Программные системы и вычислительные методы
Правильная ссылка на статью:
Дагаев Д.В. Инструментальный подход к программированию в системе МультиОберон // Программные системы и вычислительные методы. 2024. № 1. DOI: 10.7256/2454-0714.2024.1.69437 EDN: WVZVVU URL: https://nbpublish.com/library_read_article.php?id=69437

Инструментальный подход к программированию в системе МультиОберон

Дагаев Дмитрий Викторович

ORCID: 0000-0003-0343-3912

Генеральный директор, ООО "СКАДИ"

115230, Россия, г. Москва, Зеленый проспект, 5/12, строение 3, пом. 1

dvdagaev@oberon.org

Статья из рубрики "Языки программирования"

DOI:

10.7256/2454-0714.2024.1.69437

EDN:

WVZVVU

Дата направления статьи в редакцию:

25-12-2023

Аннотация: Объектно-ориентированные подходы к программированию, имеют свою область применимости. Для ряда задач традиционно отдают предпочтение классическим методам структурного программирования. Эти предпочтения нередки для детерминированного мира и в системах, ориентированных на машинное представление. Исторически классические методы развивались от архитектуры фон Неймана представления машины. При решении проблем детерминированного мира выявляются преимущества подходов, противоположных объектно-ориентированному мышлению. Например, язык и системы на базе модульного языка программирования Оберон в классической реализации демонстрируют минималистский путь для достижения надежности, существенно отличающийся от большинства программных систем, стремящихся к максимизации числа поддерживаемых функций. Технология программирования, управляемого данными также отходит от традиционной объектной модели, требуя сепарации кода от данных. Предлагаемый автором статьи инструментальный подход объединяет Оберон-технологии с программированием, управляемым данными, при этом оставляя присущие для ООП механизмы

взаимодействия по интерфейсам. Вместо объекта предложен ассоциированный с объектом инструмент, не сохраняющий в себе данные. Предложенный в данной статье инструментальный подход отличается как от объектного представления, так и от классического структурного. Он позволяет сохранять преимущества обоих подходов. При этом инструментальный подход работает в инфраструктуре программирования управляемого данными. От объектно-ориентированного подхода берется полиморфизм и возможность работы по интерфейсам. От классического структурного программирования берется определение структур данных и взаимодействие с ними. От программирования, управляемого данными используется сепарация кода от данных и жизненный цикл последних в персистентном виде. Новизной является то, что инструментальный подход предлагает отличную от ООП ветвь развития для классического языка программирования Оберон и классического подхода. Реализованные в системе МультиОберон, инструментальный подход позволяет решать ряд важных задач, в частности, задачи автоматизации в критически важных системах.

Ключевые слова:

инструментальный подход, Оберон, ограничение, компилятор, модульность, программирование управляемое данными, МультиОберон, СКАДА, Информатика-21, Метаданные

Введение

Объектно-ориентированные подходы к программированию, равно как и другие методы, имеют свою область применимости. Для одних задач осмысливание видов «все является объектом», «думай как объект» является естественным способом построения архитектуры. Для других задач объектные построения представляют собой лишний уровень абстракций, не имеющий явного отношения к решаемой проблеме, но имеющий явно выраженные стоимостные показатели в виде избыточности компьютерных ресурсов.

Следует отметить, что даже сторонники ООП признают локальность предлагаемых ими решений, например в [1] мир разделяется на области, где объектная парадигма и парадигма классического структурного программирования разделяются по осям «знания-социальный мир» и «реализация-детерминированный мир». С этим утверждением согласны разработчики областей детерминированного мира, отдающие предпочтение классическим подходам на основе алгоритмов и структур данных [2]. При этом важным направлением являются достижения объектного мира в части абстрактного доступа через интерфейсы и организации взаимодействия.

Язык программирования и система Оберон [3] исторически были основаны на минималистском классическом подходе, при наличии полиморфизма в расширении типов данных. В дальнейшем Оберон-2 развивался в виде добавления методов, а семейство Оберон-07 развивалось в сторону исключения объектной, динамической функциональности для последующего применения в контроллерах и иных встроенных системах.

Автором предложен и используется подход ограничительной семантики, который вместо нескольких языков и компиляторов использует один язык и фронтенд компилятора с системой семантических ограничений. МультиОберон представляет собой программную реализацию компилятора языка Оберон с системой семантических ограничений [4].

Решения на МультиОбероне задач из детерминированного мира потребовали ограничить имеющуюся в языке семантику работы с объектами и использовать классические структуры данных. При этом возникают вопросы интерфейсов и абстракций для доступа к данным.

В данной статье предложен инструментальный подход, при котором вместо интерфейса объекта, содержащего данные неизвестной структуры, используется интерфейс инструмента, не содержащего в себе данных и связанного с экземпляром данных. Инструмент, в отличии от объекта, не аллокируется в динамической памяти, не требует сериализации/десериализации, не может потерять данных при аварийном завершении из-за исключительной ситуации.

Понятие инструмента прозрачно, если взять аналогию с созданием скульптуры через обработку на разных фазах разными инструментами от черновой до финишной отделки.

Инструментальный подход хорошо согласуется с дата-центричным программированием, когда программы отделены от данных и связи между каждым экземпляром структуры данных и программой могут мгновенно и однозначно устанавливаться.

Инструментальный подход особенно выгодно использовать в системах 24x7 детерминированного мира, где время жизни данных может существенно превышать время жизни программ, если в системе организованы персистентные данные в разделяемой памяти.

