Оригинальная статья / Original article

https://doi.org/10.21869/2223-1536-2024-14-3-223-237

УДК 004.451

Концептуальная модель и реализация объектно ориентированной системы с помощью метапрограммирования

А. А. Чаплыгин[1] ✉

[1] Юго-Западный государственный университет
ул. 50 лет Октября, д. 94, г. Курск 305040, Российская Федерация

✉ e-mail: alex_chaplygin@mail.ru

Резюме

Цель исследований *заключается в разработке концептуальной модели объектно ориентированной системы и ее реализации с помощью функционального языка с поддержкой метапрограммирования.*

Методы. *Была разработана концептуальная модель объектно ориентированной системы на основе динамической объектной системы с передачей сообщений и динамической диспетчеризацией. Разработанная объектно ориентированная система включает в себя функции создания и удаления классов, создания и удаления объектов-экземпляров класса, функции для работы со свойствами объектов, функции добавления методов в класс и передачи сообщений. Также возможно наследование, когда дочерний класс будет иметь свойства родительского класса и свои собственные свойства.*

Результаты. *Объектно ориентированная система была реализована с помощью набора макросов и вспомогательных функций. В качестве базовой структуры данных выступила хеш-таблица с возможностями доступа по ключу, которая также была реализована с помощью макросов. При создании класса автоматически генерируется конструктор экземпляра с установкой начальных значений полей объекта. Наследование реализовано как включение полей родительского класса в дочерний объект. Полиморфизм реализован как динамическая диспетчеризация: при обработке сообщения происходит поиск метода-обработчика. Полученная объектно ориентированная система вместе с модулем хеш-таблиц занимает 97 строк.*

Заключение. *В результате работы была создана и реализована концептуальная модель объектно ориентированной системы, включающей в себя основные функции объектно ориентированной парадигмы программирования. Было показано, что с помощью макросов можно создать компактную объектно ориентированную систему, при этом в интерпретатор функционального языка не нужно добавлять никаких новых функций или возможностей.*

Ключевые слова: *объектно ориентированное программирование; концептуальная модель; интерпретатор; Common Lisp; макрос.*

Конфликт интересов*: Авторы декларируют отсутствие явных и потенциальных конфликтов интересов, связанных с публикацией настоящей статьи.*

Для цитирования: Чаплыгин А. А. Концептуальная модель и реализация объектно-ориентированной системы с помощью метапрограммирования // Известия Юго-Западного государственного университета. Серия: Управление, вычислительная техника, информатика. Медицинское приборостроение. 2024. Т. 14, № 3. С. 223–237. https://doi.org/ 10.21869/2223-1536-2024-14-3-223-237

Поступила в редакцию 14.07.2024 *Подписана в печать 12.08.2024* *Опубликована 30.09.2024*

Concept model and realization of object orientied system with metaprogramming

A. A. Chaplygin[1] ✉

[1] Southwest State University
50 Let Oktyabrya Str. 94, Kursk 305040, Russian Federation

✉ e-mail: alex_chaplygin@mail.ru

Abstract

The purpose of the research *is to develop a conceptual model of an object oriented system and implement it using a functional language with metaprogramming support.*

Methods. *A conceptual model of an object oriented system was developed based on a dynamic object system with message passing and dynamic dispatch. The developed object oriented system includes functions for creating and deleting classes, creating and deleting object instances of a class, functions for working with object properties, functions for adding methods to a class and passing messages. Inheritance is also possible, when a child class will have the properties of the parent class and its own properties.*

Results. *The object oriented system was implemented using a set of macros and helper functions. The basic data structure was a hash table with key access capabilities, which was also implemented using macros. When a class is created, an instance constructor is automatically generated, setting the initial values of the object's fields. Inheritance is implemented as the inclusion of fields of a parent class in a child object. Polymorphism is implemented as dynamic dispatch: when processing a message, a handler method is searched. The resulting object oriented system, together with the hash table module, occupies 97 lines.*

Conclusion. *As a result of the work, a conceptual model of an object oriented system was created and implemented, which includes the main functions of the object oriented programming paradigm. It has been shown that macros can be used to create a compact object oriented system without the need to add any new functions or capabilities to the functional language interpreter.*

Keywords: *object oriented programming; conceptual model; interpreter; Common Lisp; macro.*

Conflict of interest: *The Authors declares the absence of obvious and potential conflicts of interest related to the publication of this article.*

