RESEARCH

On the Use of Metaprogramming and Domain Specific Languages: An Experience Report in the Logistics Domain

Pedro HT Costa*, Edna D Canedo and Rodrigo Bonifácio

*Correspondence:
pedro.costa@aluno.unb.br
Computer Science Department, University of Brasília - (UnB), P.O. Box 4466, 70910-900 Brasília, BR
Full list of author information is available at the end of the article

Abstract

In this paper we present the main design and architectural decisions to build an enterprise system that deals with the distribution of equipment to the Brazilian Army. The requirements of the system are far from trivial, and we have to conciliate the need to extract business rules from the source code (increasing software flexibility) with the requirements of not exposing the use of declarative languages to use a rule-based engine and simplifying the tests of the application. Considering these goals, we present in this paper a seamless integration of meta-programming and domain-specific languages. The use of meta-programming allowed us to isolate and abstract all definitions necessary to externalize the business rules that ensure the correct distribution of the equipment through the Brazilian Army unities. The use of a domain specific language simplified the process of writing automatic test cases related to our domain. Although our architecture targets the specific needs of the Brazilian Army, we believe that logistic systems from other institutions might also benefit from our technical decisions.

Keywords: Software Architecture; Generative Programming; Domain Specific Languages; Military Logistics

Introduction

One of the logistics' phases to distribute equipment to an army force contemplates the *distribution planning* of materials throughout military unities. This phase is known as *material endowment*, and involves several steps, including the specification of relevant materials, often independent of vendor; the definition of rules that link military materials and equipment's to organizational unities (e.g., departments, regiments, and military posts); the execution of the rules for deriving the materials originally intended for each *abstract military unit*; and the designation of the materials that must be actually distributed to the *concrete military organizations*, which may suffer variations due to the inclusion and suppression of positions for a particular organizational unit of an army force. In the context of this paper, the reader might consider the scenario of the Brazilian Army.

Decision support systems are essential in this domain, given the importance and complexity of the processes related to the allocation of military employment materials. In particular, the rules used to predict military materials and equipment are not trivial, and implementing these rules directly in the source code makes the systems difficult to understand and maintain. This problem mostly occurs because

the number of conditions that need to be expressed is proportional to the complexity of the organizational structure and the number of materials. In the case of an entire army force, this number is noteworthy. In addition, a "*hard-coding*" solution does not allow domain experts to trivially specify and test new derivation rules. For these reasons, it is preferably to implement these rules declaratively, using languages, tools, or libraries that support some sort of inference mechanism.

Nevertheless, introducing either a logic language (e.g., Prolog or Datalog) or a specific library (such as Drools [1]) in a complex enterprise system may lead to some type of *architectural conflict*, particularly in environments whose whole ecosystem system is well established and based on a set of constraints related to both language and libraries usage. Furthermore, these solutions require a reasonable learning curve, which can hamper software maintenance activities.

This article describes the main design and architectural decisions adopted to implement the mechanisms that assist decision makers to plan the distribution of military materials and equipment across the organizational structure of the Brazilian Army. These decisions aim to both (a) abstract the adoption of a business rule engine using **meta-programming** and (b) allow domain experts to simulate the definition and testing of rules using a **domain specific language**. Accordingly, this paper presents the following contributions:

- A description of the use of generative programming techniques to raise the abstraction level related to the adoption of a logic programming language in the specification of rules for the distribution of materials and equipment through organizational unities.
- A report of an empirical evaluation of our design and architectural decisions, reinforcing that they fulfill not only the needs of the domain specialists, but also existing technical constraints.

Although we discuss the distribution of materials and equipment for the military domain (which is far from trivial), we believe that the architectural decisions discussed here can be exploited to other logistics and endorsements scenarios in which it is also desirable to transparently introduce the support for rule based engines in enterprise systems.

