Coding Excellence – Domain Specific Language

Bankole Jordan, Charron Maxime
Gummersbach institute of Computer Science and Engineering – TH Köln, Germany 2022
E-mail : jordan.bankole@smail.th-koeln.de maxime.charron@smail.th-koeln.de

Abstract: In this document we propose a presentation and a study of possibilities offered by Domain Specific language (that we will now call DSL). The communications between developers and domain specialist are primordial and DSL is a real answer to that problem. To explain and support our point, we will use the example of DSL in a specific domain and more especially in the field of medicine. Here, our support will be an article about the AGGIR (Autonomie Gérontologique et Groupe Iso Ressources—Autonomy Gerontology Iso-Resources Groups) grid model. Our aim is to provide a clear and understandable explanation of what is and what are the purpose of DSL's.

Keywords: Domain Specific Language, Feature-Oriented Programming, Pervasive Computing

1. Introduction

According to the WHO in 2021 [Disability and Health] , the number of people with disabilities and the elderly has increased significantly. Due to this increase, the amount of sheltering in charge of person with reduced autonomy and especially the effectiveness of care provided in these establishments has greatly increased. Tools have been developed to take care of those people. Among these tools, have measurement tools such as the AGGIR or software allowing classification of people with reduced autonomy.

The next step was to improve this software by making them usable through DSL.
These increasingly advanced DSLs have enabled companies to better take charge of people in need. Since medicine is a very specific field, where the life and autonomy of living beings are at stake, we wondered how the DSL used was conceived.
Indeed, the latter must meet two challenges.
Be very simple to use to be understandable to the medical profession
They must also and above all be precise enough to allow the treatment of patients' data with great precision and flexibility.

With this information, we will discover what the DSL looks like in the world of medicine, we will make a tour of existing technologies. Understand their uses, how they use existing resources such as AGGIR, the advances they have allowed, and we will take one of them as an example to support this article.
Our goal is to understand what we can do with the DSL today in the world of medicine and to see what limits a team can reach (or even push) thanks to the DSL tools

2. The foundations of DSL

DSLs are technical solution to a communication and understanding problem, before the explosion of the use of computers, these problems were taking different forms and had to be solved with different solutions, and even now, you must follow different rules to fill the purpose of DSL properly. In this section, we will explore the different methods and tools that have been used to make the communications between different teams easier and what have been the main element that are now part of the DSL concept no matter it’s form.

2.1 Ubiquitous language

A ubiquitous language is a vocabulary shared by everyone involved in a project, from domain experts to stakeholders, to project managers, to developers. This language should be used in all documentation, tickets, conversations, meeting invites, post-it notes on your computer, chalkboards that only appear in your dreams — everywhere. The goal is to reduce the ambiguity of communication. Second, **a useful shorthand can develop**. Once the ubiquitous language becomes more ingrained, and good habits are established, an enormous amount of information and context is communicated when someone uses the word "article."

This technique can be compared to a “real life” DSL as it allows different part and different profession inside of a same team to communicate more easily. To be efficient, it must follow some rules:

- Must be expressed in the domain model (a system of abstractions that describes selected aspects of a sphere of knowledge, influence, or activity)

- It is not a business language imposed by domain expert, it must be built in collaboration between the domain expert team and the technical team so that everyone clearly understands the purpose and meaning of all the vocabulary and grammar.

- Its purpose is to evolve over time, it’s not defined entirely in a single meeting. Obviously, it must change and grow like the project, the team, the domains…

- Concepts that are not part of the language should be rejected, indeed, inside of an only company or team group, you can have multiple languages and should place clear barrier between domains. However, every term and concept that compose the language must be related to the domain or the technical aspect so that everyone is clear on the meaning of the vocabulary, and you don’t import any off-topic concept.

It's important to keep those rules in mind because they’re all completely applicable to the DSL concept as it is in fact a ubiquitous language on a computer-oriented medium.

2.2 Domain Driven Design

The Domain Driven Design is a way of thinking and building a project on accordance with the expertise domain. The goal of this process is to build a software, the structure, the language, to match the business domain, as theorized by Eric Evans in his book *Domain-Driven Design: Tackling Complexity in the Heart of Software* in 2004, DDD is:

A sphere of knowledge, influence, or activity. The subject area to which the user applies a program is the domain of the software.