For citation: Chaplygin A.A. Concept model and realization of object orientied system with metaprogramming. *Izvestiya Yugo-Zapadnogo gosudarstvennogo universiteta. Serija: Upravlenie, vychislitel'naja tekhnika, informatika. Meditsinskoe priborostroenie = Proceedings of the Southwest State University. Series: Control, Computer Engineering, Information Science. Medical Instruments Engineering*. 2024;14(3):223–237. (In Russ.) https://doi.org/10.21869/2223-1536-2024-14-3-223-237

Received 14.074.2024 *Accepted 12.08.2024* *Published 30.09.2024*

Введение

Объектно ориентированное программирование (ООП) – это современная парадигма программирования, которая используется почти во всех современных языках программирования [1]. Основная концепция ООП – это объекты, которые содержат данные в виде полей и код в виде процедур (методов) [2]. Программы в ООП разрабатываются как множество

объектов, взаимодействующих друг с другом [3].

Впервые термин «объект» стал употребляться как обозначение атомов в языке Лисп [4; 5; 6]. Эти объекты имели свойства или атрибуты. Другим примером может являться графическая система Sketchpad, где появляется понятие «класс» (как образец).

Первым объектно ориентированным языком был язык Симула [7]. Этот язык создал важнейшие концепции ООП: классы, объекты, наследование, динамическое связывание [8].

Дальнейшее развитие идей Симула отразилось в разработке языка Smalltalk, который повлиял на все последующие объектно ориентированные языки программирования [9]. Smalltalk изначально включал в себя окружение для программирования, динамическую типизацию и был реализован в виде интерпретатора [10]. В отличие от Симула это была полностью динамическая система, где классы могли создаваться и меняться динамически [11].

В 80-х годах стали появляться гибридные языки программирования, где объектно ориентированная парадигма соединялась с другими парадигмами, такой как функциональное программирование. В языках Flavors, Interlisp и Common Lisp [12; 13; 14] появляется множественное наследование и метаобъектный протокол.

Язык Objective-C [15] возник как расширение языка C посредством препроцессора, чтобы добавить объектно ориентированные возможности, как в языке Smalltalk. Объекты-экземпляры взаимодействуют друг с другом с помощью сообщений. Обработчик сообщения выбирается во время выполнения программы, следствием чего является отсутствие проверки типов. Ответ на сообщение не гарантируется. Данный подход имеет как достоинства, так и недостатки. К достоинствам Smalltalk относятся: динамичность отправки сообщений, возможность отправки одного сообщения множеству объектов, методы не обязательно должны быть реализованы, что дает возможность раннего тестирования без ошибок времени выполнения. К недостаткам данного подхода можно отнести меньшую скорость работы по сравнению с статическими системами.

Язык C++ [16; 17] также расширил язык C, добавив в него ООП, но за основу была взята модель языка Simula. Вместо передачи сообщений происходит вызов метода, который определяется статически во время компиляции, кроме случаев вызовов виртуальных методов. Это повышает скорость работы программы, но лишает ее динамичных свойств и требует обязательной реализации абстрактных методов.

Современные языки, такие как Python [18] и Java [19], поддерживают ООП. Язык Python соединяет процедурное программирование с объектно ориентированным, а язык Java является чистым объектно ориентированным языком, программировать, используя другие парадигмы, там нельзя.

Материалы и методы

В качестве базовой модели ООП возьмем модель языка Smalltalk с передачей сообщений, поздним связыванием и одиночным наследованием. Опишем концептуальную модель данной системы. Она должна обладать следующими функциями:

– динамическое создание нового класса;

– динамическое удаление класса;

– создание экземпляра класса;

– удаление объекта-экземпляра;

– получение свойства объекта;

– установка свойства объекта;

– добавление обработчика сообщения (метода) в класс;

– передача сообщения объекту;

– передача сообщения родительскому классу.

Каждый класс должен иметь уникальное имя, с помощью которого происходит работа с данным классом. Сам класс по своей сути является структурой или записью, которая содержит поля (свойства) и методы. В функциональном программировании удобно отделить методы от классов и сделать их обычными функциями. Тогда при создании класса нужно указать список его полей, которые являются переменными (символами) для доступа к свойствам экземпляра класса. Также при создании класса необходимо указать имя родительского класса, чтобы была возможность наследования (рис. 1).

(defclass point () (x y)) *;* возвращает имя класса

Рис. 1. Создание класса для точки с параметрами x и y

Fig. 1. Creation of point class with parameters x and y

При наследовании экземпляр дочернего класса будет иметь свойства родительского класса и свои собственные свойства, поэтому при создании дочернего класса достаточно указать только поля, принадлежащие дочернему классу. Например, класс «Линия» наследуется от класса «Точка», добавляя еще одну пару координат (рис. 2).

