Formalizing OO Frameworks and Framework Instantiation

Christiano de O. Braga, Marcus Felipe M. C. da Fontoura, Edward H. Hæusler, and Carlos José P. de Lucena

Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 225, 22453-900 Rio de Janeiro, Brazil
e-mail: [cbraga, mafe, hermann, lucena]@inf.puc-rio.br[1]

ABSTRACT

Case studies have shown that high levels of software reuse can be achieved through the use of object-oriented frameworks, instantiating them to create specific applications. The most common way to instantiate a framework is to inherit from some abstract classes defined in the framework hierarchy and write the code that is called by the framework itself. However, it is not always easy to identify which code and where it should be written since frameworks class hierarchies can be very complex, especially for non-expert users. This paper presents formalization of the concepts of frameworks and framework hotspots using the General Problem Theory proposed by Veloso and Veloso. Through this formalization a domain specific language (DSL) can be defined to specify the framework's flexible parts.

KEY WORDS: object-oriented frameworks, framework kernel, hotspots, problem theory, domain specific languages.

1. INTRODUCTION

Prior research has shown that high levels of software reuse can be achieved through the use of object-oriented frameworks [13]. An object-oriented framework captures the common aspects of a family of applications. It also captures the design experience required while producing applications, and thus, it allows designers and implementers to reuse their experience at the design and code levels.

However, the "common" instantiation process, which is based on inheriting from abstract methods, is not always the ideal way to describe a particular application. This happens because of several facts:

- the language used to build and use the framework is a wide spectrum language, where it is not always easy to express the user intentions [11]. A short example would be the definition of event handlers in Microsoft MFC. The user has to write a macro to register the methods that will be called when a button is pressed or a list is scrolled. With an interface definition language [7], this task becomes easier and more intuitive;
- the complexity of framework class hierarchies and the difficulty of finding the points where the code should be written, that are the framework hotspots or flexible points [10].

This paper proposes a formalization for frameworks, framework hotspots [10] and framework instantiated applications. The formalization is based on Problem General Theory [12], using the ideas of problems, generic problems, and a new concept called meta-problem.

[1] This work has been partially sponsored by the Brazilian Government Educational Ministry PRONEX project.

2. PROBLEMS, GENERIC PROBLEMS AND META-PROBLEMS

Maybe the concept of problem is one of the most familiar to humans. However, there is not a common agreement of what a problem is. This is because this concept, as well as most of the concepts in our language, is not formalized [12, 4].

For a scientific study of problems, and their relation to frameworks, we will try to give a formal definition to problems. The definition used here is the same presented in [4], and it is based on three questions [9]:

- What are the possible inputs?
- What are the possible outputs?
- What is the condition of the problem?

The condition of the problem is a precise definition of when an element is really an output, given a predefined input. The specification of the problem condition is always necessary, since it determines the nature of a formalized problem.

We will also consider two kinds of problems: generic and specific. An example of a generic problem is matrix multiplication. A specific problem could be the multiplication of two predefined matrixes.

Definition 1: A generic problem P is a 2-sorted structure (D, R, q), where:

- D is a non-empty set, called data domain;
- R is a non-empty set, called result domain;
- q is a binary relation from D to R, called problem condition.

Definition 2: A solution s of a problem P is a total function from D to R that associates to each element $d \in D$ a result s(d) that satisfies the condition of the problem. That is, $\forall d \in D$ $(d, s(d)) \in q$.

We will define meta-problem as a generic problem where the problem condition cannot be specified. Instead of specifying the problem condition, in a meta-problem we specify some problem properties that will restrict the problem condition for all the problems derived from that meta-problem. One meta-problem defines a class of problems, which are related through the meta-problem properties.