Объектное мышление против структурного программирования

Про преимущества ООП было сказано достаточно, гораздо реже говорилось об областях эффективного применения. И эти области эффективного применения не охватывают 100% возникающих задач. Кроме этого, при использовании объектных подходов меняется технологическая культура производства ПО, основанная на совершенно других абстракциях, нежели классические подходы. Абстрагирование от структур данных и сосредоточение на интерфейсах как неких правилах несет несомненно преимущества в части более систематического представления каждой предметной области. При этом объект становится некоторой универсальной сущностью, но при этом теряет свои уникальные особенности в части структур данных. Следовательно, решение об использовании или неиспользовании ООП имеет смысл применять на самых ранних стадиях создания программных систем.

Чем руководствоваться и какие при этом есть варианты? Д.Херш в [1] рассматривает оси «знания-социальный мир» и «реализация-детерминированный мир» (рисунок 1).

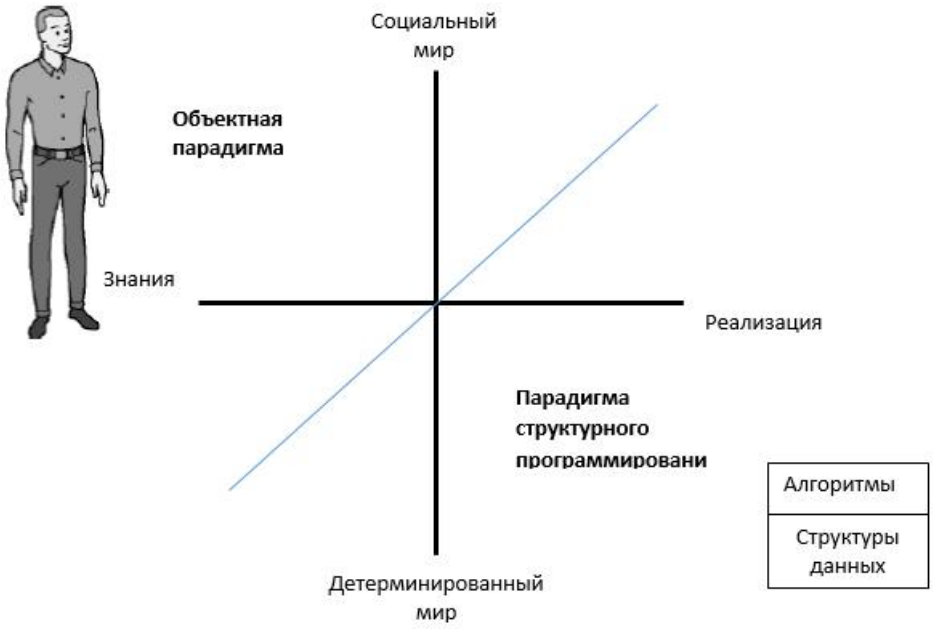

Рис. 1. Оси знания-социальный мир и реализация-детерминированный мир

Fig.1 Comprehension-Sociocultural world and Implementation-Computer Science

Разделение областей использования ООП для знаний и социального мира и классического структурного программирования для систем детерминированного мира или тех, в которых превалирует реализация, представляется логичным, особенно, учитывая, что это сформулировал теоретик ООП.

В данной статье мы будем рассматривать подходы к реализации систем детерминированного мира.

Программирование, управляемое данными

Для детерминированных программ характерна первичность данных и состояния, в таких задачах часто применяются технологии программирования, управляемого данными (data-oriented programming DOP), основные принципы которой изложены в [5].

Программы, управляемые данными, предназначены для построения слабосвязанных (loose-coupling) систем, использующих механизмы восстановления. Принципы программирования, управляемого данными диктуют следующие решения.

1. Предоставление данных и метаданных. Прямо противоположно инкапсуляции в ООП – данные о состоянии ПО предоставляются в рамках прав доступа программных компонентов. Метаданные несут информацию об имени, типе и далее о составе входящих в данный тип вложенных типов. Такое представление должно обеспечивать достаточную информацию для чтения и выполнения представленных данных формальной машиной абстрактного исполнителя.

2 . Скрытие подробностей кода. DOP скрывает программный код и основывается на сообщениях между компонентами. Интерфейс являет некритическим свойством системы. Все, что мы знаем, что имеются некие слабо связанные системы в распределенной конфигурации. В отличие от ООП зависимости между объектами и, как следствие, взаимное влияние отсутствует.

3. Сепарация кода от данных является также принципом DOP.

4 . Обработка данных должна осуществляться на основе метаданных реализующей платформы.

Важной чертой программирования, управляемого данными, является персистентность данных. Время жизни данных становится больше времени жизни программного кода. Даже при сбоях и восстановлении программного обеспечения не требуется восстановление данных.

Рисунок 2 иллюстрирует сбой и восстановление данных в ООП-системе и в DOP-системах вверху и внизу соответственно. Сбой приводит к исключению и завершению работы некоторого компонента программного обеспечения. В случае ООП при сбое возникает необходимость сериализации данных во временное хранение, что корректно реализуется достаточно редко. При восстановлении должна осуществляться десериализация данных. После восстановления должны восстанавливаться связи у объектов, адресующих данный. Все указанные механизмы являются достаточно проблемными в части реализации, особенно по событию в виде сбоя программы.

В случае DOP осуществляется аварийное завершение и рестарт программного модуля. Других действий не требуется, при условии сепарации кода от данных. Восстановление связей у объектов, адресующих данный, не требуется, т.к. все связи между объектами обеспечены метаданными реализующей платформы и находятся в персистентной памяти.

