource of a doctor's appointment time was wasted on writing recurring prescriptions and the patients had to spend time waiting for their appointment [4].

BACKGROUND

A. domain Information

In this paper, this study present a simple case study to introduce the reader to the typical design flow developing a GME-based too set DevOps [5] .The domain of Prescriptions is written by doctors and sent to the pharmacy. The patient goes directly to the pharmacy, where the pharmacist systems to receive the recipe and make sure as regular medication and then handed over to the patient. Whether patient already have a prescription from your doctor, or need to get medication prescribed privately, online prescription offers you more choice and flexibility with managing and receiving your medication DevOps [5]. When a doctor prescribes medicine using the system, he or she does so electronically, with the aid of an online form [6]. The framework DevOps stores incoming prescriptions (messages) and sends patients' prescriptions on demand to a pharmacy's information system. The prescription is then immediately accessible in every pharmacy on request. The project will make the health care framework environmentally friendlier. Since the data communications are digitized, it significantly reduces paper usage throughout the healthcare sector [7].

Doctors spend significantly less time issuing prescriptions. They will get feedback as to whether the medication was actually purchased by the patient. In terms of treatment, it is very important for doctors to know whether their patients have actually purchased the medications that have been prescribed to them [8]. If a patient's health does not improve, then it is easier to know whether the problem is with unsuitable medication or with a patient who has simply not followed instructions. Pharmacies will spend significantly less time filling prescriptions and can pay more attention to serving clients [9]. For them, things are simplified by the fact that the greater part of prescription data on the prescription is already entered into the system by the doctor, which is why they are only required to add to the prescription information actual delivered medication and sales data. In most cases this can be done using a barcode reader. The prescription is then ready for e-invoicing to the Health Insurance Fund [10].

Patients no longer have to worry about carrying a paper prescription or losing the prescription. Another major advantage of the system is that doctor visits are no longer needed for routine refills. Patients with chronic illnesses needing a new prescription for their regular medication would only have to contact their doctor by e-mail, Skype or phone, and the doctor can issue refills with a couple clicks of the mouse [11]. After that, the patient can have the drugs collected from the pharmacy by a third party providing they have the patient's ID number and show their own ID.

B. Domain Specific Languages

A domain-specific language (DSL) DevOps is a computer language specialized to a particular application domain. This is in contrast to a general-purpose language (GPL) DevOps, which is broadly applicable across domains. There are a wide variety of DSLs, ranging from widely used languages for common domains, such as HTML for web pages, down to languages used by only one or a few pieces of software, such as Emacs Lisp for GNU Emacs and XEmacs. DSLs can be further subdivided by the kind of language, and include domain-specific markup languages, domain-specific modeling languages (more generally, specification languages) [12], and domain-specific programming languages. Special-purpose computer languages have always existed in the

computer age, but the term "domain-specific language" has become more popular due to the rise of domain-specific modeling. Simpler DSLs (DevOps), particularly ones used by a single application, are sometimes informally called mini-languages [13]. The line between general-purpose languages and domain-specific languages is not always sharp, as a language may have specialized features for a particular domain but be applicable more broadly, or conversely may in principle be capable of broad application but in practice used primarily for a specific domain. For example, Perl was originally developed as a text-processing and glue language, for the same domain as AWK and shell scripts, but was mostly used as a general-purpose programming language later on. By contrast, PostScript is a Turing complete language, and in principle can be used for any task, but in practice is narrowly used as a page description language.

To define a language, one needs a language to write the definition in. The language of a model is often called a metamodel, hence the language for defining a modeling language is a meta-metamodel [10]. Meta-metamodels can be divided into two groups: those that are derived from or customizations of existing languages, and those that have been developed specifically as meta-metamodels. Furthermore, derived meta-metamodels include Entity Relationship Diagrams, Formal languages, Extended Backus-Naur form (EBNF), Ontology languages [11], XML Schema, and Meta-Object Facility (MOF) DevOps. The strengths of these languages tend to be in the familiarity and standardization of the original language.