Для удаления класса необходим другой оператор (рис. 3).

(defclass line point (x2 y2))

Рис. 2. Создание класса «Линия», наследуемого от класса «Точка»

Fig. 2. Creating a "Line" class inherited from the "Point" class

(remove−class line) *;* возвращает *nil*

Рис. 3. Удаление класса

Fig. 3. Class removal

Для создания экземпляра класса достаточно знать только имя класса. При этом создается объект, у которого свойства будут иметь пустое значение, например nil (рис. 4).

```
(make−instance point) ; возвращает объект − экземпляр
```

Рис. 4. Создание экземпляра класса

Fig. 4. Creation of class instance

Так как пустые свойства могут быть неудобными, необходим способ задать начальные значения полей. Для экземпляров дочерних классов необходимо задавать значение всех полей: как родителя, так и потомка. Такую функцию обычно называют конструктор объекта-экземпляра (рис. 5).

```
(make−point 10 10) ; создание точки с координатами 10, 10
(make−line 10 10 20 30) ; создание линии с заданными координатами
```

Рис. 5. Конструкторы экземпляра класса

Fig. 5. Constructors of class instance

Так как в функциональных языках происходит автоматическое управление памятью, то для удаления экземпляра класса не требуется специальной функции. Сборщик мусора сам выявит объекты, на которых нет ссылок от других объектов, и тогда память, занимаемая объектом, будет автоматически освобождена.

Для получения значения свойства объекта нужен сам объект и имя-символ свойства (рис. 6).

```
(setq p2 (make−point 10 20)) ; в переменной p2 − новый экземпляр
(slot p2 'x) ; возвращает 10
```

Рис. 6. Получение значения свойства объекта

Fig. 6. Getting of object property value

Чтобы присвоить значение свойства объекту, необходим оператор присваивания. Удобно использовать макрос, который сам определяет вид присваивания – простое присваивание значения переменной или присваивание значения свойству объекта (рис. 7).

```
(setq p2 (make−point 10 20)) ; в переменной p2 − новый экземпляр
(setf (slot p2 'x) 50) ; присваиваем новое значение полю x
(slot p2 'x) ; возвращает 50
```

Рис. 7. Присваивание значения свойству объекта

Fig. 7. Assigning of new value to object propery

Как уже было сказано выше, методы класса удобно сделать как обычные функции, чтобы не создавать новый интерфейс передачи сообщений, а использовать простой вызов функции. Когда объект принимает сообщение, то он должен выбрать метод, который будет вызван при обработке сообщения. Выбор метода будет происходить по имени метода. Если метод с таким именем есть в классе объекта, то он будет вызван, если нет – поиск продолжится в классе родителя и т. д. Это называется полиморфизмом – разные обработчики одного и того же сообщения в разных классах. Если родительского класса нет, а метод не найден, то должна выводиться ошибка (рис. 8).

```
(setq p1 20 20) ; создали точку
(setq l1 (make−line 1 1 3 3)) ; создали линию
(move p1 20 20) ; перемещение точки
(move l1 20 20) ; перемещение линии
(test l1 10)    ; возвращает ошибку − неизвестный метод test
```

Рис. 8. Передача сообщений объектам

Fig. 8. Sending messages to objects

Для создания метода необходим оператор, которому передается имя метода, имя класса, параметры метода и тело метода. Также необходима переменная, которая получает ссылку на экземпляр класса, обычно self (рис. 9).

```
(defmethod move ((self point) dx dy) ; создаем метод для класса точки
(setf (slot self 'x) (+ (slot self 'x) dx)) ; dx, dy − вектор перемещения
(setf (slot self 'y) (+ (slot self 'y) dy)))
(defmethod move ((self line) dx dy) ; для объекта класса линии
(setf (slot self 'x) (+ (slot self 'x) dx)) ; перемещаются обе точки
(setf (slot self 'x2) (+ (slot self 'x2) dx))
(setf (slot self 'y) (+ (slot self 'y) dy))
(setf (slot self 'y2) (+ (slot self 'y2) dy)))
```

Рис. 9. Создание метода с одним именем для разных классов

Fig. 9. Creation of methods with the same name to different classes

Для удаления метода достаточно указать имя метода и имя класса (рис. 10).

Кроме возможности передачи сообщения объекта необходима также возможность перенаправления сообщения родительскому классу или суперклассу (рис. 11).

```
(remove−method move point) ; у класса точки удален метод move
```

Рис. 10. Удаление метода

Fig. 10. Method removal

```
(setq l1 (make−line 1 1 3 3)) ; создали линию
(super move l1 20 20) ; будет перемещена только первая точка линии
```

Рис. 11. Перенаправление сообщения родительскому классу

Fig. 11. Message redirection to parent class

Приведенный выше интерфейс обеспечивает все базовые возможности для объектно ориентированного программирования, такие как динамическая работа с объектами, классами, наследование, полиморфизм.