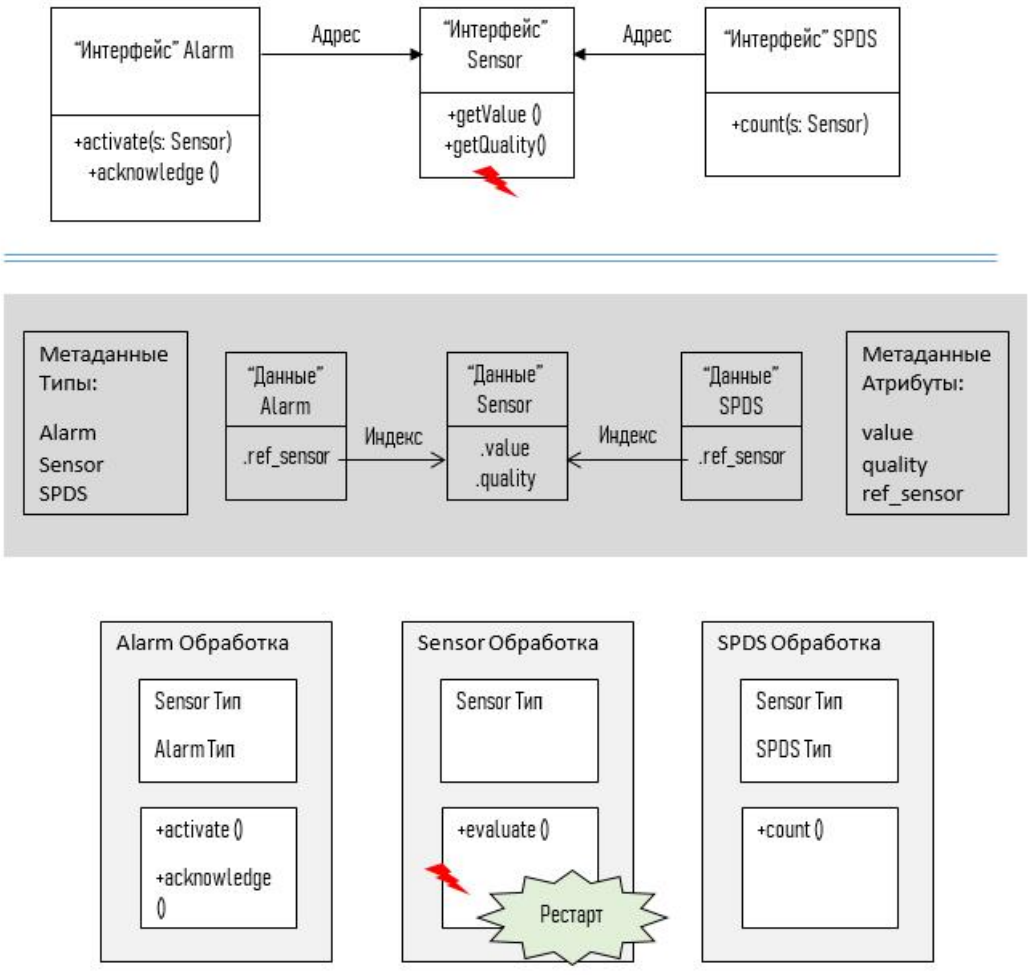

Рис. 2. Сохранение и восстановление данных ООП и DOP

Fig.2 OOP and DOP дата saving and retrieving

Сепарация кода от данных также означает сепарацию хранения, когда данные хранятся в персистентной памяти, а коды в памяти, в которую загружается программный модуль.

Семантические ограничения МультиОберона

Реализация загрузки и перезагрузки программного обеспечения может быть обеспечена либо на уровне рестарта процесса, либо средствами языка, поддерживающего модульное программирование. Оберон[3] является языком модульного программирования. Каждый программный модуль представляет собой единицу хранения, компиляции, загрузки и исполнения.

Построение систем с повышенными требованиями к функциональной безопасности приводят к запретам выделения и использования динамической памяти. В системе МультиОберон[4] указанные запреты означают, что в модуле непосредственно функция NEW быть вызвана не может. Следовательно аллокирование динамической памяти декларируется как отсутствующее. Разумеется, такие определения должны охватывать все рассматриваемые модули. Помимо рассмотрения ограничений импортируемого модуля, можно запретить использование системных преобразований, использующих данные из внешних функций.

RESTRICT –NEW, -SYSTEM;

При наличии ключевых слов NEW и SYSTEM проверка не будет пройдена и завершится синтаксической ошибкой.

Инструментальный подход вместо объектного

Отказ от динамического аллокирования приводит к изменениям в механизме реализации методов программ. Первый способ, очевидно, должен использовать подходы классического структурного программирования. Второй способ может использовать статические объекты без передачи и присваивания как указатели. Оба указанные способа теряют присущую ООП способность выстраивать системы и паттерны по интерфейсам[6], а реализации подставлять, как соответствие определенного интерфейса.

Автором предлагается инструментальный подход, заключающийся в создании вместо объекта инструмента, не содержащего в себе данные. Указатель на инструмент, подобно указателю на объект, может использоваться при подстановке и присваиваниях как данному, так и всем базовым типам. Указатель на инструмент адресует тип данных, находящейся в памяти модуля. В таблице 1 приведены отличия инструмента от объекта.