We organized this article as follows. Section presents the concepts related to the domain of material logistics. Section presents the concepts related to business rules and rule based engines and Section presents some background information related to *generative programming*—techniques that we use to support the design and implementation of our approach. Section **??** presents the main design and architectural decisions we take to develop the mechanism responsible for the distribution of materials. In Section **??** we present the results of an empirical study that validate our approach. Finally, Section **??** and Section **??** relate our work with the exising literature and present some final remarks.

Technical Background

Military Logistics Systems

Military Logistics is the method used to implement military strategies and tactics as a method to gain competitive advantage [2]. Logistics is defined as the art of moving armies, as well as accommodating and supplying the military [3]. Strategy

and tactics provide the framework for conducting military operations, while logistics provides the means. In the context of this paper, we are more interested in the supplying aspects of logistics, which refers to the set of activities that deals with the provisioning of the material of different classes (including vehicles, armaments, munition, uniform, and fuel) necessary to the military unities (including a rich organizational structure and a huge number of soldiers).

In order to facilitate the administration and control of materials and equipment, the Brazilian Army uses a catalog system that is similar to one defined by the North Atlantic Treaty Organization (OTAN) [4], which classifies items into groups and classes. The Brazilian Army catalog uses a classification that considers the *type*, *class*, and *family* of the materials, ignoring some details (such as specific brands and suppliers). More detailed information is considered by other areas of the logistic system. Several rules govern the identification of materials and equipment, to avoid duplication and simplify the process of decision making. A logistic system is the integrated set of organizations, personnel, principles, and technical standards intended to provide the adequate flow for supplying materials [5]. This is a complex system, and thus must be organized in different modules.

In this paper we present the main design and architectural decisions related to one of the modules (SISDOT) of the *Integrated Logistics System* (SIGELOG) of the Brazilian Army. SISDOT deals with the identification of the necessary materials and equipment (hereafter MEMs) for the organizational unities of the Brazilian Army. Each organizational unity is a combination of a set of *generic organizations of specific types*, which describes the expected number of military unities of an organization (including armies, divisions, brigades, and so on). The goal of SISDOT is to estimate the set of MEMs that should be assigned to a generic organization, producing one of its main products (named QDM). Based on a set of QDMs, SISDOT then derives another estimate of MEMs that should be assigned to a concrete organizational unity (a QDMP).

Before generating a QDM, it is necessary to catalog all relevant MEMs to a given class of materials and then specify *high-level rules* (HLRs) that relate MEMs to the generic organizational unities, considering different elements of the structure—from the type of the unity and sub-unities to the qualifications of a soldier. That is, it is possible to elaborate a HLR that states that a military ambulance would be assigned to all *drivers within medical unities*. Differently, it is also possible to state that an automatic rifle will be assigned to all *sergeant that has been qualified as a shooter*. There is some degree of flexibility for writing these rules, that might be from two different kinds: rules that consider the entire structure of the Brazilian Army and rules that refer to specific elements of the organizational structure. The former details rules that are more specific for a set of military qualifications. The second allows the generation of rules that might be reused through different unities.

Deriving QDMs and QDMPs is not trivial and its complexity is due to several factors, including the rich organizational structure of the Brazilian Army, the huge number of existing functions and qualifications that must be considered, and the reasonable number of MEMs. The process might also vary depending on the class of the materials. Moreover, it is possible to specify the *high-level rules* in different ways, which in the end might interact to each other leading to undesired results (e.g., the same material being assigned several times to a given soldier).

Due to number of logical decisions, using an imperative language to implement the process of QDM generation is not an interesting choice. However, the SIGELOG architecture constraints the development to use the Java Enterprise Edition platform—so we had little space to introduce a more suitable language (such as Prolog) to generate QDMs. Other constraints guided the architectural decisions of SISDOT, including:

- Maintenance: the set of languages and libraries are limited to those already used in the SIGELOG development.
- Performance: considering a total of 1000 *high-level rules*, the system must be able to generate a QDM in less than 10 seconds.
- Testing: the technical decisions must not compromise the testability of the system, being expected a high degree of automated tests.