Definition 3: A meta-problem MP is a generic problem (D, R, Π), where:

- Π, called meta-problem properties, is a formal specification that may restrict the possible problem conditions.

$$\Pi : D \times R \rightarrow \{0,1\}$$

Definition 4: A meta-problem MP = (D, R, Π) defines a class of derived problems DP_{MP}, as a set of problems that satisfies the meta-problem properties and has the same data and result domains.

$$DP_{MP} = \{(D_i, R_i, q_i) \mid \forall i, D_i = D, R_i = R, q_i \in \{q \mid \Pi(q)\}\}$$

The process of specifying the problem condition for a derived problem is called meta-problem instantiation. Figure 1 shows the relation between a meta-problem and it's derived problems, as well as the instantiation process.

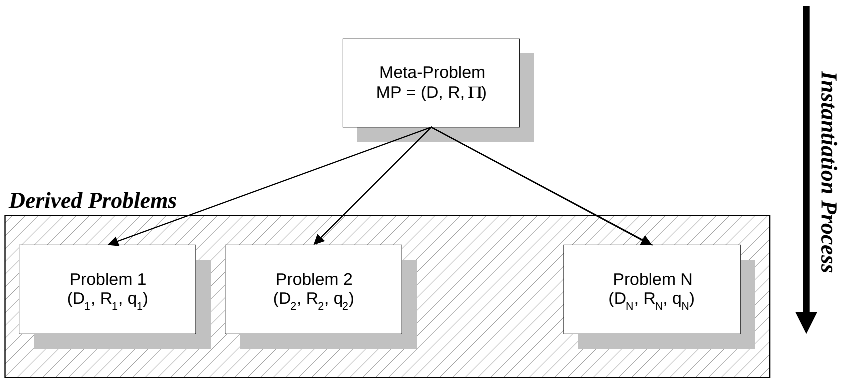

Figure 1: Meta-Problems, Derived Problems, and Meta-Problem Instantiation Process

3. FORMALIZING FRAMEWORKS, HOTSPOTS AND INSTANTIATED APPLICATIONS

The approach to framework construction presented in this paper is based on the idea that any framework can be divided into two parts: the Kernel sub-system and the Hotspot sub-system [10]. The Kernel sub-system is common to all applications that the framework may generate, and the Hotspot sub-system describes the different characteristics of each application that can be supported by the framework. The Hotspot sub-system uses the method and the information provided by the Kernel sub-system and may extend it.

The framework kernel can be seen as a function that "connects" the hotspot components, and implements the behavior that is common to any application derived from the framework. A hotspot can be seen as a meta-problem, which specifies the possible variations of a framework flexible point. The framework Hotspot sub-system is formalized as a tuple of meta-problems. The actual framework definition is the integration of the Hotspot sub-system and the Kernel sub-system.

Definition 5: A framework is an structure with two components: the first component is a tuple of meta-problems, characterizing the hotspots (Hotspot sub-system) and the second component is a function that takes derived problems as parameters and returns a generic problem, which characterizes the framework Kernel sub-system.

$$\text{Framework} = \langle mp, \text{kernel}\rangle, \text{ where:}$$

$$mp = \langle MP_1, MP_2, \ldots, MP_n\rangle$$

$$\text{kernel} = DP_{MP1} \times DP_{MP2} \times \ldots \times DP_{MPn} \rightarrow P$$

Definition 6: An instantiated application consists of the kernel of the framework with the hotspots filled with application specific behavior.

$$\forall P_i \in DP_{MPi}$$

$$\text{Application} = \text{kernel}(P_1, P_2, \ldots, P_n)$$

Thus, the instantiation process consists of the selection of a derived problem for each framework hotspot. Note that the set of derived problems is possibly infinite.

We may also state that to every framework there is an associated meta-problem where its Π predicate has its extension given by Range(kernel) $\subseteq$ P. The framework may instantiate several different applications. This group of applications defines the derived problems of the associated meta-problem. Figure 1 can be used to illustrate this statement by looking at the drawing from the leaves to the top. Each problem in the figure represents an application instantiated from the framework and there is an associated meta-problem to that class of problems. Thus, we have proven that there is a meta-problem associated with the framework.