Most existing domain-specific language takes place with domain-specific language environments, either commercial such as MetaEdit+ or Actifsource, open source such as GEMS, or academic such as GME [1, 2, 13]. The increasing popularity of domain-specific language has led to domain-specific language frameworks being added to existing IDEs, e.g. Eclipse Modeling Project (EMP) with EMF and GMF, or in Microsoft's DSL Tools for Software Factories. The vocabulary of the domain-specific languages implemented by different GME configurations is based on a set of generic concepts built into GME itself. The choice of these generic concepts is the most critical design decision. GME supports various concepts for building large-scale, complex models. These include: hierarchy, multiple aspects, sets, references, and explicit constraints [1, 12].

GME is a domain-specific modeling environment that can be configured and adapted from meta-level paradigm specifications. Thus, based on the paradigm, GME can be adapted quickly to a domain-specific tool that represents a particular engineering domain [1, 10, 12].

C. Metamodeling

The first thing must do using GME is define a sketch of a metamodel, which is basically a UML Class Diagram extended with some additional concepts. These additional concepts include defining any necessary OCL constraints and also some GME specific features such as configurable model visualization properties. After the metamodel is initially defined, it can be iteratively refined until it reaches a mature state that captures all pertinent features of the domain. This refinement results in an improvement to the domain-specific modeling language (DSML). As the quality of the DSML improves, one can express better domain models using the DSML (DevOps).

GME metamodels must be created using the MetaGME paradigm, which is installed and registered with GME. In addition, MetaGME is just another modeling language; however, its own metamodel can be considered the meta-metamodel. That is, it defines the concepts that are built-in to GME.

- *Atom* – used to represent an atomic element,
- *Model* – used to represent a container element,
- *Reference* – used as a pointer to other elements,
- *Set* – used to group elements, and

- *Connection* – used to associate elements.


These elements are called *first class objects* (FCOs) in GME. Furthermore, FCOs can contain both textual Attributes (of Enumeration, Boolean and Field type) and Constraints, which are OCL-based expressions for providing verifiability for the models. Another important concept in GME is the *Aspect*. Moreover, models can have multiple aspects (or viewpoints) that select a subset of the modeling concepts to show to the modeler at once. For example, a model of a distributed software system might have a data flow and control flow aspect. The most basic step in the metamodeling process consists of determining two things: the entities used by the model, and the relations between them. Information used to identify and qualify certain entities and relations will be assigned to them as attributes. Metamodeling, in a nutshell, is the mapping of specification concepts onto entities, relations and attributes. In addition, the project is named "prescription" and is associated with the MetaGME paradigm (DevOps), the GME built-in metamodeling paradigm that configures the environment for use as a metamodeling tool.

II. A Metamodeling for prescription

When this research study design meta-model initially, prescription domain must be analyzed to find the basic concepts that the metamodel must contain. First, I will build the metamodel for prescription flow paradigm. I will explain the metamodel creation of the prescription flow paradigm.

In this research study prescription domain, the metamodel contain only one model and some Atoms, also contain connection and connector the first term that may identify is model will be the entity that represents a prescription. Give it the name [10] "Prescription Model" by selecting it and clicking the topmost input field in the Attributes / Preferences / Properties window and change in root folder=true. then create five<<Atom>> classes, and name them "Prescription", "Actor", "Pharmacy", "FillOrder" and "Payment". The relations will be represented by connection entities, so create a <<Connection>> [11].