	Указатель на объект	Указатель на инструмент
Доступ по интерфейсу	Да	Да
Наличие данных	Есть	Нет
Ассоциирование с данными	Нет	Возможны
Расположение в памяти	Динамическая	Статическая
Число экземпляров	N – число объектов	Один

Таб. 1. Отличия инструмента от объекта

Table 1. Tool and object difference

Получение инструмента системной функцией TOOL

Системная функция TOOL с одним аргументом предназначена для запроса инструмента. Например, функция TOOL возвращает stdlog - указатель на инструмент типа StdLog.Hook. Инструмент представляет собой запись, не содержащую полей данных. Указатель на инструмент используется как интерфейс консоли вывода информации.

TOOL(stdlog);

stdlog.String("Tool for an object"); stdlog.Ln;

Тип инструмента должен быть определен в секции переменных.

VAR stdlog: StdLog.Hook;

Повторный и все последующие вызовы системной функции TOOL приведут в возврату того же адреса.

TOOL(stdlog2); ASSERT(stdlog=stdlog2);

Поскольку тип StdLog.Hook содержит методы и не содержит данных, то нет необходимости в выделении дополнительной памяти под каждый экземпляр инструмента. В этом заключается основное отличие системной функции TOOL от системной функции NEW.

Рис. 3. Получение адреса операторами TOOL и NEW

Fig.3 Retrieving and address by TOOL and NEW operators

Определения типов структур в языке Оберон содержатся в метаданных, имеющихся для каждого программного модуля (рисунок 3). В частности, в метаданных определен массив наследуемых типов, начиная с самого базового и заканчивая данным типом. Указатель на инструмент содержит адрес следующей за тэгом типа ячейки. Содержимое адресуемой ячейки не имеет значения, т.к. инструмент не содержит полей данных. Адресуемая ячейка располагается в области метаданных модуля. Каждый вызов TOOL возвращает адрес одной и той же ячейки.

В свою очередь, объект с данными в динамической памяти также должен содержать тэг типа и возвращать адрес следующей за тэгом типа ячейки. Па рисунке объект содержит динамические данные «Поле 1» и «Поле 2».

Использование стандартного механизма адресации с тэгом типа позволяет работать с

указателями на инструменты как с указателями на динамически аллокируемые объекты.

Однако есть важное различие между этими указателями. Указатели на динамические объекты обрабатываются сборщиком мусора после завершения использования. К указателям на инструменты сборщик мусора неприменим. Поэтому в определении указателя на инструменты должно присутствовать ключевое слово tool. Например, для нашего примера с консолью вывода информации.

Hook = POINTER TO RECORD [tool] (Log.Hook) END;

Использование системной функции TOOL допустимо только для указателей на инструменты.

Расширение базового типа инструмента

Тип данных StdLog.Hook включает в себя указатель на инструмент и используемые методы. В нашем примере он является реализацией и расширением базового типа Log.Hook. Базовый тип является абстрактным. Его определения задают интерфейсы, необходимые к реализации.

MODULE Log;

RESTRICT +TOOL;

Hook = POINTER TO ABSTRACT RECORD [tool] END;

PROCEDURE (log: Hook) String* (IN str: ARRAY OF CHAR), NEW, ABSTRACT;

PROCEDURE (log: Hook) Ln*, NEW, ABSTRACT;

Инсталляция инструмента типом StdLog обязывает реализовать в кодах все абстрактные функции, чтобы имелась возможность подстановки вместо абстрактного интерфейса log реализацию log := stdlog. Тогда каждый вызов log.String() будет приводить к вызову stdlog.String(), а архитектуру программного обеспечения можно будет выстраивать по паттернам и интерфейсам.

При реализации другой версии, например, в модуле MyStdLog, указателю будет присвоено другое значение log := mystdlog, адрес которого будет отличаться от предыдущей реализации.

Поскольку использование инструментов является расширением языка, то включение механизмов с флагами [tool] требует снятия семантического ограничения на инструменты. Это делается оператором RESTRICT +TOOL.

Ассоциирование с данными

Технология программирования, управляемого данными, предполагает наличие данных, отделенных от программного кода. При этом возникает вопрос, каким инструментом обрабатывать каждый экземпляр данных. Должны быть использованы метаданные реализующей платформы. Что это означает для инструментального подхода?

Для структуры данных могут быть ассоциированы инструменты, для обработки данной структуры. Инструментов может быть несколько, на различных стадиях обработки используются разные инструменты.

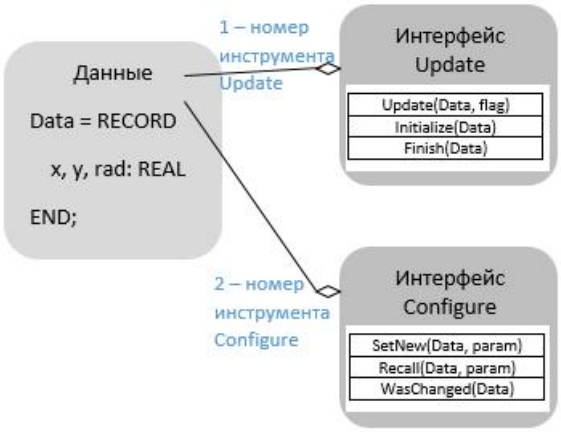

Рис. 4. Инструменты, ассоциированные с данными

Fig.4 Tools, associated with data

К метаданным реализующей платформы выдвигаются требования получения для каждого типа данных и номера инструмента указателя на инструмент. Если по данному номеру нет ассоциации, то, естественно, возвращается нулевой указатель.

Также следует добавить необходимость эффективного доступа к получению инструмента. Деревья и ассоциативные массивы для этих целей не подходят, требуется прямой индексный доступ на уровне информации в метаданных.

В МультиОбероне реализован эффективный индексный доступ к указателям на инструмент, рисунок 5.