To deal with the first constraint, we decided to use a meta-programming approach that derives *low-level rules* for Drools (a rule based engine for Java) *on the fly*, from a set of *high-level rules* represented as domain objects that must exist in a database. In this way, it is not necessary to understand the language used to specify rules in Drools. Surely, although this approach increases flexibility and allows us to externalize the *low-level rules*, it might introduce some undesired effect on the performance of the system: we generate all low-level rules every time a QDM is built. We are using a cache mechanism and performance tests (as we discuss in Section **??**) shows that our architecture fulfills the second constraint. For the third constraint, we implemented a DSL that simplifies the specification and execution of test cases.

In the next sections (Section and Section) we introduce some technical background that might help the reader to understand the design and architectural decisions we take while implementing SISDOT.

Rule Based Systems

The process for deriving QDMs consists of applying a set of business rules that relates MEMs to the organizational structure of the Brazilian Army. In this paper, we named this set of business rules as *high-level rules*. A business rule is a compact, atomic, and well-formed statement about an aspect of the business that can be expressed in terms that may be directly related to the business and its employees using a simple and unambiguous language that might be accessible to all stakeholders [6]. This type of structure is very important for organizations, as they need to deal more and more with complex scenarios. These scenarios consist of a large number of individual decisions, working together to provide a complex assessment of the organization as a whole [7].

Rule Based Systems (RBS), also known as production systems or expert systems, are the simplest form of artificial intelligence that help to design solutions that reason about and take actions from business rules (a feature that is interesting in the context of SISDOT), using rules such as knowledge representation [8].Instead of representing knowledge in a declarative and static way as a set of things that are true, RBS represents knowledge in terms of a set of rules that tells what to do or what to accomplish in different situations [8]. RBS consist of two basic elements: a *knowledge base* to store knowledge and an *inference engine* that implements algorithms to manipulate the knowledge represented in the knowledge base [8, 9, 10].

The knowledge base stores facts and rules [11]. a fact is a usually static statement about properties, relations, or propositions [11] and should be something relevant to the initial state of the system [8]. A rule is a conditional statement that links certain conditions to actions or results [12], being able to express policies, preferences, and constraints. Rules can be used to express deductive knowledge, such as logical relationships, and thus support inference, verification, or evaluation tasks [11].

A rule "if-then" takes a form like *"if x is A then take a set of actions"*. The conditional part is known as antecedent, premise, condition, or left-hand-side (LHS). The other part is known as consequent, conclusion, action, or right-hand-side (RHS) [8, 12]. Rule-based systems works as follows [8]: it starts with a knowledge base, which contains all appropriate knowledge coded using "if-then" rules, and a working memory, which contain all data, assertions, and information initially known.

The system examines all rule conditions (LHS) and determines a subset of rules whose conditions are satisfied, based on the data available at the working memory. From this information set, some of the rules are triggered. The choice of rules is based on a conflict resolution strategy. When the rule is triggered, all actions specified in its `then` clause (RHS) are executed. These actions can modify the working memory, the rule base itself, or take whatever action that the system programmer chooses to include. This *trigger rule loop* continues until a termination criterion is met. This end criterion can be given by the fact that there are no more rules whose conditions are satisfied or a rule is triggered whose action specifies that the program should be finalized.

Drools is an example of a business logic integration platform (BLiP) written in the Java programming language and that fits some requirements of SISDOT (in particular, its integration with Java and the Wildfly application server). We use Drools as the *rule-based system* in the QDM generation. However, to abstract the use of a RBS, we generate the Drools rules using a meta-programming approach, a specific technique for *generative programming*.

Generative Programming

Generative Programming (GP) is related to the design and implementation of software components that can be combined to generate specialized and highly optimized systems, fulfilling specific requirements [13]. With this approach, instead of developing solutions from scratch, domain-specific solutions are generated from reusable software components [14]. GP is applied on the manufacturing of software products from components in a system in an automated way (and within economic constraints), i.e., using an approach that is similar to that found in industries that have been producing mechanical, electronic, and other goods for decades [15]. The main idea in GP is similar to the one that led people to switch from Assembly languages to high-level languages.