But this is a large concept and there is a lot of defini

the goal of DDD is to ease the creation of complex application by connecting the related pieces of software into an evolving model built to follow the rules of the technical domain. First, Eric Evans defines some vocabulary to clarify the work with DDD and assure that you will follow the good practices with this method:

- The **Model** is a system of abstractions that describes selected aspects of a domain and can be used to solve problems related to that domain

- The **Context** is very important, the moment and position in which a word or statement is used determines its meaning. Statement about a model can only be understood with a context.
- The **Ubiquitous Language**

- The **Bounded Context** is a description of a boundary (typically a subsystem or the work of a specific team) within which a particular model is defined and applicable.

Obviously, any vocabulary is useless if it's not applied in a concrete context, DDD should follow three main principles:

- It should focus on the project domain and domain logic.

- The base complex design should follow the models of the domain

- The collaboration with domain experts to improve the application model and resolve any emerging domain-related issues.

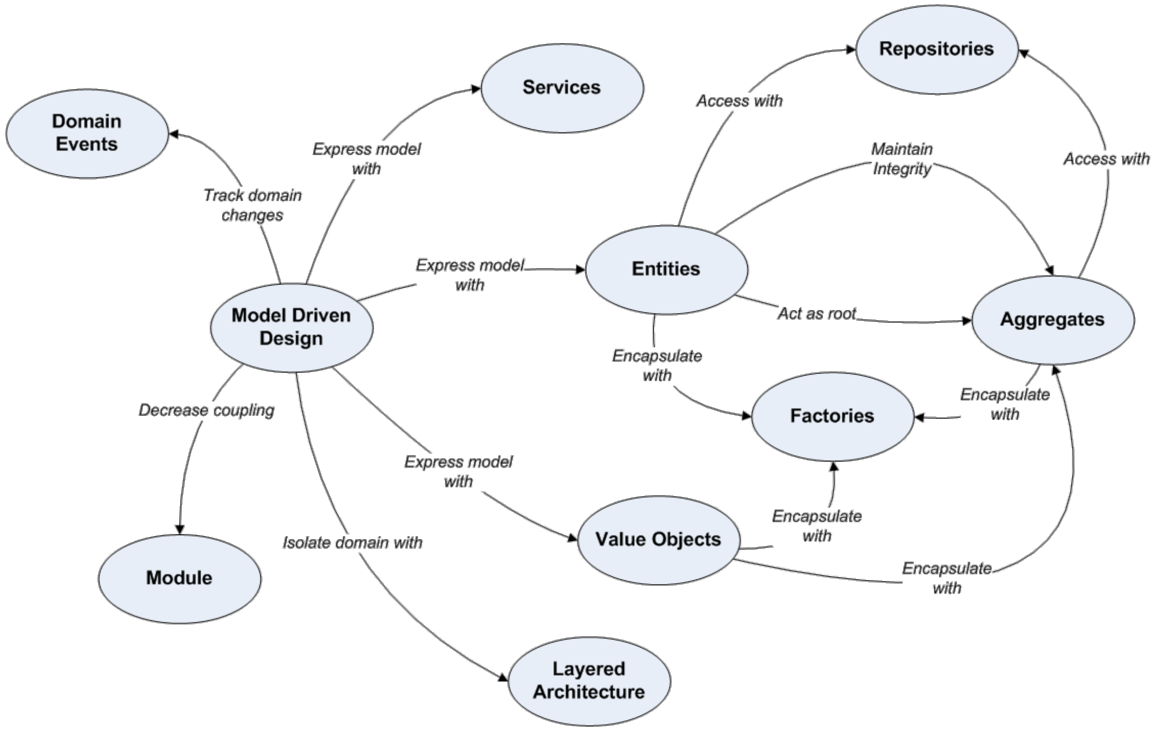
After that, the last important thing about The Domain-Driven Design is to be clear about several high-level concepts to be able to create and modify domain models, these notions are more technical but as important as the domain vocabulary:

- An **Entity** is an object identified by it's consistent, not his attributes

- A **Domain Event** is an object that record a discrete event related to model activity within the system, it's only created for a value or topics that domain experts care about.