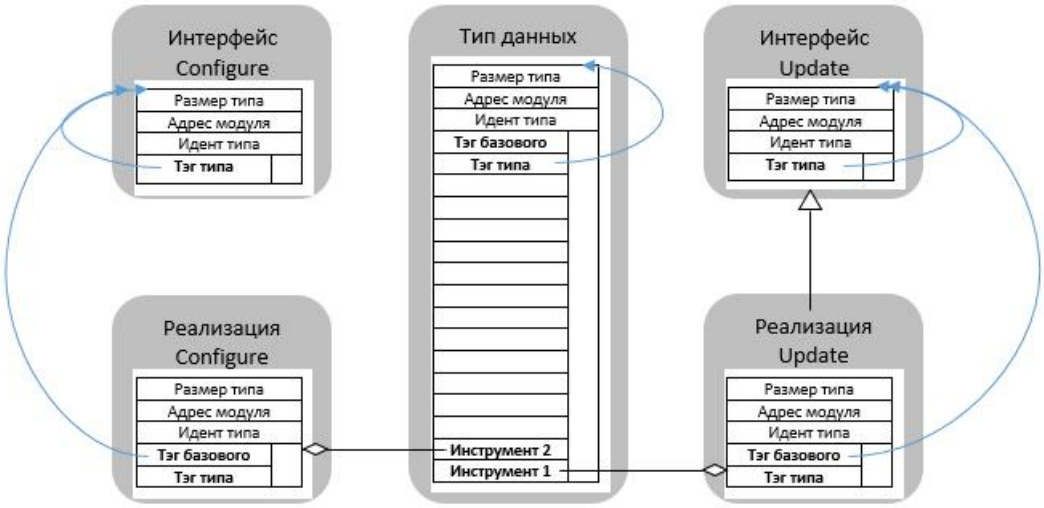

Рис. 5. Индексный доступ к инструментам

Fig.5 Indexed tool access

Для хранения указателей на тип инструмента используются свободные ячейки массива наследуемых типов base структуры Kernel.Type. Размер массива равен 16, что отражает максимальную глубину вложенности типов. При использовании инструментального подхода вложенные типы используются менее интенсивно. Для ассоциирования с первым инструментом используется ячейка 15, со вторым – 14 и т.д. Максимальное число ассоциированных инструментов определит компилятор.

Использование инструментов по данным

Для использования интерфейса Update по отношению к данным Data в ООП производится инсталляция объекта Data и вызов метода Update, который может быть виртуальным и подменяться реализациями для конкретных подтипов Data.

```
var Data data;

data.Update(flags);
```

Использование инструментального подхода предполагает наличие данных в персистентной области и запрос инструментов к ним.

```
VAR data: Data; tupdt: DataUpdate;

TOOL(tupdt, data, I_UPDT);

tupdt.Update(data, flags);
```

Системная функция TOOL с тремя аргументами предназначена для запроса инструмента, ассоциированного с данным типом по номеру интерфейса (в нашем случае, IFACE_UPDATE). Полученный инструмент применяется к данным вызовом метода Update с флагами.

Платой на разделение кода и данных является меньшая выразительность кода по сравнению с вариантом ООП.

Ассоциирование инструмента данным задается статически при определении типов.

```
CONST I_UPDT = 1;

Data = RECORD

x, y, rad: INTEGER

END;

DataUpdate = POINTER TO RECORD [tool] (IUpdate, Data, I_UPDT)

END;
```

Константа I_UPDT определяет номер инструмента. В определении инструмента закладывается, что инструмент ассоциирован с типом данных Data под номером I_UPDT.

Естественно, нет необходимости запроса TOOL(tupdt, data, I_UPDT) перед каждым вызовом метода. Это можно сделать один раз и далее использовать инструмент с относящимися к нему данными.

Выявленные проблемы и характеристики эргодичности

Проблема сохранения и восстановления сохраненных данных при сбоях и завершениях модулей программной системы не имеет простого решения для объектных архитектур. В сложных объектных системах сохранение требует сериализации, а восстановление - десериализации, что не всегда возможно выполнить при частичной неработоспособности распределенной системы. В результате такого восстановления возникает необходимость приведения объектов в требуемое состояние, что не всегда происходит. Это характеризуется внешними проявлениями, когда система работает, но не во всем наборе свойств.

В работе [7] рассмотрен ряд причин, приводящих к деградации свойств компьютерной системы со временем. Важным критерием является эргодичность, т.е. стабильность этих свойств, что требует доказательств, в том числе и архитектурного характера.

Использование данных в фиксированных персистентных хранилищах делает жизненный цикл изменения состояний системы более предсказуемым. Данные в таких хранилищах сбрасываются во временные файлы и восстанавливаются из них.

Вторая проблема восстановления связана с тем, что при таком восстановлении нужен механизм рестарта и приведения данных в соответствие. При разделении кода и данных при отказах в программных модулях кода нет никакой необходимости выгружать структуры данных из памяти. Модуль кода может перегружаться при необходимости, а данные остаются в памяти. Инструментальный подход гарантирует отсутствие подлежащих инициализации данных, т.к. инструменты, в отличие от объектов, не имеют полей данных.

Третьей проблемой является излишняя гибкость архитектуры системы реализующей платформы. Гибкость означает, что структуры данных и поведенческие характеристики объектной системы будут неопределенными, что не позволит оценивать изменение состояния компьютерной системы. В случае инструментального подхода структуры данных становятся более определенными, и более предсказуемой получается архитектура системы. Определенность структур данных означает, что состояние системы определяется этими структурами, и только ими. И, следовательно, стабильность этих структур данных приведет к улучшению характеристик эргодичности системы в целом.