Результаты и их обсуждение

Для реализации модели объектно ориентированной системы воспользуемся интерпретатором функционального языка с возможностями метапрограммирования [20].

Заметим, что в классах и объектах присутствует общая структура. И там и там есть поля или свойства и их значения, в т. ч. функции-методы. Подобная структура называется «словарь» или «хеш-таблица» (ассоциативный массив), потому что чаще всего она реализуется с использованием хеш-функции для быстрого поиска. В таком объекте каждому ключу-символу соответствует значение как другой объект, поэтому данная структура универсальна и может хранить любые иерархические данные.

Хеш-таблица может поддерживаться языком напрямую, но ее также можно легко реализовать как А-список (рис. 12).

Рассмотрим интерфейс для работы с хеш-таблицей (рис. 13).

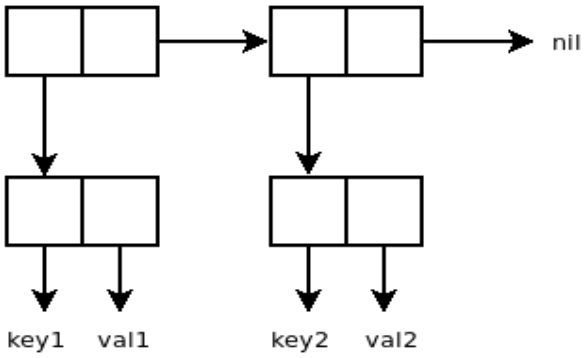

Рис. 12. Устройство хеш-таблицы

Fig. 12. Hash table structure

```
(setq a (make−hash)) ; в переменной a новая хеш − таблица
(set−hash a 'x 5)    ; добавление значения по ключу
(set−hash a 'y 10)   ; добавление значения по другому ключу
(set−hash a 'x 15)   ; установка нового значения по ключу
(get−hash a 'x)      ; получение значения по ключу
(delete−hash a 'x)   ; удаление значения с данным ключом
```

Рис. 13. Интерфейс хеш-таблицы

Fig. 13. Hash table interface

Теперь хеш-таблица будет основной структурой для классов и объектов. В качестве значения по ключу в хеш-таблице может храниться другая хеш-таблица. Это позволяет создавать иерархические данные, что необходимо для реализации классов.

Объекты-классы будут храниться в глобальной хеш-таблице *class-table*. Ключами в этой таблице будут символы – имена классов. Значениями являются сами объекты-классы в виде другой хеш-таблицы. Она содержит следующие поля:

– parent – имя-символ класса родителя;

– slots – список свойств класса;

– другие поля с ключом *имя метода* и таким значением, как lambda-функция метода.

Рассмотрим пример таблицы классов (рис. 14).