4. EXAMPLE

Let us consider a framework for the problem of managing academic symposiums. To simplify the problem we will consider only two hotspots, one for registering participants and other for delivering material to already registered participants.

MP Registration: This hotspot is responsible for registering participants in the symposium. The participants must provide the registration date, the registration fee, and the payment option. The framework checks the information and generates a registration number as output.

$D = \text{Date} \times \text{Value} \times \{\text{credit\_card}, \text{cash}\}$

$R = \text{Integer}$

$$\Pi(d, v, p, i) = \begin{cases} 1 \text{ if } d < \text{Max\_Date} \wedge \text{Min\_Val} < v < \text{Max\_Val} \wedge i > 0 \\ 0 \text{ otherwise} \end{cases}$$

Max_Date is the value of the maximum registration date and Min_Val and Max_Val are respectively the minimum and maximum registration fee allowed.

The meta-problem properties imposes some restrictions over the input values and states that the output registration number must be greater than 0. We also have the restriction that the registration numbers must be unique, as formalized bellow.

$\Pi(d, v, p, i) = 1 \wedge \Pi(d', v', p', i) = 1 \Rightarrow d = d' \wedge v = v' \wedge p = p'$

Two possible instantiation cases (derived problems) are considered for this meta-problem: the first is when the participant is a student and the second is when he or she is not a student.

DP_1 for MP Registration – Student: Let us consider that the students can register using credit card until 10 days before the maximum date and from there until the last registration day they have to pay in cash. Let us also consider that the registration fee is always the minimum fee for students. These restrictions are formalized as follows.

$(d, v, p, i) \in q \Leftrightarrow (d > \text{Max\_Date} - 10 \wedge v = \text{Min\_Val} \wedge p = \text{cash}) \vee (d \leq \text{Max\_Date} - 10 \wedge v = \text{Min\_Val})$

DP_2 for MP Registration – Non student: Non students have different restrictions, as formalized bellow.

$(d, v, p, i) \in q \Leftrightarrow (d > \text{Max\_Date} - 10 \wedge v = \text{Max\_Val} \wedge p = \text{cash}) \vee (d \leq \text{Max\_Date} - 10 \wedge v = \text{Min\_Val})$

MP Material Delivery: This hotspot models the delivering of tutorial material to already registered participants. It takes the registration number and a list of the tutorials that the participant will attend as input and returns the number of tutorial notes delivered as output.

$D = \text{Integer} \times \text{Tutorial}^*$

$R = \text{Integer}$

$$\Pi(\mathrm{i, t, n}) = \begin{cases} 1 \text{ if } 0 < \mathrm{i} < \text{Last\_Registration\_Number} + 1 \\ 0 \text{ otherwise} \end{cases}$$

Last_Registration_Number is the value of the last symposium registration number, Tutorial is the set $\{t_1, t_2, \ldots, t_{n\_tutorial}\}$, and n-tutorial is the number of available tutorials.

The only restriction that the meta-problem imposes is that the registration number must be a valid one. As an example, let us consider two possible instantiations for this hotspot.

DP_1 for MP Material Delivery – First 50 participants: This hotspot is used for defining a rule that the first 50 participants that have been registered for the symposium have the right to receive notes for all the tutorials. The operator # returns the cardinality of the set.

$(i, t, n) \in q \Leftrightarrow (0 < i < 51 \wedge n = n\_tutorial) \vee (i > 50 \wedge n = \#t)$

DP_2 for MP Material Delivery – More than 4 tutorials: This hotspot is used for defining a rule that if a participant register for more than 4 tutorials he or she receives one more tutorial note for free.