Четвертой проблемой является возможность применимости решений для компактного встроенного ПО без использования указателей на объекты, размещенные в динамически выделяемой памяти. Указатель на инструмент является по определению единственным и не требует размещения экземпляра объекта в памяти.

Инструментальный подход и архитектурные решения, связанные с разделением кода и данных, приводят к более стабильным по характеристикам и доказуемым системам. Характеристики эргодичности предлагаемого подхода становятся лучше за счет явного состояния, хранимого в структурах данных. В конечном итоге динамическое поведение системы определяется состоянием и последовательностью входных воздействий, и, чем стабильнее начальное состояние, тем более предсказуемым окажется конечное состояние.

Другие способы решения и подходы

Объектные способы представления охватывают как представление данных, так и метаданных [8]. Более того, объектные модели, основанные на паттернах проектирования в настоящее время наиболее широко применяются для задач оси «знания-социальный мир» [9-11]. Наиболее важным отличием объектных подходов является то, что как данные, так и метаданные представляют собой объекты, вызываемые удаленно через адрес. При загрузках/восстановлениях кодов программ эти адреса, равно как и ссылки на них, изменяются и подлежат инициализации.

Время жизни модулей кода объектных систем будет равным времени жизни самой системы. Для решений, связанных с разделением кода и данных, время жизни системы будет равно времени жизни модулей данных и превышать время жизни модулей кода.

Важным механизмом повышения времени жизни систем является принцип

«ортогональной персистентности» [12], определяющий механизмы независимого сохранения состояния данных. Применение указанного механизма для Java-систем описывалось в [13]. Реальным большим шагом стала разработка ОС Phantom (Digital Zone, Россия), построенной на принципах ортогональной персистентности. Общим инструментального подхода и подхода на основе ортогональной персистентности является безударные механизмы сохранения и восстановления состояния. Но есть важные различия. Инструментальный подход позволяет реализовывать системы с восстановлением как на основе стандартных ОС, так и для критически важных систем, где ОС не рекомендуются или прямо запрещены [14]. Инструментальный подход не требует использования сборщика мусора, который «является серьезной проблемой для операционных систем» [15]. Микромир и системы реального времени накладывают еще большие ограничения на использования таких механизмов, как сборка мусора. Применение инструментального подхода более соответствует оси «реализация-детерминированный мир» и является предпочтительным механизмом для указанной области.

Применение в ПО СКАДА-платформы

Инструментальный подход использовался при разработке ПО СКАДИ (информация по https://scadi.tech), расшифровываемой как средства комплексной автоматизации и диагностики. Распределенная архитектура СКАДИ состоит из узлов, связанных между собой сетевыми протоколами. Каждый узел представляет собой адресуемый по сети компьютер с установленным ПО СКАДИ.

Обработчики осуществляют доступ к данным общей памяти, включая таблицы и очереди сообщений. Средства синхронизации ПО СКАДИ обеспечивают идентичность данных на любом узле с проверками и выдачей диагностики, если синхронизация на узлах не обеспечена.

Общая схема обновления данных по очередям сообщений представлена на рисунке 6.

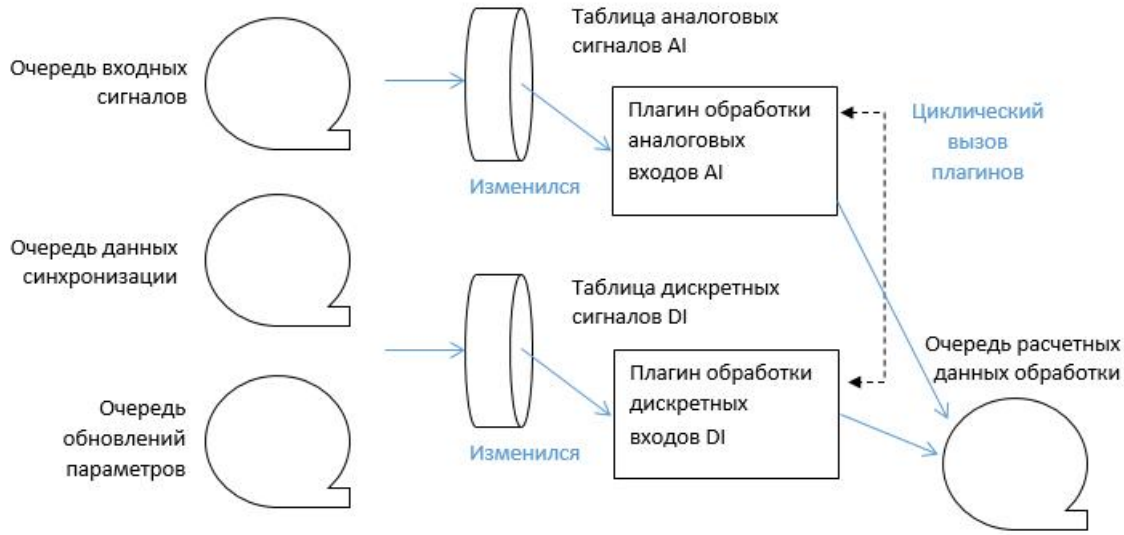

Рис. 6. Обработка данных СКАДИ

Fig.6 SCADI data processing

Программные сервисы DsDVP осуществляют обновление табличных данных узлов по

входным значениям очередей сообщений. Программа DsDVP каждому типу данных в общей памяти ассоциирует плагин обработки. Плагины обработки содержат исполняемые коды инструментов, соответствующих указанным типам данных. При переключении состояний задействованы автоматные механизмы [16], приведенные к виду инструмент – структура данных.