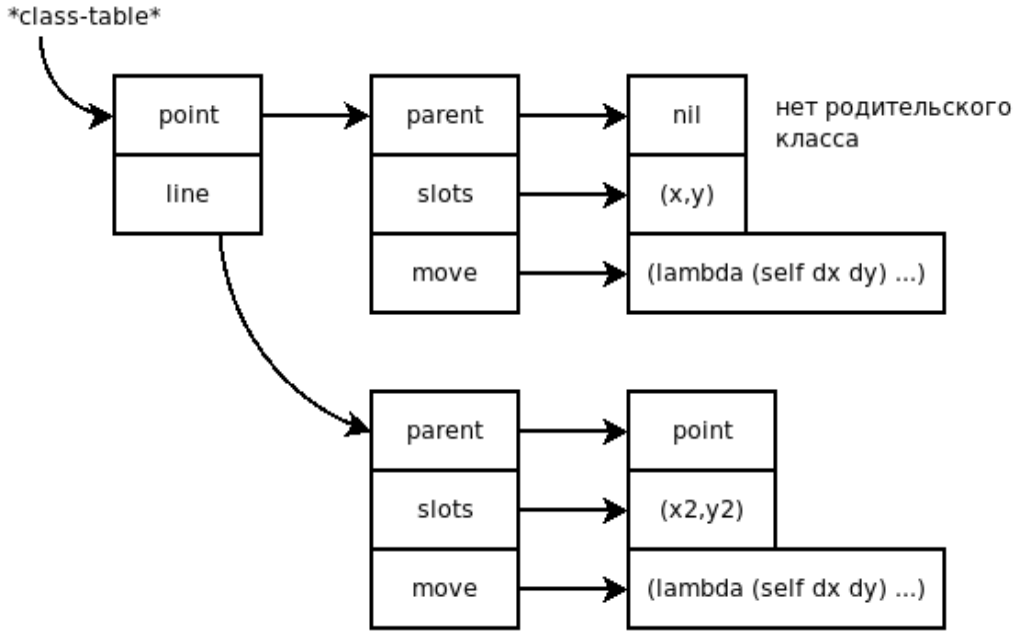

Рис. 14. Таблица классов

Fig. 14. Class table

При создании класса создается объект таблицы классов, заполняются его поля, и он добавляется в таблицу. Также здесь удобно автоматически создать конструктор экземпляра с инициализацией полей класса. Имя конструктора генерируется как “make-” и имя класса. Тогда макрос создания класса будет иметь следующий вид (рис. 15).

В конструкторе создается экземпляр объекта, а далее используется генерация кода инициализации полей с помощью функции отображения списка полей. Вспомогательная функция get-slots получает список свойств класса, рекурсивно добавляя к нему список свойств родительского класса (рис. 16).

```
(defmacro defclass (name parent slots)
"Создание␣нового␣класса  ␣␣name␣−␣имя,␣parent␣−␣родительский␣
класс,
    ␣␣slots␣−␣список␣полей"
     "(defclass␣point␣()␣(x␣y))"
 ‘(let ((class (make−hash)))
    (set−hash class ’parent ’,parent)
    (set−hash class ’slots ’,slots)
    (set−hash ∗class−table∗ ’,name class)
    ’,name)
 ‘(defun ,(intern (concat "MAKE−" (symbol−name name))) ,(get−slots name)
   (let ((obj (make−instance ,name)))
   ,@(map ’(lambda(s) ‘(setf (slot obj ’,s) ,s)) (get−slots name))
   obj))
 ‘’,name)
```

Рис. 15. Макрос создания класса

Fig. 15. Class creation macro

```
(defun get−slots (class)
 "Получение␣списка␣свойств␣класса␣class"
 (if (null class)  nil
  (let∗ ((cl (get−hash ∗class−table∗ class))
    (slots (get−hash cl ’slots))
    (parent (get−hash cl ’parent)))
   (append (get−slots parent) slots))))
```

Рис. 16. Функция получения свойств класса

Fig. 16. Class properties getting function

Удаление класса не представляет сложностей, так как совпадает с функцией удаления значения из хеш-таблицы классов по ключу – имени класса.

Объект-экземпляр класса представляет собой хеш-объект со всеми свойствами (своими и всех родителей), также здесь необходим ключ class, который хранит имя класса данного объекта (рис. 17).

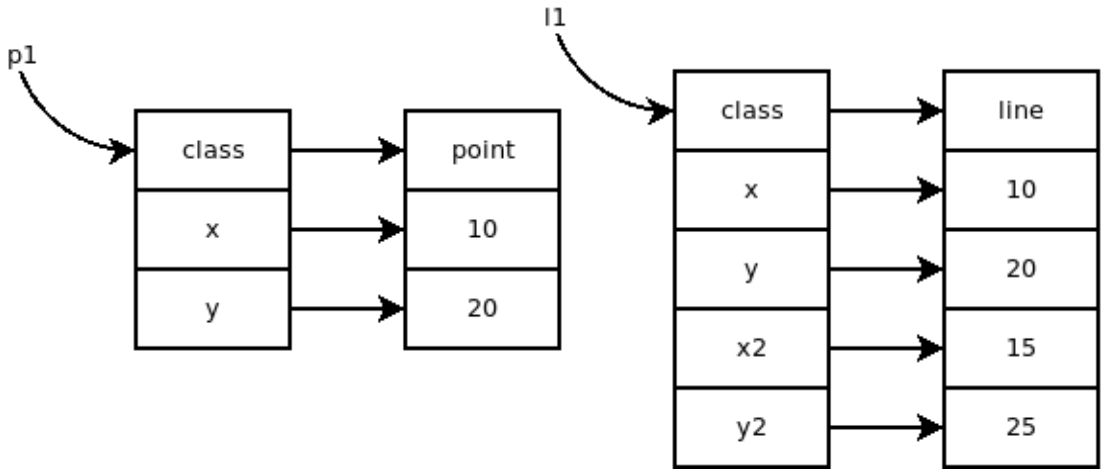

Рис. 17. Устройство объектов

Fig. 17. Objects structure

Создание экземпляра объекта представляет собой создание хеш-объекта, у которого устанавливается имя класса и все поля присваиваются в начальное значение nil (рис. 18). Также происходит проверка, что требуемый класс присутствует в таблице классов.