- An **Aggregate** is a cluster of entities and value objects with defined boundaries around the group. Now, external objects only have access to the single aggregate root item and use that to pass along instructions to the group as a whole

- the DDD meaning of a **repository** is a service that uses a global interface to provide access to all entities and value objects within a particular aggregate collection.



2. DSL

Now that we have place a context and talked about good practices around the principle of DSL, we will work on explaining what a DSL is, what are the purposes, features that this way of working allows and what are the specificities of building a project based on a domain.

First, we need to define clearly what is a DSL. As there is no consensus on the exact definition of what is a DSL, there is no real historic. The first programming languages (Cobol, Lisp, Fortran) can be considered as DSL even if they're not named as so. These first language were, by definition, dedicated to a specific domain and as the computer science domain was evolving, the problem that this technology could solve have diversified. Because of these new possibilities, new need for specialized languages to solve specific needs continued to be felt. Several solutions have been found:

- The **libraries** provide specialized subroutines for a given application domain.
- The **frameworks** object oriented are like libraries for object-oriented purposes.
- **Domain-specific** languages, which offer the possibility to enable expression in the targeted application domain.

The term "dedicated language" appeared for the first time in September 1984, in a publication by J.M. Neighbors. The idea of dedicated language is then found under different names, in particular "little language" or "micro language" in 1986. In 1996, a conceptualization of the dedicated language is implemented.

2.1 External vs internal DSL

There are two main different types of DSLs in a technical way, internal and external, the distinction between those two principles is clear.

An internal DSL is a DSL that can be defined within an existing general-purpose language. That is, the base language contains the possibility to extend itself with new constructs. Using this you can create a domain concept within the base languages. In the following part, we will look at DSL in a real-life area. This is the field of medicine. We will first make an inventory of the medical field, then we will analyze the contributions of DSL to this field.

An external DSL is a standalone DSL. There is no base language, and the DSL contains nothing more and nothing less than exactly the domain concepts. It then needs a parser to interpret this software to be understood by another software.

For instance, RSpec or Rake are internal DSL, but the term external DSL offers a lot more possibilities, Cucumber, but also CSS, and Sass are external DSLs.

2.2 horizontal vs vertical DSL

A horizontal DSL is a DSL that has a technical domain. As these are technical DSLs, they are usually used by developers to make writing software easier.

A vertical DSL, also called a business DSL is a DSL that is defined for a business domain. Quite often users of such a DSL are businesspeople, who can express what they want in their own business terms.

Most of the time, internal DSLs are considered as horizontal, because they're not made to be seen by people that don't know anything about computer science. Indeed, the format is often close to a library or still need an IDE and en development environment to be worked on as they solve internal technical issues.

As external DSLs offer more liberties on the format and the technologies that you use, they're most of the time used to build a DSL that will be used by domain experts as they may not be familiar with development (Sass…)

2.3 The building process

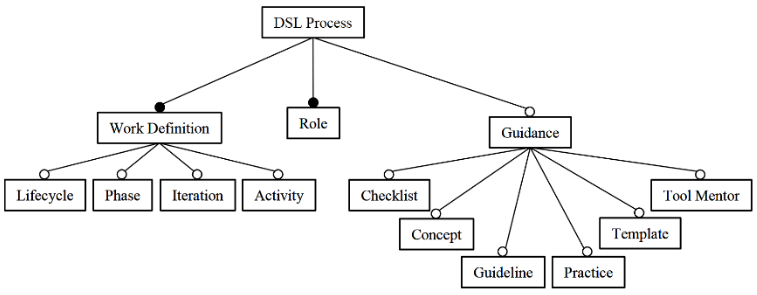
The building process of a DSL is not defined as sequential, but iterative; that means that you must check if everything is ok with de domain expert team at each step of the creation process, the discussion needs to always be open between the team and refactor or principle changes can happen at any time. For the language to be as close as possible to the targeted domain, the construction requires either a double competence (of the application domain and of the language programming), or a close collaboration between the team of programmers and the experts of the treated domain.

The first step to the DSL creation is to understand the Domain, as the goal of a DSL is to let the domain team work on their own, it's primordial to be perfectly clear with the domain experts and their demands. Therefore, a constant and regular follow up with the experts is primordial to ensure the exactitude and complete fulfill of the requirement.