Например, плагин обработки для аналоговых данных AI содержит инструменты OvhAI, а плагин обработки дискретных данных DI – OvhDI. При этом объект обработки аналоговых данных AI содержит данные без методов, в то время как инструмент OvhAI обработки аналоговых данных содержит методы без данных.

Заключение

Предложенный в данной статье инструментальный подход позволяет сохранять преимущества классического структурного и объектно-ориентированного программирования. При этом инструментальный подход работает в инфраструктуре программирования управляемого данными. От объектно-ориентированного подхода берется полиморфизм и возможность работы по интерфейсам. От классического структурного программирования берется определение структур данных и взаимодействие с ними. От программирования, управляемого данными используется сепарация кода от данных и жизненный цикл последних в персистентном виде.

Сохранение и восстановление данных позволяет строить на базе инструментального подхода программные системы, у которых данные располагаются в специализированных, отдельно структурированных областях. Сохранение и восстановление таких данных не требует операций сериализации/десериализации и может осуществляться атомарно. Это особенно важно при построении высоконадежных систем 24x7, устойчивых к отказам в коде программных модулей. В свою очередь, программные коды сосредотачиваются в инструментах, которые не имеют состояния и, следовательно, не должны это состояние сохранять и восстанавливать.

Инструменты представляют собой программные объекты без данных, которые находятся в единственном экземпляре для каждого типа. Это позволяет отказаться от использования динамически выделяемой памяти на объекты, т.к. отсутствие данных не требует отсутствие экземпляра в памяти. Вместе с тем инструменты могут вызываться и обновлять ассоциированные с ними объекты – структуры данных.

Недостатком инструментального подхода является меньшая выразительность кода по сравнению с кодом ООП. В семантике вызова инструмента должны присутствовать как сам инструмент, так и ассоциированный экземпляр данных.

Инструментальный подход реализован в системе МультиОберон. Ограничительная семантика МультиОберона позволяет отключать функциональность ООП и включать функциональность инструментального подхода.

Подход «от ограничений» предполагает декларирование разработчиками того факта, что используется инструментальный подход, а значит все состояния данных сохраняются в персистентной памяти. Это декларации могут быть обусловлены стремлением повысить качество разрабатываемой системы или требованиями проектирования.

Использование инструментов и персистентных хранилищ данных существенно облегчает рассмотрение вопросов эргодичности. Такое разрабатываемое ПО будет меньше деградировать со временем.

Построение СКАДИ – системы с требованиями повышенной надежности реализовано с использованием инструментального подхода. Такое решение позволило позволило совершить еще один шаг по повышению качества программного продукта для построения ПО критически важных систем.

Библиография

1. David West. Object Thinking. Microsoft Press, 2004. С. 87-89.
2. Вирт Н., Алгоритмы и структуры данных. ДМК-Пресс, 2016. С. 272.
3. Н. Вирт, Ю. Гуткнехт. Разработка операционной системы и компилятора. Проект Оберон. ДМК-Пресс, 2017. С. 560.
4. Дагаев Д.В. Ограничительная семантика языка в системе МультиОберон // Программные системы и вычислительные методы. – 2023. – № 1. – С. 26-41. DOI: 10.7256/2454-0714.2023.1.36217 EDN: IWIODR
5. Rajive J., Ph.D. Data-Oriented Architecture: A Loosely-Coupled Real-Time SOA, Real-Time Innovations, Inc., 2007 August. С. 19-23.
6. Гамма Э., Хэлм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб: Питер, 2019. С. 368.
7. Дагаев Д.В. О разработке Оберон-системы с заданными свойствами эргодичности. Труды ИСП РАН, том 32, вып. 6, 2020 г., стр. 67-78. DOI: 10.15514/ISPRAS–2020–32(6)
8. Копылов М.С., Дерябин Н.Б., Денисов Е.Ю. Объектно-ориентированный подход к поддержке сценариев в системах оптического моделирования // Труды Института системного программирования РАН. 2023. No 35(2). стр. 169-180.
9. Nazar N., Aleti A., Zheng Y. Feature-based software design pattern detection // Journal of Systems and Software, 2022, vol. 185, pp. 1-12.
10. Yu D., Zhang P., Yang J., Chen Z., Liu C., Chen J. Efficiently detecting structural design pattern instances based on ordered sequences // Journal of Systems and Software, 2018, vol. 142, pp. 35–56.
11. Lo S. K., Lu Q., Zhu L., Paik H.-Y., Xu X., Wang C. Architectural patterns for the design of federated learning systems // Journal of Systems and Software, 2022, vol. 191, p. 357.
12. Hosking A., Nystrom N., Cutts Q., Brahnmath K. Optimizing the read and write barriers for orthogonal persistence // Advances in Persistent Object Systems, Morrison, Jordan, and Atkinson (Eds.). Morgan Kaufmann, 1999. p. 11.
13. Lefort A. A Support for Persistent Memory in Java // Computer science. Institut Polytechnique de Paris, 2023. English. p. 10.
14. ГОСТ Р МЭК 60880, Программное обеспечение компьютерных систем, выполняющих функции категории А // 2009. стр. 220.
15. Таненбаум А. Современные операционные системы // 4-е изд. – СПб.: Питер, 2015. Стр. 100-101.
16. Дагаев Д.В. Исполняющая машина автоматных программ // Научно-технический вестник информационных технологий, механики и оптики. 2021. Т. 21, № 4. С. 525-534.

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.

Со списком рецензентов издательства можно ознакомиться здесь.