Actor represents a user's. users could be doctor or Pharmacist or Patient derive both the doctor and the Pharmacist and the Patient atoms from a base actor atom. We use the "Inheritance" operator (Δ) for this . first, connect the base class to the inheritance operator, then connect the inheritance operator to one of the objects to be derived. Thus, the same way creates Payment and FillOrder. create relationships between entities [12]. Relationships are represented by lines, so switch the

editor mode to "Add Connection" (). we use connector (*), This is a ternary relation between the association class and the two endpoints of the association, so another helping object, . Only one source can be specified for each connection, but we can get around this by using the common base class. for example, Connect "Doctorto" the connector dot. The line displays "src" as the role, and "0..*" to specify the destination: first click the connector dot, and then the "Prescription" class. The association class, "Connection", is the third leg of the association relationship. When you connect this class to the connector dot (in any order), GME displays a window asking you to clarify the role of this relation. Select "Association Class" here as presented in Figure. 1. Below.

After the completion of the design metamodel we create Model by converting the metamodel into a GME paradigm. By clicking on the button with the cogwheel icon (metaGME Interpret) in the toolbar .

A metamodel must first be interpreted with the MetaGME interpreter, and the generated paradigm description (an XML file which has an "xmp" extension) must be registered in the GME registry before any models of that paradigm can be built. After doing both of these steps, we can create a

model of Prescription The first step in building this application is to create a new project in GME whose metamodel is our Prescription paradigm, and then to insert an App element into the Root Folder (the top level container) of this project.(Figure 2) .

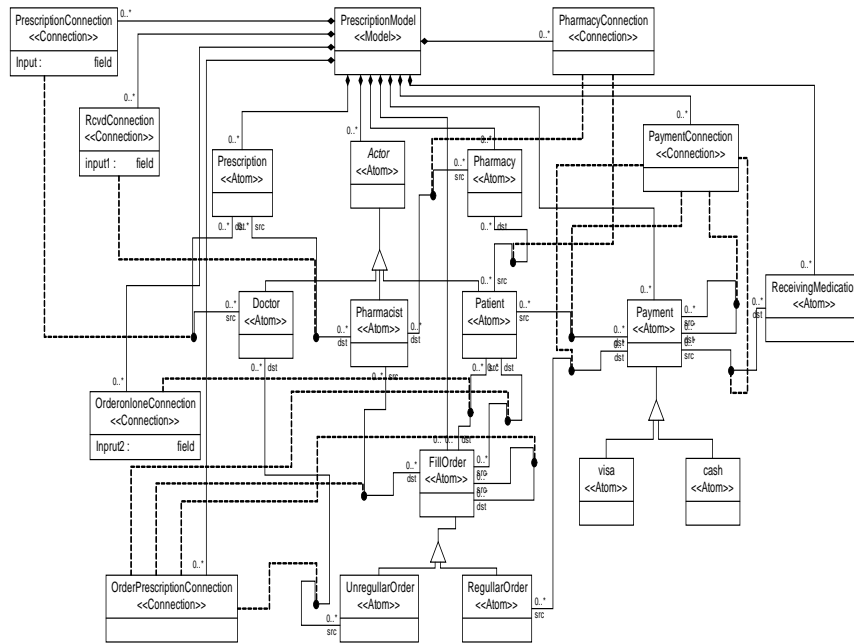


Figure.1. he Prescription Metamodel.