It's important to define some point in the domain and technical field to avoid mistakes and be sure of a good communication.

First, in a domain way, the lifecycle of the project is the field of action of the project and the life span of the software that will result. The phase and the Iteration are the different steps of the project and the number of feedback and follow-up that will be necessary (mid follow-up can obviously bet set up but it's important to implement a calendar to monitor at best the evolution of the project). The activity is the field of action, not of the project, but of the domain experts and the larger problem that will be solved by the DSL.

Then, in a technical way, you have to follow a guidance, a procedure to follow to be able to format your architecture and development process on the Domain way, especially when you're working on a internal DSL as it's supposed to be seen by the domain experts. This "obligation" is less important on some form of external DSLs as the main point of developing them can be to get a user interface or a complete software that will only be seen and used by the domain team from the outside or through an interface.



After the conception of your DSL, you must keep in mind that as the domain/team/technology are evolving and so must your DSL. It's important to have a person from the domain team formed to be able to work on and explain the DSL in case the technical team is changing, or a person from the technical team that stay in the team to train a new technical team.

2.4 advantages and disadvantages

The DSLs show an interesting potential in productivity, comprehensibility, reusability, and communication. They're concise and largely self-documenting and can be reused for different purposes. Application domain experts can more easily modify a program without programming knowledge in general purpose languages.

Internal DSLs offer the possibility to let you use the knowledge and power of the host language and every tool that are available for this language.

In an external dedicated language, the syntax of the language will be able to represent any concept that it will be desirable to use with the targeted domain, allowing the latter to be fully expressed using its own terms and symbols. Since the language is not tied to a host language, it should not suffer from implementation biases or compromises related to the environment in which it will be integrated, especially in terms of syntax or the target platform

Since a dedicated language is, by definition, intended to be used in a defined domain, it may be less easy to find examples of source code to complete the documentation than in the case of general-purpose languages. The cost related to the production of the dedicated language, its maintenance, and the training of users. The difficulty caused by the choice of the ratio in the construction between dedicated languages and generalist languages. People who are not experts in the application domain may find it difficult to read or modify a program written in a dedicated language. The use of dedicated languages can lead to the proliferation of redundant and non-standard languages, which poses compatibility and portability problems. In terms of performance, programs developed in dedicated languages usually use processor time less efficiently than general-purpose languages that generally have compilers that optimize execution or high-performance virtual machines

In the following section, we will look at DSL in a real-life domain. This domain is medicine. We will first make an inventory of the medical domain, then we will analyse the contributions of DSL to this domain.

3 The field of medicine

The recent needs in the field

With the advances in science, we can now afford to take effective care of people with a reduced level of autonomy. Indeed, there are health centers as well as assistance programs to help these people in their daily life. The problem is that as we mentioned earlier in this article, the number of elderly and disabled people has increased dramatically over the last decades. On a global scale, this represents a significant cost. According to a CNBC article (it takes millions), it would cost an American family 3M plus state aid to provide for their autistic child.

Taking this into account, it is easy to understand that one of the challenges of our era is to reduce the costs of care and specially to increase its efficiency.

In this perspective of improving care services, we have seen the emergence of categorizations and classifications of people with a lack of autonomy. Tools and evaluation grids have been designed to have predefined care procedures for all types of patients. It is in this perspective that the AGGIR (Autonomie Gérontologique et Groupe Iso Ressources) appeared. A French rating system allowing a classification of people with reduced autonomy according to many criteria. These criteria include coherence, orientation, toileting, dressing, eating, elimination, transfers, moving around inside, moving around outside and communication at a distance. These criteria will be enriched by remarks

indicating whether the person treated needs assistance in certain areas of life. His/her ability to cook, manage his/her belongings, do housework, take transportation, go shopping, follow medical treatment, engage in activities to occupy his/her