Предмет исследования. Статья должна быть посвящена, исходя из названия, инструментальному подходу к программированию в системе МультиОберон. В целом, статья соответствует заявленной теме, однако отдельные моменты требуют доработки, о чём будет отмечено в соответствующих блоках рецензии.

Методология исследования автором выстроена на анализе и синтезе различных данных о предмете исследования. Ценно, что автор использует графический инструментарий представления полученных результатов. При этом, на представленных рисунках следует представить все надписи на русском языке (диссонирует основной текст статьи на русском языке и англоязычные рисунки). Также было бы хорошо увеличить разрешение подготовленных рисунков, т.к. внизу первого рисунка надписи на круге не видны.

Актуальность исследования вопросов, связанных с организацией программирования не вызывает сомнения, т.к. это напрямую связано с обеспечением технологического суверенитета Российской Федерации и вкладом в достижение цифровой трансформации (как одной из национальных целей развития Российской Федерации на период до 2030 года).

Научная новизна в представленном на рецензирование материале присутствует. В частности, связана с изложенными на рисунке 1 «осями знания-социальный мир и реализация-детерминированный мир». Более того, интерес представляют обозначенные семантические ограничения МультиОберона, что может быть использовано как в других научных исследованиях, так и в практической деятельности.

Стиль, структура, содержание. Стиль изложения научный. Структура статьи автором выстроена, способствует раскрытию темы, но её было бы интересно добавить блоком по решению выявленных проблем. Ознакомление с содержанием статьи не позволило чётко сформировать представление о системе выявленных автором проблем и путях их решения. При доработке статьи следует уделить внимание этому вопросу, т.к. это значительно расширит востребованность научной статьи.

Библиография. Автором сформирован достаточно скудный библиографический список, состоящий из 7 источников. В то же время ценно, что автор использовал как отечественные, так и зарубежные научные публикации. При проведении доработки статьи рекомендуется расширить перечень источников хотя бы до 15-20, прежде всего, за счёт изучения современной научной мысли (2022-2024 гг.), т.к. сейчас основной фокус внимания автор обращён к публикациям, вышедшим ранее 2020 года.

Апелляция к оппонентам. К сожалению, в тексте рецензируемой статьи автор не вступает в научные дискуссии с другими авторами, что снижает впечатление от ознакомления с данным материалом. При доработке рекомендуется устранить данное замечание, т.к. это позволит усилить глубину аргументации авторских суждений, а также покажет прирост научного знания по сравнению с тем, что уже имеется в научной литературе.

Выводы, интерес читательской аудитории. С учётом всего вышеизложенного заключаем о том, что статья может быть опубликована после проведения доработки по указанным в тексте рецензии замечаниям. Представляется, что статья на данную тему будет представлять большой интерес читательской аудитории журнала «Программные системы

и вычислительные методы».

Результаты процедуры повторного рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Рецензируемая статья посвящена разработке инструментального подхода к программированию на основе изучения преимуществ классического структурного и объектно-ориентированного программирования.
Методология исследования обобщении опыта классического структурного и объектно-ориентированного программирования, анализе научных публикаций и стандартов по изучаемой теме.
Актуальность работы обусловлена необходимостью интеграции преимуществ структурного и объектно-ориентированного программирования.
Научная новизна рецензируемого исследования по мнению рецензента, состоит в предложенном инструментальном подходе, при котором вместо интерфейса объекта, содержащего данные неизвестной структуры, используется интерфейс инструмента, не содержащего в себе данных и связанного с экземпляром данных, при этом инструмент, в отличии от объекта, не аллокируется в динамической памяти, не требует сериализации/десериализации, не может потерять данных при аварийном завершении из-за исключительной ситуации.
В статье структурно выделены следующие разделы: Введение, Объектное мышление против структурного программирования, Программирование, управляемое данными, Семантические ограничения МультиОберона, Инструментальный подход вместо объектного, Получение инструмента системной функцией TOOL, Расширение базового типа инструмента, Ассоциирование с данными, Использование инструментов по данным, Выявленные проблемы и характеристики эргодичности, Применение в ПО СКАДА-платформы, Заключение, Библиография.
В статье рассмотрены преимущества и области эффективного использования объектно-ориентированного программирования; применение программ, управляемых данными для построения слабосвязанных систем, использующих механизмы восстановления; освещены особенности языка модульного программирования Оберон; предложен инструментальный подход, заключающийся в создании вместо объекта инструмента, не содержащего в себе данные; отражены причины, приводящие к деградации свойств компьютерной системы со временем; применение средств комплексной автоматизации и диагностики. От объектно-ориентированного подхода автором берется полиморфизм и возможность работы по интерфейсам, от классического структурного программирования – определение структур данных и взаимодействие с ними; от программирования, управляемого данными – сепарация кода от данных.
Библиографический список включает 16 источников – публикации отечественных и зарубежных ученых по теме статьи, а также интернет-ресурсы и ГОСТы, на которые в тексте имеются адресные ссылки, подтверждающие наличие апелляции к оппонентам.
К недостаткам представленного на рецензирование материала можно отнести использование аббревиатур без расшифровок (например, ООП), а также оформление таблиц с отступлением от принятых правил – наименование таблицы 1 отражено после нее, а не перед таблицей как это принято ГОСТ.
Статья отражает результаты проведенного авторами исследования, соответствует направлению журнала «Программные системы и вычислительные методы», содержит элементы научной новизны и практической значимости, может вызвать интерес у

читателей, рекомендуется к опубликованию после внесения корректив в соответствии с высказанными замечаниями.