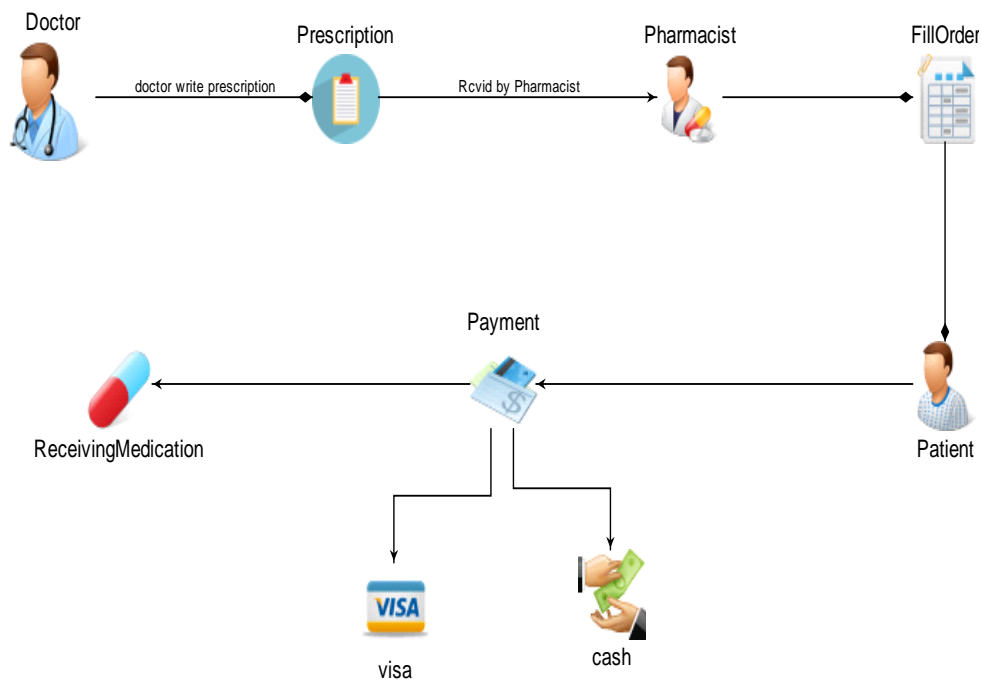


Fig. 1. **The research first model.**

The first model DevOps which represent doctor can give prescription directly to the pharmacist and the patient will order the payment and get the medication which can be helpful for this kind of transaction.

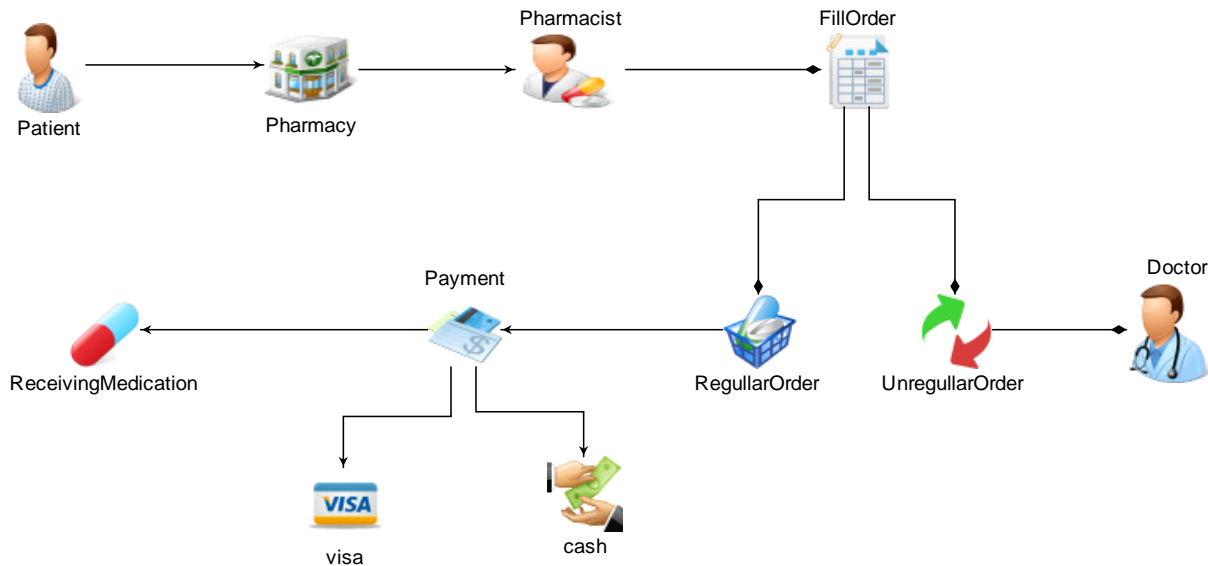


Fig. 2. **Model.2**

The second model represent the patient can go to the pharmacy and ask nthe parmasist about his stste and the pharماسist can give him first prescription which have some medication so he can pay for that medication .