Получение свойства объекта реализуется просто как получение значения из хеш-объекта по ключу (рис. 19).

```
(defmacro make−instance (class)
 "Создать␣экземпляр␣объекта␣класса␣class"
 "(make−instance␣'point)␣−>␣((X.nil)(Y.nil))"
 (if (not (check−key *class−table* class)) (concat "no␣class␣" (symbol−name class))
  `(let ((object (make−hash)))
   (set−hash object 'class ',class)
   (app '(lambda (x) (set−hash object x nil)) ',(get−slots class))
   object)))
```

Рис. 18. Макрос создания экземпляра класса

Fig. 18. Class instance creation macro

```
(defmacro slot (obj key)
 "Возвращает␣значение␣свойства␣key␣y␣объекта␣obj"
 `(get−hash ,obj ,key))
```

Рис. 19. Макрос получения значения поля

Fig. 19. Property value getting macro

Установка нового значения в поле объекта происходит аналогично, используя функцию set-hash хеш-объекта.

Для реализации полиморфизма необходимо организовать диспетчеризацию – поиск метода. Сначала проверяется, есть ли метод с данным именем в классе объекта, а затем функция рекурсивно вызывается с классом родителя объекта (рис. 20).

```
(defun get−method (class−name method−name)
 "Рекурсивно ␣ возвращает ␣ тело ␣ метода ␣ method−name ␣ из ␣ класса ␣ class−name"
 (if (null class−name) (concat "no ␣ method ␣ " (symbol−name method−name))
  (let ((class (slot ∗class−table∗ class−name)))
   (if (check−key class method−name)
   (slot class method−name)
  (get−method (slot class 'parent) method−name)))))
```

Рис. 20. Функция поиска метода по имени и классу

Fig. 20. Method search function based on name and class

Когда определена вспомогательная функция поиска метода, возможно реализовать макрос создания нового метода. Этот макрос создает lambda-функцию в объекте класса, а также создает функцию с именем метода, которая осуществляет динамическую диспетчеризацию – поиск метода (рис. 21).

```
(defmacro defmethod (name args &rest body)
 "Определяет ␣ метод ␣ с ␣ именем ␣ name"
 "args ␣ − ␣ аргументы, ␣ первый ␣ аргумент ␣ состоит ␣ из ␣ имени ␣ экземпляра ␣
объекта ␣ и ␣ имени ␣ класса"
 "body ␣ − ␣ тело ␣ метода"
 `(let∗ ((class−name (cadar args))
    (class (slot ∗class−table∗ class−name)))
   (set−hash class ',name '(lambda ,(cons (caar args) (cdr args)) ,@body))
   (defun ,name ,(cons (caar args) (cdr args))
    (funcall (get−method (slot ,(caar args) 'class) ',name) ,(caar args) ,@(cdr args)))))
```

Рис. 21. Макрос создания метода и генерации функции диспетчеризации

Fig. 21. Method creation and generation of dispatch function macro

При вызове метода родителя возможен бесконечный цикл поиска метода. Эту проблему можно решить, если изменить класс объекта на класс родительского объекта, вызвать метод, а затем восстановить имя класса (рис. 22).

```
(defmacro super (method−name obj &rest args)
 "Вызов␣метода␣method−name␣родителя␣экземпляра␣класса␣obj␣с␣аргументами␣args"
 `(let ((origclass (slot ,obj 'class)) (res nil))
   (setf (slot ,obj 'class) (slot (slot *class−table* (slot ,obj 'class)) 'parent))
   (setq res (funcall (get−method (slot ,obj 'class) ',method−name) ,obj ,@args))
   (setf (slot ,obj 'class) origclass)
   res))
```

Рис. 22. Макрос вызова метода родительского класса

Fig. 22. Parent class method call macro

В результате реализации разработана компактная объектно ориентированная система. Весь код этой системы вместе с функциями хеш-объектов занимает 97 строк. Здесь представлены только базовые возможности ООП, но можно добавить и другие функции, подобно тем, которые есть в других системах.

CLOS (Common Lisp Object System) – это одна из самых мощных объектных систем. Как и разработанная система, она динамическая, обладает динамической диспетчеризацией, но также позволяет множественное наследование, интроспекцию и содержит метаобъектный протокол. Разработанная система обладает гораздо меньшей функциональностью, но также меньше на несколько порядков.