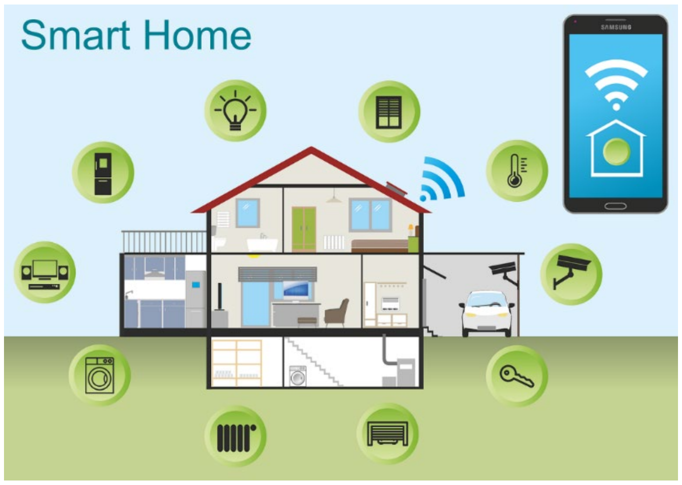
free time. All these criteria enrich the patient's categorization information (How does the AGGIR grid work?)

This tool, which was originally developed in France, has been used in hospitals abroad, such as in Belgium, since 2005. A reference to it can be found in a HAL Open Science study. A journal focused on medicine.

GIR	degree of dependency
GIR 1	Person confined to a bed or chair, whose mental functions are seriously impaired and who requires the continuous and indispensable presence of caregivers -- Person at the end of life
GIR 2	Person confined to a bed or chair, whose mental functions are not totally impaired and whose condition requires care for most activities of daily living -- Person whose mental functions are impaired but who can move about and requires constant supervision
GIR 3	Person who has retained his mental autonomy, partially his locomotor autonomy, but who needs daily and several times a day assistance for body care
GIR 4	A person who is not able to transfer on their own but who, once up, can move around inside their home, and who needs help with washing and dressing -- Person who does not have locomotor problems but who needs help with personal care and meals
GIR 5	People who only need occasional help with bathing, meal preparation and cleaning
GIR 6	Person still autonomous for the acts of everyday life

An example of application

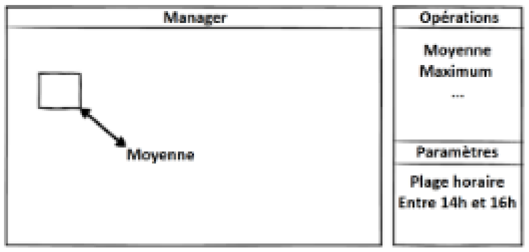
As the usefulness of this tool was recognized, processes were put in place to calculate the level of autonomy of patients. In 2014, a project was launched. Placing a patient whose level of autonomy we want to know in a smart home. This smart home named iCasa is a house filled with sensors that give permanent information on the equipment of the house and its resident. The position of the resident in the house, the lights on or off, the chairs on which he sits and for how long. These data together allow us to determine the habits of the resident, to see which tasks require the most effort and to see which ones he tends not to perform (potentially because of his health condition). From the data collected and what we learn about the resident's daily life, we can deduce their level of autonomy.



3.2 The contributions of the DSL in medicine

An example of DSL use

The results of the previous project are analysed by members of the medical profession. To make this analysis easier, a DSL in the form of a graphical interface has been added. Through this interface, the medical staff can obtain the resident's data at any time. This data can be from any time of the day and from any equipment in the house equipped with sensors. It becomes easier for those untrained caregivers on home technologies to report on what is going on.



Display of results

Depending on the parameters selected and the type of operation requested, the type of display will not be the same.

The doctor can ask for an average of the values of a sensor over a period, a maximum duration of activation, or a minimum duration. The idea is to allow staff who are not familiar with the equipment in the house to be able to use all the capabilities it offers through this DSL interface.

What it brought to the experience

This DSL was not just a display. Indeed, the display was used to control the calculations made by the program. This last one by gathering the information of the sensors manages to detect events in the kitchen. A press on a chair after the activation of the microwave and the computer records that the patient has heated a meal and is eating on a chair. All these events are stored and allow the computer to deduce not only the patient's habits but also irregularities and time-consuming tasks.

The fact of obtaining the habits of the patient finally allows to know what he can do alone and thus to calculate automatically the AGGIR values of the latter.