$(i, t, n) \in q \Leftrightarrow (\#t < 5 \wedge n = \#t) \vee (\#t > 4 \wedge n = \#t + 1)$

Framework structure: The combination of the hotspots (flexible parts) and the Kernel sub-system (not flexible part) provides the structure of this simple framework. Validation of credit card and selection of tutorials, for example, are not a hotspots for this framework and, for this reason, belong to the Kernel sub-system. The function kernel formalizes the framework design structure:

```
kernel($DP_{Registration}$, $DP_{Material Delivery}$) {
        let (data, fee, payment, number) $\in$ $DP_{Registration}$
        if (payment = credit_card) then P-ValidateCard endif
        let (number, list) $\in$ P-SelectTutorialList
        let (number, list, notes) $\in$ $DP_{Material Delivery}$
        …
```

}, where P-ValidateCard is the problem of validating credit card information and P-SelectTutorialList is the problem of selecting the tutorial list from the available tutorials.

5. CONCLUSIONS AND FUTURE WORK

This paper formalizes the concepts of framework, framework kernel, framework hotspots, and framework instantiated application. We now want to investigate the use of domain specific languages (DSLs) [1, 3, 5, 8] to help the framework instantiation. We argue that DSLs designed precisely for the framework domain may help the domain expert to instantiate a framework in a more straightforward way. To implement this approach transformational systems [6, 2] may be used to transform DSLs specifications into framework instantiation code. It is important to note that DSLs could be transformed into other DSLs, thus creating a domain network, in a way similar to the one described in [8], providing an implementation path for new DSLs.

Note that various DSLs could be developed to the same meta-problem and the semantics of the DSL must respect the associated meta-problem properties Π. Two major points have to be further investigated:

- the definition of the kernel function;
- the DSL design process, from the hotspots specifications.

6. REFERENCES

1. J. Bell et al, “Software design for reliability and reuse: A proof-of-concept demonstration”, In TRI-Ada ’94 Proceedings, pages 396-404, ACM, Nov. 1994.
2. J. Cordy and I. Carmichael, “The TXL Programming Language Syntax and Informal Semantics”, Technical Report, Queen’s University at Kinkston, Canada, 1993.
3. E. W. Dijkstra, “The humble programmer”, Communications of the ACM, 15(10), Oct. 1972.
4. M. Endler, “O Método da Redução e a Decomposição de Problemas: Alguns Aspectos”, Dissertação de Mestrado, Departamento de Informática, PUC-Rio, 1987 (in Portuguese).
5. P. Hudak, “Building Domain-Specific Embedded Languages”, Computing Surveys, 28A(4), ACM, 1996.
6. J. C. S. P. Leite, M. Sant’anna, and F. G. Freitas, “Draco-PUC: a Technology Assembly for Domain Oriented Software Development”; Proceedings of the 3rd IEEE International Conference of Software Reuse, 1994.
7. C. Levy, D. Cowan, C. J. Lucena, M. Gattass, and L. H. Figueiredo, “IUP/LED: A Portable User Interface Tool”, Software Practice and Experience (accepted for publication).
8. J. M. Neighbors, “The Draco Approach to Constructing Software from Reusable Components”, IEEE Transactions on Software Engineering, vol. 10, no.5, Sept. 1984.
9. G. Polya, “How to solve it: A new aspect of the mathematical method”, Princeton University Press, Princeton, 1957.
10. W. Pree, “Design Patterns for Object-Oriented Software Development”, Addison-Wesley, 1995.
11. C. Simonyi, “The Death Of Computer Languages, The Birth of Intentional Programming”, Technical Report MSR-TR-95-52, September 1995.
12. P. A. S. Veloso and S. R. M. Veloso, “Problem Decomposition and Reduction: Applicability, Soundness, Completeness”, Trappl, R. (ed.) Progress in Cybernetics and Systems Research, Vol. 8, 1981, Hemisphere Publ. Co., Washington DC, p. 199-203.
13. R. Wirfs-Brock and R. Johnson, “Surveying current research in object-oriented design”, Communications of the ACM, 33(9), 1990.