III. Conclusion

In this paper this study have presented domain of prescription Applications through Generic Modeling Environment DevOps, GME, which can be helpful for Prescription system and can be more officiant in the performance every transaction in a good way. Besides this system can be reusability transformed to any system as well as the productivity has been improved further .

In DSM, language engineers have full control over the languages, and can thus decide on an appropriate model integration approach for their situation to fix all the problems that can face the actors such as patient or doctor or pharmacist . Sometimes it can be possible to integrate several areas of interests, e.g. persistency, navigation, layout, data, into a single modeling language DevOps, whereas at other times the use of different languages DevOps, and explicit integration among the models is preferred. In any case, there will always be multiple model diagrams to integrate, whether by elements having subdiagrams, or elements being reused or referenced in several diagrams. In addition, as a result this domain specific language can be helpful for this situation to reduce cost and reusable systems.

References

1. Ruiz-Rube, I., Person, T., Dodero, J. M., Mota, J. M., & Sánchez-Jara, J. M. (2020). Applying static code analysis for domain-specific languages. *Software and Systems Modeling*, 19(1), 95-110.
2. Ghosh, R., Hockett, H. E., Quirk, A. J., & Sun, L. (2020). U.S. Patent No. 10,530,842. Washington, DC: U.S. Patent and Trademark Office.
3. De Nicola, R., Ferrari, G., Pugliese, R., & Tiezzi, F. (2020). A formal approach to the engineering of domain-specific distributed systems. *Journal of Logical and Algebraic Methods in Programming*, 111, 100511.
4. Goumopoulos, C., & Mavrommati, I. (2020). A framework for pervasive computing applications based on smart objects and end user development. *Journal of Systems and Software*, 162, 110496.
5. Akesson, B., Hooman, J., Sleuters, J., & Yankov, A. (2020). Reducing design time and promoting evolvability using Domain-Specific Languages in an industrial context. In *Model Management and Analytics for Large Scale Systems* (pp. 245-272). Academic Press.
6. Ferreira, J. J., & Monteiro, M. D. S. (2020). Do ML Experts Discuss Explainability for AI Systems? A discussion case in the industry for a domain-specific solution. arXiv preprint arXiv:2002.12450.
7. Elrakaiy, Y., Spoletini, P., & Nuseibeh, B. (2020). Optimal by Design: Model-Driven Synthesis of Adaptation Strategies for Autonomous Systems. arXiv preprint arXiv:2001.08525.
8. Gerl, A. (2020). Modelling of a Privacy Language and Efficient Policy-based De-identification.
9. Ruiz-Rube, I., Person, T., Dodero, J. M., Mota, J. M., & Sánchez-Jara, J. M. (2020). Applying static code analysis for domain-specific languages. *Software and Systems Modeling*, 19(1), 95-110.
10. Bernelas, J. M. G., Junker, U. M., & Mery, S. (2020). U.S. Patent No. 10,552,534. Washington, DC: U.S. Patent and Trademark Office.
11. Iskrenova-Ekiert, E., Deppen, T. O., Dierker, D. J., & Patnaik, S. S. (2020). Towards a Common Modeling Environment for Aircraft Power and Thermal Systems Design and Optimization: Introducing the Simulation Platform APTT-SP. In *AIAA Scitech 2020 Forum* (p. 2118).
12. Tarcar, A. K., Tiwari, A., Rao, D., Dhaimodker, V. N., Rebelo, P., & Desai, R. (2020). Healthcare NER Models Using Language Model Pretraining. In *HSDM 2020 Workshop on Health Search and Data Mining* (Vol. 1).
13. Reist, J., Frazier, J., Rottingham, A., Welsh, M., Viyyuri, B. R., & Witry, M. (2020). Provider beliefs on the Barriers and Facilitators to Prescription Monitoring Programs and Mandated Use. *Substance use & misuse*, 55(1), 1-11.