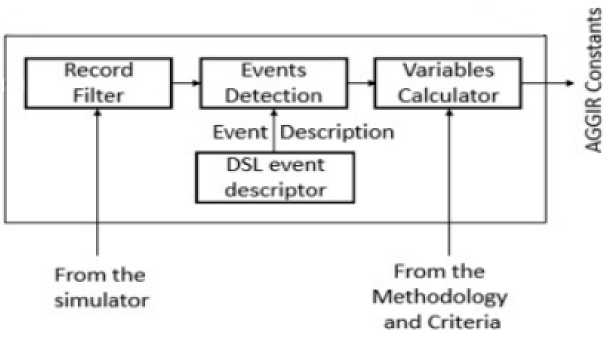
This saves time for the medical staff. Moreover, it has been proven that the choice of a good level of care for the patient was very determinant on his length of stay in a hospital (Geriatric patients:

AGGIR PATHOS SOCIOS, a tool to consider the specificity of the geriatric patient in hospital financing complementary to APRDRG), thanks to its calculations, it is simpler to decide on the intensity of the care and to allow a better rehabilitation to the patient.

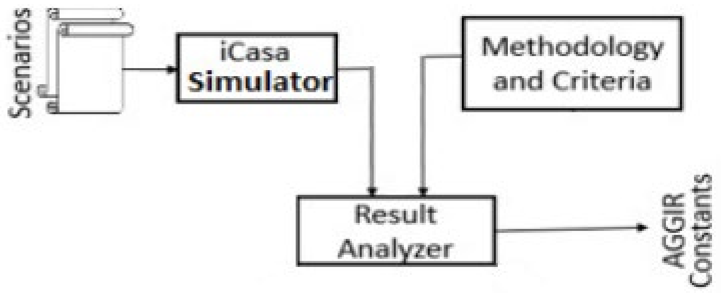
The Role of the DSL in this Operation

Within this project, the DSL had several uses

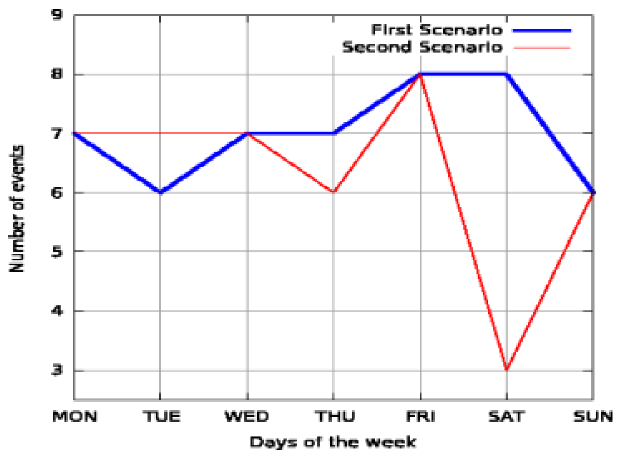
- An automated calculation of the patient's AGGIR constants
  Thanks to the variables, it is possible to understand what the patient does (the events) and to deduce the value of these constants. We save time in this process and reduce the possibility of human error. Indeed, the human being here acts only as a supervisor. He defines the events and verifies the consistency of the calculated constants



- Allow the creation of simulation
  It is possible to use the simulator to create scenarios and train the artificial intelligence or to test the event detection criteria.



- Enable customizable visualization of patient behaviour
  With the user interface created, a caregiver can display on his screen the data from a sensor according to selected parameters. With the right parameters, he can easily analyse his patient's habits, detect anomalies, and make graphs on the evolution of a patient's constant. Here is an example

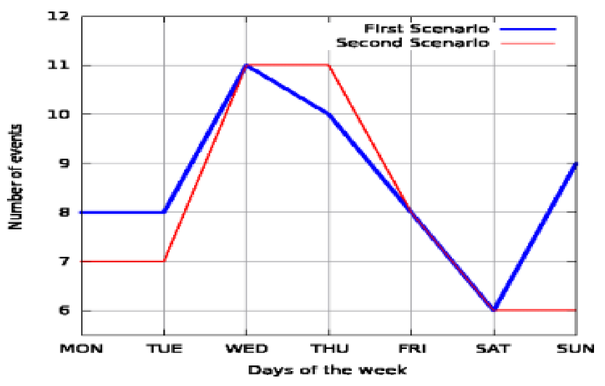


Transfer events

Here we see a graphical representation of the results of the simulation. The caregiver does not need to look at numerical values returned by the sensors to detect that in the first scenario, the patient has a better ability to move than in scenario 2.