In our context, we use GP to deal with two different aspects of SISDOT. First, we raise the level of abstraction by automatically generating low-level Drools rules from an existing set of *high-level rules* (previously defined by the domain experts). This is a type of *meta-programming* based on template engines—we use a template to generate a set of facts and rules (which are the fundamental mechanisms of logic

programming) from domain objects. We also raise the level of abstraction to assist developers in the specification and execution of test cases, using a domain specific language and a set of customized libraries and tools.

Metaprogramming and Template Engines

Metaprogramming concerns to the implementation of programs that generate other programs. In the context of SISDOT, we use a metaprogramming technique to generate programs that are interpreted by Drools. The main goal is to abstract the use of a specific language for implementing the *low-level rules* used to generate QDMs. There are many different approaches for metaprogramming, including preprocessing, source to source transformations, and template engines. A template engine is a generic tool for generating textual output from templates and data files. They can be used in the development of software that requires the automatic generation of code according to specific purposes [16].

The need for dynamically generated web pages has led to the development of numerous template engines in an effort to make web application development easier, improve flexibility, reduce maintenance costs, and enable parallel encoding and development using HTML like languages. In addition to being widely used in dynamic web page generation, templates are used in other tasks, being one of the main techniques to generate source code from high level specifications. More recently, languages such as Clojure, Ruby, Scala, and Haskell adopted the use of template and *quasiquote* mechanisms as a strategy for code reuse and program generation (some of them considering type systems). A template is a text document that combines placeholders and linguistic formulas used to describe something in a specific domain [17].

In this work we use the FreeMarker template engine, which is an open-source software designed to generate text from templates [18]. FreeMarker was chosen for several reasons, such as: it follows a general purpose model; it is faster than other template engines; it supports shared variables and model loaders; and its templates can be loaded or reloaded at runtime without the need to (re)compile the application [16, 19].

Domain Specific Languages

A language is a set of valid sentences, serving as a mechanism for expressing intentions [20]. Creating a new language is a time-consuming task, requires experience, and is usually performed by engineers specialized in languages construction [21]. To implement a language, one must construct an application that reads sentences and reacts appropriately to the phrases and input symbols it discovers. The need for new languages for many growing domains is increasing, as well as the emergence of more sophisticated tools that allow software engineers to define a new language with reasonable effort. As a result, a growing number of DSLs are being developed to increase developer productivity within specific domains [21].

Domain Specific Languages (DSL) are specification languages or programming languages with high level of abstraction, simple and concise [22], focused on specific domains, and are designed to facilitate the construction of applications, usually declaratively, with limited expressiveness lines of code, which solve these specific

problems [23]. A DSL is a computer language of limited expressiveness, through appropriate notations and abstractions, focused and generally restricted to a specific problem domain [24, 25]. DSLs have the potential to reduce the complexity of software development by increasing the level of abstraction for a domain. According to the application domain, different notations (textual, graphical, tabular) are used [26].

In this work we used Xtext [27] (a language workbench) to implement a DSL that helps developers to specify and execute test cases related to the process of QDM generation. The main rationale to building this DSL was the significant costs related to specifying *high-level rules* (particularly during acceptance tests). In the next section we present more details about our architectural decisions and implementation of our metaprogramming and DSL approach, which deals with the generation and test of QDMs.

Content

Text and results for this section, as per the individual journal's instructions for authors.

Section title

Text for this section ...

Sub-heading for section

Text for this sub-heading ...

Sub-sub heading for section

Text for this sub-sub-heading ...