MerooNet – еще одна простая реализация объектно ориентированной системы на языке Scheme. По своим возможностям она совпадает с разработанной системой, но также имеет дополнительные возможности, такие как автоматическая генерация функций для получения и установки полей объекта, а также предикатов проверки принадлежности объекта конкретному классу. Также она включает определенные оптимизации.

Выводы

В результате работы была предложена концептуальная модель объектно ориентированной системы, которая была реализована с помощью функционального языка с поддержкой метапрограммирования. Было показано, что с помощью метапрограммирования можно создать маленькую по размеру кода, но в то же время функциональную динамическую объектную систему. Таким образом, в интерпретаторе с поддержкой метапрограммирования не нужно включать поддержку ООП, а существующий язык может быть расширен до необходимой функциональности с помощью макросов.

Список литературы

1. Bloch J. Effective Java: Programming Language Guide. URL: https://theswissbay.ch/pdf/Gentoomen%20Library/Programming/Java/Addison%20Wesley%20-%20Effective%20Java%20Programming%20Language%20Guide.pdf (дата обращения: 11.06.2024).

2. Цыбулько Д. М. Сравнение подходов проблемно ориентированного и объектно ориентированного программирования // Математические методы в технике и технологиях – ММТТ. 2015. № 7(77). С. 278–280. EDN WFEKVH

3. Шарыкин Р. Е. Методология разработки программного обеспечения с использованием модели распределенных объектно ориентированных стохастических гибридных систем // Информатика. 2022. Т. 19, № 1. С. 88–95. https://doi.org/10.37661/1816-0301-2022-19-1-88-95. EDN DIHJAV

4. Abelson H., Sussman G. J., Sussman J. Structure and Interpretation of Computer Programs. Cambridge, Massatchusetts: The MIT Press, 2004. 268 p.

5. Domkin V. Programming Algorithms in Lisp. Berkeley: Apress, Ins., 2021. 377 p.

6. Вторников А. Lisp: маленький гигант // Системный администратор. 2016. № 6(163). С. 64–69. EDN VZGXVD

7. Ершов А. П., Покровский С. Б. Эволюция языков программирования // Проблемы информатики. 2017. № 2(35). С. 70–79. EDN ZOLFPJ

8. Романов С. С. Ключевые понятия и особенности объектно ориентированного программирования // Таврический научный обозреватель. 2016. № 12-2(17). С. 141–146. EDN YFXCCN

9. Паттерны объектно ориентированного проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. СПб.: Питер, 2024. 448 с.

10. Nystrom R. Crafting interpreters. Moscow: Nobel Press, 2024. 640 p.

11. How (and why) developers use the dynamic features of programming languages: the case of smalltalk / O. Callaú, R. Robbes, É. Tanter, D. Röthlisberger // Empirical Software Engineering. 2013. Vol. 18, N 6. P. 1156–1194. https://doi.org/10.1007/s10664-012-9203-2. EDN YOGDYK

12. Пол Г. ANSI Common LISP. СПб.: Символ-Плюс, 2020. 448 с.

13. Малов А. В., Родионов С. В. Реализация упрощенного алгоритма Байеса в среде функционального программирования COMMON LISP // Известия СПбГЭТУ «ЛЭТИ». 2015. № 2. С. 32–37.

14. Сайбель П. Практическое использование Common Lisp. М.: ДМК Пресс, 2017. 488 с.

15. Кочан С. Программирование на Objective-C. М.: ЭКОМ Паблишерз, 2014. 550 с.

16. Страуструп Б. Язык программирования C++. Краткий курс. М.: Вильямс, 2019. 320 с.

17. Кувардин М. В. Сравнение структур данных связной список и массив структур с использованием примеров на языке программирования C++ // Научные исследования XXI века. 2021. № 2(10). С. 69–72. EDN TVPPBJ

18. Стивен Ф. Лотт, Дасти Ф. Объектно ориентированный Python. 4-е изд. СПб.: Питер, 2024. 704 с.

19. Шилд Г. Java. Полное руководство. 12-е изд. М.: Диалектика-Вильямс, 2023. 1344 с.

20. Чаплыгин А. А. Моделирование интерпретатора функционального языка программирования с возможностями метапрограммирования // Известия Юго-Западного государственного университета. Серия: Управление, вычислительная техника, информатика. Медицинское приборостроение. 2024. № 14(2). С. 181–193. https://doi.org/10.21869/2223-1536-2024-14-2-181-193

References

1. Bloch J. Effective Java: Programming Language Guide. Available at: https://theswissbay.ch/pdf/Gentoomen%20Library/Programming/Java/Addison%20Wesley%20-%20Effective%20 Java%20Programming%20Language%20Guide.pdf (accessed 11.06.2024).