- Enable detection of habit changes

  In the example below, we see the number of hygiene-related events over a week. It is clearly visible without having to analyse any figures that on Saturdays, the residents devote fewer events to their hygiene.



Hygiene events

The DSL here enables fast, efficient, and accurate analysis of collected data.

These graphs show us one thing, that there is no need to know anything about hardware, or how the smart-home works, to collect and analyse data. The data is still usable and understandable for any user in the medical field. This usage perfectly highlights the usefulness of a DSL.

This way, the domain specialists can work with a technical solution without the help

of the technical team. The enterprise can increase its efficiency, the quality of nurse and lighten the workload of his employees.

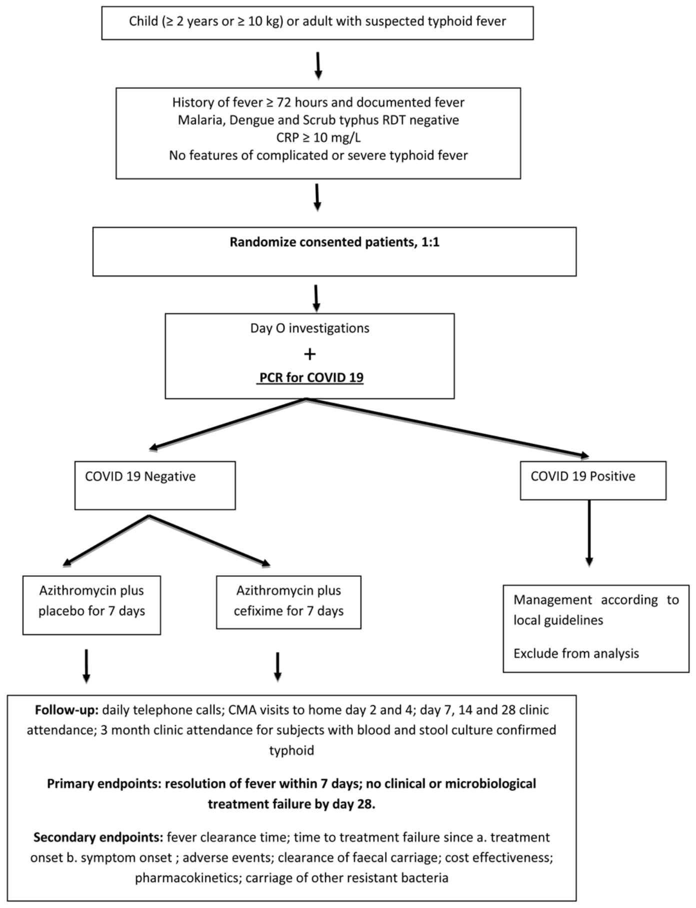
4 Our project

The main questions of our report are:

- What are the problems worth using a DSL to solve them?
- What are the possibilities with DSL for a limited time and Team?

We know that in the medical field, a patient treatment always follows a protocol. these medical protocols vary greatly depending on the patient and his condition.

For instance, this picture shows the protocol for testing and treating the Covid 19



Child (≥ 2 years or ≥ 10 kg) or adult with suspected typhoid fever

History of fever ≥ 72 hours and documented fever
Malaria, Dengue and Scrub typhus RDT negative
CRP ≥ 10 mg/L
No features of complicated or severe typhoid fever

Randomize consented patients, 1:1

Day O investigations
+
PCR for COVID 19

COVID 19 Negative

COVID 19 Positive

Azithromycin plus placebo for 7 days

Azithromycin plus cefixime for 7 days

Management according to local guidelines

Exclude from analysis

Follow-up: daily telephone calls; CMA visits to home day 2 and 4; day 7, 14 and 28 clinic attendance; 3 month clinic attendance for subjects with blood and stool culture confirmed typhoid

Primary endpoints: resolution of fever within 7 days; no clinical or microbiological treatment failure by day 28.