Sub-sub-sub heading for section Text for this sub-sub-sub-heading ... In this section we examine the growth rate of the mean of Z_0, Z_1 and Z_2. In addition, we examine a common modeling assumption and note the importance of considering the tails of the extinction time T_x in studies of escape dynamics. We will first consider the expected resistant population at vT_x for some $v > 0$, (and temporarily assume $\alpha = 0$)

$$E\big[Z_1(vT_x)\big] = E\bigg[\mu T_x \int_0^{v\wedge 1} Z_0(uT_x)\exp\big(\lambda_1 T_x(v-u)\big)\,du\bigg].$$

If we assume that sensitive cells follow a deterministic decay $Z_0(t) = xe^{\lambda_0 t}$ and approximate their extinction time as $T_x \approx -\frac{1}{\lambda_0}\log x$, then we can heuristically estimate the expected value as

$$\begin{aligned} E\big[Z_1(vT_x)\big] &= \frac{\mu}{r}\log x \int_0^{v\wedge 1} x^{1-u}x^{(\lambda_1/r)(v-u)}\,du \\ &= \frac{\mu}{r}x^{1-\lambda_1/\lambda_0 v}\log x \int_0^{v\wedge 1} x^{-u(1+\lambda_1/r)}\,du \\ &= \frac{\mu}{\lambda_1-\lambda_0}x^{1+\lambda_1/rv}\bigg(1-\exp\bigg[-(v\wedge 1)\bigg(1+\frac{\lambda_1}{r}\bigg)\log x\bigg]\bigg). \quad (1) \end{aligned}$$

Thus we observe that this expected value is finite for all $v > 0$ (also see [?, ?, ?, ?, ?]).

Competing interests
The authors declare that they have no competing interests.

Author's contributions
Text for this section . . .

Acknowledgements
Text for this section . . .