2. Tsybulko D.M. Comparison of problem-oriented and object-oriented programming approaches. *Matematicheskie metody v tekhnike i tekhnologiyakh – MMTT = Mathematical Methods in Engineering and Technology – MMTT*. 2015;(7):278–280. (In Russ.) EDN WFEKVH

3. Sharykin R.E. Methodology of software development using a model of distributed object-oriented stochastic hybrid systems. *Informatika = Informatics*. 2022;19(1):88–95. https://doi.org/10.37661/1816-0301-2022-19-1-88-95. (In Russ.) EDN DIHJAV

4. Abelson H., Sussman G.J., Sussman J. Structure and Interpretation of Computer Programs. Cambridge, Massatchusetts: The MIT Press; 2004. 268 p.

5. Domkin V. Programming Algorithms in Lisp. Berkeley: Apress, Ins.; 2021. 377 p.

6. Tuesdays A. Lisp: The Little Giant. *Sistemnyi administrator = System Administrator*. 2016;(6):64–69. (In Russ.) EDN VZGXVD

7. Ershov A.P., Pokrovsky S.B. Evolution of programming languages. *Problemy informatiki = Problems of Computer Science*. 2017;2(35):70–79. (In Russ.) EDN ZOLFPJ

8. Romanov S.S. Key concepts and features of object-oriented programming. *Tavricheskii nauchnyi obozrevatel' = The Tauride Scientific Observer*. 2016;(12):141–146. (In Russ.) EDN YFXCCN

9. Gamma E., Helm R., Johnson R., Vlissides J. Patterns of object oriented design. St. Petersburg: Piter; 2024. 448 p. (In Russ.)

10. Nystrom R. Crafting interpreters. Moscow: Nobel Press; 2024. 640 p.

11. Callaú O., Robbes R., Tanter É., Röthlisberger D. Chow (and why) developers use the dynamic features of programming languages: the case of smalltalk. *Empirical Software Engineering*. 2013;18(6):1156–1194. https://doi.org/10.1007/s10664-012-9203-2. EDN YOGDYK

12. Paul G. ANSI Common LISP. St. Petersburg: Simvol-Plyus; 2020. 448 p. (In Russ.)

13. Malov A.V., Rodionov S.V. Implementation of the simplified Bayes algorithm in the COMMON LISP functional programming environment. *Izvestiya SPbSETU "LETI" = Proceedings of Saint Petersburg Electrotechnical University*. 2015;(2):32–37. (In Russ.)

14. Saibel P. Practical use of Common Lisp. Moscow: DMK Press; 2017. 488 p. (In Russ.)

15. Kochan S. Programming in Objective-C. Moscow: ECOM Publishers; 2014. 550 p. (In Russ.)

16. Stroustrup B. C++ programming language. Short course. Moscow: Vil'yams; 2019. 320 p.

17. Kuvardin M.V. Comparison of data structures a linked list and an array of structures using examples in the C++ programming language. *Nauchnye issledovaniya XXI veka = Scientific Research of the XXI Century*. 2021;(2):69–72. (In Russ.) EDN TVPPBJ

18. Stephen F. Lott, Dusty F. Object Oriented Python. 4th ed. Moscow: Piter; 2024. 704 p. (In Russ.)

19. Shield G. Java. A complete guide. 12th ed. Moscow: Dialectics-Williams; 2023. 1344 p. (In Russ.)

20. Chaplygin A.A. Modeling of a functional programming language interpreter with metaprogramming capabilities. *Izvestiya Yugo-Zapadnogo gosudarstvennogo universiteta. Seriya: Upravlenie, vychislitel'naya tekhnika, informatika. Meditsinskoe priborostroenie = Proceedings of the Southwest State University. Series: Control, Computer Engineering, Information Science. Medical Instruments Engineering*. 2024;14(2):181–193. (In Russ.) https://doi.org/ 10.21869/2223-1536-2024-14-2-181-193

Информация об авторе / Information about the Author

Чаплыгин Александр Александрович,
кандидат технических наук, доцент кафедры программной инженерии, Юго-Западный государственный университет,
г. Курск, Российская Федерация,
e-mail: alex_chaplygin@mail.ru,
ORCID: 0009-0009-8739-2695

Aleksandr A. Chaplygin, Candidate of Sciences (Engeneering), Associate Professor of the Department of Software Engeneering, Southwest State University,
Kursk, Russian Federation,
e-mail: alex_chaplygin@mail.ru,
ORCID: 0009-0009-8739-2695