Secondary endpoints: fever clearance time; time to treatment failure since a. treatment onset b. symptom onset ; adverse events; clearance of faecal carriage; cost effectiveness; pharmacokinetics; carriage of other resistant bacteria

Here, the treatment depends on the patient's history of fever, his PCR tests and his age.

For each variation of those variables, the protocol change. Both the treatment length and content differ from one situation to another.

The idea that we had is to create DSLs to automate the creation of a patient treatment.

We will record some medical protocols and create programs that will ask for the patient's information. Based on them, the DSLs will create a profile and compute the patient's whole treatment following a protocol.

Another thing we want to do is to compare different DSL, to benchmark their efficiency and speed. To do that, we will create one DSL for each protocol recorded and build them using different tools. Those DSLs will be created with the same architecture and will works the same ways.

We already thought about the tools we will be using

- MPS by JetBrains
  This tool created by the JetBrains company is an IDE for DSL creation. It offers a lot of great features. Among which: an auto-complete based on the DSL we are creating, the possibility to compile our DSL in different programming languages (useful for a complete and rich benchmark), the possibility to create a language for the code generation we talk just before and a great user interface

- ANTLR

  ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. It's widely used to build languages, tools, and frameworks. From grammar, ANTLR generates a parser that can build and walk parse trees.

With those tools we can as many DSL as we want. We just need to record a few medical protocols, create our DSL architecture and build them. The last step will then be to create the process to get the patient's information and compare our DSL when a patient information is sent.

5.Conclusion

The DSL is a powerful way to build a project, but you must be careful about how, why, for who you build your DSL, indeed there is so much different way to build a lot of different form of DSL depending on the level of the user, the domain that you're studying and many other things. As this tool is very powerful, it also has some disadvantages, economical, technical and evolution limitations.

A lot of things and practices can help you to work on a Domain oriented way, the ubiquitous language and the data driven design are very useful and almost necessary to correctly build a DSL in the best conditions. For our project, we will try to explore different tools to see what a limited team in a limited amount of time is able to do and then answer de question "What are the problems worth using a DSL to solve them?" by trying to find the limits of DSL for a small team.

REFERENCES

CNBC (2017) *it takes millions to care for your special needs child heres how to do it*, https://www.cnbc.com/2017/06/02/it-takes-millions-to-care-for-your-special-needs-child-heres-how-to-do-it.html

Portail national d'information pour les personnes âgées et leurs proches (2022) "*Comment fonctionne la grille AGGIR ?",* https://www.pour-les-personnes-agees.gouv.fr/preserver-son-autonomie-s-informer-et-anticiper/perte-d-autonomie-evaluation-et-droits/comment-fonctionne-la-grille-aggir

BHC Health Serv Res (2008) "*Geriatric patients: AGGIR PATHOS SOCIOS, a tool to take into account the specificity of the geriatric patient in hospital financing complementary to APRDRG*", https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3313307/

HAL Open Science (2020) *"Proposal and Validation of a Domaine Specific Language for the Representation of the AGGIR Constants", https://hal.archives-ouvertes.fr/hal-02133841/document*

Matt Robison Naming (2020) *Content Types Using a Ubiquitous Language,* Drupal Development & Drupal Site Building

Domain model Wikipédia page (2015 - 2022) https://en.wikipedia.org/wiki/Domain_model

Felipe De Freitas Batista (2019) *developping the ubiquitous language,* https://thedomaindrivendesign.io/developing-the-ubiquitous-language/

Frances Banks (2022) *Domain-Driven Design: What is it and how to use it ?,* https://blog.airbrake.io/blog/software-design/domain-driven-design

WHO (2021) *Disability and health,* https://www.who.int/news-room/fact-sheets/detail/disability-and-health

Tutorial Domain Driven design (2015), *building block Domain Driven Design, http://uniknow.github.io/AgileDev/site/0.1.8-SNAPSHOT/parent/ddd/core/building_blocks_ddd.html*

https://fr.wikipedia.org/wiki/Langage_d%C3%A9di%C3%A9

Jos Warmer (2018) *Domein specific Language,* https://www.openmodeling.nl/dsl.html

Charles Consel, Fabien Latry, Laurent Réveillère, and Pierre Cointe (2005) *A Generative Programming Approach to Developing DSL Compilers*

https://hal.archives-ouvertes.fr/hal-00350045/document