Describing Dynamic Organizational Structures through the Common Lisp Object System

Lennard Wolf
lennard.wolf@student.hpi.de

Hasso Plattner Institute
Prof.-Dr.-Helmert-Straße 2-3
14482 Potsdam
Germany

Abstract. Describing Dynamic Organizational Structures through the Common Lisp Object System Describing Dynamic Organizational Structures through the Common Lisp Object System Describing Dynamic Organizational Structures through the Common Lisp Object System Describing Dynamic Organizational Structures through the Common Lisp Object System Describing Dynamic Organizational Structures through the Common Lisp Object System Describing Dynamic Organizational Structures through the Common Lisp Object System Describing Dynamic Organizational Structures through the Common Lisp Object System

1 Introduction

My notations

– "term" : wording unsure

The following is a summary of each section. In section...

2 Problem

– To gain an overview of the "workings" within organizational structures it is often useful to have the participants and their "occupations"/"roles" written down somewhere.
– say a system needs all participants to be objects
– their occupations would define their possible interactions, rights etc
– in many structures will not only the people come and leave, but they might also change their occupations
– example case university: professors can also be "CEOs", students can also be tutors etc
– how would it be possible to "abbilden" this within a software system, so that it is easily maintainable - meaning that individuals can change their jobs etc - and that it is easily extendable - meaning that new occupations can easily added and combined with already existing ones

3 Context

A "solid" solution to the problem introduced in Section 2 would be to use the *Common Lisp Object System* (CLOS), an object system that extends Common Lisp[1] to support object-orientation. [**?**] However to understand how it could be employed and why it would be "a good idea" to do so, some background information will need to be provided first.

In this Section we will hence start out by introduce the general ideas behind object-orientation as well as the programming language Lisp, and then we will introduce the three main concepts that make CLOS, an for Lisp, unique. Afterwards an overview of the history of CLOS will be provided to better understand its origins.

3.1 Object-Orientated Languages

A definition of *object-orientated* programming languages requires a preceding explanation of the terms *object*, *class*, and *inheritance*. *Objects* have a set of *attributes* that define its *state*, as well as a number of *messages* that it can receive to evoke certain behavior. By sending such messages, objects can interact with one another. *Classes* are templates for objects, so that objects of the same class can have uniform interfaces and behavior. *Inheritance* opens up the possibility to create class hierarchies, so that classes can be specializations of others and *inherit* certain behavior while adding something unique.

An *object-orientated* programming language is one that has these three concepts "integrated". They give programmers the ability to model human language based conceptualizations of the real world, since these are also just seperations of phenomena into groups with common traits.

3.2 Lisp

Stemming from the term **Lis**t **P**rocessor, Lisp is a programming language in which *everything is a list*. This means that there is *no discernment between data and code*. Hence an expression such as `(plus 3 4)` is without context nothing more than a "meaningless" listing of the "meaningless" expressions `plus`, `3`, and `4`. An expression like that only gets its meaning from an *evaluation* by the interpreter which considers it in a certain context. In such a context the expression `plus` could be a function to add the arguments it is given, but that is entirely arbitrary.

Common Lisp is a standardized version of Lisp which provides certain data types and operations. But this type system can be extended through the use of *macros*, which let developers give meaning to expressions. This feature of Lisp makes it "easy" to create *Domain Specific Languages*[2]. [**?**] They also form the basis of CLOS which in its entirety consists of 8 macros and 33 functions.

– explain Macros in detail?

[1] `https://common-lisp.net`, accessed: XXXXXX
[2] XXXX whats an DSL??

3.3 History of CLOS

- Many implementations ("Tower of Babel" situation) (Flavors, CommonLoops)
- 1986: Workgroup of different researchers from Xerox PARC (CommonLoops) and Symbolic Inc. (New Flavors) worked on it together
- CLOS: System built on top of Common Lisp
- meta-object protocol - was das
- CLOS is portable to different LISP implementations

3.4 Main Concepts

Generic Functions blablablablablablablablablablablabla

Multiple Inheritance blablablablablablablablablablablabla

Method Combination blablablablablablablablablablablabla

4 Approach

- 3 concepts will be used
- maybe give function that lets user add classes

5 Implementation

- give code details for concepts from Approach

6 Evaluation

- DSL type declaration is easily maintainable
- using langs like Java (?) might be problematic, give example

7 Conclusion

Write me pl0x