References

1. Browne, P.: JBoss Drools Business Rules. Packt Publishing Ltd, ??? (2009)
2. Rutner, S.M., Aviles, M., Cox, S.: Logistics evolution: a comparison of military and commercial logistics thought. The International Journal of Logistics Management **23**(1), 96–118 (2012). doi:10.1108/09574091211226948
3. Prebilič, V.: Theoretical aspects of military logistics. Defense & Security Analysis **22**(2), 159–177 (2006). doi:10.1080/14751790600764037
4. OTAN, L.C.: NATO Logistics Handbook. 2nd Edition, 2012 (2012). https://www.nato.int/docu/logi-en/logistics_hndbk_2012-en.pdf
5. Brasil, M.d.D.: Manual de Campanha C 100-10 - Logistica Militar Terrestre (2ª edição, 2003). http://bibliotecamilitar.com.br/wp-content/uploads/2016/02/02-manual_c_100-10-logistica_militar_terrestre.pdf
6. Graham, I.: Business Rules Management and Service Oriented Architecture: a Pattern Language. John wiley & sons, ??? (2007)
7. Salatino, M., De Maio, M., Aliverti, E.: Mastering JBoss Drools 6. Packt Publishing Ltd, ??? (2016)
8. Grosan, C., Abraham, A.: Rule-Based Expert Systems, pp. 149–185. Springer, ??? (2011). doi:10.1007/978-3-642-21004-4_7. https://doi.org/10.1007/978-3-642-21004-4_7
9. Lucas, P.J.F., van der Gaag, L.C.: Principles of Expert Systems. International computer science series. Addison-Wesley, ??? (1991)
10. Gallacher, J.: Practical introduction to expert systems. Microprocessors and Microsystems **13**(1), 47–53 (1989). doi:10.1016/0141-9331(89)90034-3
11. Hayes-Roth, F.: Rule-based systems. Commun. ACM **28**(9), 921–932 (1985). doi:10.1145/4284.4286
12. Abraham, A.: Rule-Based Expert Systems. American Cancer Society, ??? (2005). Chap. 130. doi:10.1002/0471497398.mm422. https://onlinelibrary.wiley.com/doi/abs/10.1002/0471497398.mm422
13. Czarnecki, K.: Generative programming - principles and techniques of software engineering based on automated configuration and fragment-based component models. PhD thesis, Technische Universität Illmenau, Germany (1999). http://d-nb.info/958706700
14. Arora, R., Bangalore, P., Mernik, M.: Developing scientific applications using generative programming. In: 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering, pp. 51–58 (2009). doi:10.1109/SECSE.2009.5069162. https://ieeexplore.ieee.org/document/5069162/
15. Barth, B., Butler, G., Czarnecki, K., Eisenecker, U.: Generative programming. In: Frohner, Á. (ed.) Proceedings of the Workshops on Object-Oriented Technology. ECOOP '01, pp. 135–149. Springer, Berlin, Heidelberg (2002). http://dl.acm.org/citation.cfm?id=646781.705926
16. Benato, G.S., Affonso, F.J., Nakagawa, E.Y.: Infrastructure based on template engines for automatic generation of source code for self-adaptive software domain. In: Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE, pp. 30–35 (2017). doi:10.18293/SEKE2017-147. http://ksiresearchorg.ipage.com/seke/seke17paper/seke17paper_147.pdf
17. Segura, S., Durán, A., Troya, J., Cortés, A.R.: A template-based approach to describing metamorphic relations. In: 2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET), pp. 3–9 (2017). doi:10.1109/MET.2017.3
18. Radjenovic, J., Milosavljevic, B., Surla, D.: Modelling and implementation of catalogue cards using freemarker. Program **43**(1), 62–76 (2009). doi:10.1108/00330330910934110
19. Parr, T.J.: Web application internationalization and localization in action. In: Proceedings of the 6th International Conference on Web Engineering. ICWE '06, pp. 64–70 (2006). doi:10.1145/1145581.1145650. http://doi.acm.org/10.1145/1145581.1145650
20. Parr, T.: Language Implementation Patterns: Create Your Own Domain-specific and General Programming Languages. Pragmatic Bookshelf, ??? (2010)
21. Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schindler, M., Völkel, S.: Design guidelines for domain specific languages. CoRR **abs/1409.2378**(October), 7 (2014)
22. Raja, A., Lakshmanan, D.: Domain specific languages. International Journal of Computer Applications **1**(21), 105–111 (2010). doi:10.5120/37-640. arXiv:1011.1669v3
23. Neeraj, K.R., Janardhanan, P.S., Francis, A.B., Murali, R.: A domain specific language for business transaction processing. In: 2017 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES), pp. 1–7 (2017). doi:10.1109/SPICES.2017.8091270. https://ieeexplore.ieee.org/document/8091270/
24. Fowler, M., Parsons, R.: DSL: Linguagens Específicas de Domínio. Bookman, ??? (2013)
25. van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: An annotated bibliography. SIGPLAN Not. **35**(6), 26–36 (2000). doi:10.1145/352029.352035

26. Pfeiffer, M., Pichler, J.: A comparison of tool support for textual domain-specific languages. In: Proceedings of the 8th OOPSLA Workshop on Domain-Specific Modeling. DSM'08, pp. 1–7 (2008). http://www.dsmforum.org/events/dsm08/papers.html
27. Eysholdt, M., Behrens, H.: Xtext: Implement your language faster than the quick and dirty way. In: Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion. OOPSLA '10, pp. 307–309. ACM, New York, NY, USA (2010). doi:10.1145/1869542.1869625. http://doi.acm.org/10.1145/1869542.1869625

Figures

Figure 1 Sample figure title. A short description of the figure content should go here.

Figure 2 Sample figure title. Figure legend text.

Tables

Table 1 Sample table title. This is where the description of the table should go.

	B1	B2	B3
A1	0.1	0.2	0.3
A2	...	..	.
A3	..	.	.

Additional Files

Additional file 1 — Sample additional file title
Additional file descriptions text (including details of how to view the file, if it is in a non-standard format or the file extension). This might refer to a multi-page table or a figure.

Additional file 2 — Sample additional file title
Additional file descriptions text.