Make for rustlang based applications

Postulates

`First White Postulate`

Сжимающаяся система, сжимаемое содержание

`Second Black Postulate`

Расширяемая система, сжимаемое содержание

`Third Yellow Postulate`

Расширяемая система, расширяемое содержание

Sequance of Ierarchy

A First Red Order

Первый порядок и первый слой в иерархии отвечает за структуры общего назначения, как правило, одиночно или количественно генеративного характера. Они держатся особняком от вложенности, но в редких случаях допускается вложенное применение для отладки всей системы.

M Second Orange Order

Второй порядок и второй слой системы первого порядка представляют собой слой между низкоуровневым и высокоуровневым управлением данными. Он интерпретирует, контролирует и управляет процессами.

C Third Gray Order

Материально ориентированный третий порядок и слой субсистемы второго порядка представляет собой скрепляющую подсистему, работающую с материальными объектами из реального мира или симулирующую, отображающую их и/или реагирующую на них

B Fourth Green Order

Четвертый порядок и слой субсистемы третьего порядка работает с математическим и абстрактным представлением реальных объектов. Он является самым последним и низовым слоем субсистем.

System of Modules

A > **Entity** relate to `First White Postulate` :

Служит контейнером для различных типов данных (текстовых, визуальных, аудио и т.д.). Сущности создаются генератором, могут быть обработаны манипулятором, соединены коннектором и посчитаны счетчиком. Сама сущность не интерпретирует данные, а лишь хранит их. Этот модуль относится к First White Postulate - Сжимающаяся система, сжимаемое содержание

Примеры: entity(main universal entity), string_entity, char_entity, visual_entity, digital_entity etc.

A > **Generator** relate to `Second Black Postulate` :

Генератор создает неизменные базисные сущности различного содержания. Эти сущности могут служить не только для создания новых объектов, но и для построения новых соединений или счетчиков. Если генератор создает только сущности, то он относится к First White Postulate, однако в своем общем назначении он относится ко Second Black Postulate

Примеры: entity_generator, only_text_entity_generator, counter_generator, left_robotic_hand_manipulator_generator

M > **Manipulator** relate to Third Yellow Postulate :

Управляет логикой соединений (connectors). В наборе всегда присутствуют интерпретатор и поступающая извне сущность. Используется для контроля и слаженной работы всей системы. Для этого требуется явный параллелизм, например, в подаче электроэнергии, регуляции жидкости/масла, логике движений. Второй вариант (явная параллельность) имеет лучшее время отклика, первый (последовательность) - более доступное решение. Модуль Manipulator не исключает многоуровневости и вложенности других манипуляторов. Этот модуль относится к Third Yellow Postulate - Расширяемая система, расширяемое содержание

Примеры: left_robotic_hand_manipulator, right_robotic_leg_manipulator, water_feeding_manipulator, network_manipulator

M > **Interpreter** relate to First White Postulate

Извлекает данные из сущности (entity) и интерпретирует их. Он определяет, как обрабатывать сущность в зависимости от ее текущего состояния и типа данных. Интерпретатор не изменяет сущность, а лишь дает представление о том, что она собой представляет. Интерпретатор интерпретирует сущность только один раз, при этом он поднимает флаг сущности в случае ее определения в системе. Описание сущности дается в текстовом виде, при этом для универсальной сущности можно указать, что ее роль - быть, например, собакой, кошкой или клоуном. Это текстовое описание адаптируется к свойствам, типам сущности, ее физическому или математическому представлению. Этот модуль относится к First White Postulate - Сжимающаяся система, сжимаемое содержание

Примеры: intrepeter, physics_entity_interpreter, entity_intrepeter, data_intrepeter, form_entity_intrepeter

C> **Connector** relate to `Second Black Postulate`:

Служит точкой входа для доступа и взаимодействия с вычислительными ресурсами и расчетов с данными. Он материально ориентирован и группирует данные в материально ориентированную систему. Коннектор не управляет логикой системы, но обеспечивает связь с основными функциями, предоставляемыми счетчиками. Коннектор дает доступ к операциям над данными в специализированном пространстве, но не изменяет методы и подходы к расчетам в математическом и физическом планах. Он предоставляет доступ к модулям физики и математики во внутреннем порядке, инкапсулируя абстрактную (B) и материально ориентированную логику (C) друг от друга. Этот модуль относится к Second Black Postulate - Расширяемая система, сжимаемое содержание

Примеры: hydraulic_connector, pneumatic_connector, clock_connector и т.д.

C> A M B> **Render** relate to `Second Black Postulate` :

Отображение данных, может использовать math_counter для этого, работает из любого слоя. Искомое место в третьем сером порядке, однако для отладки системы может быть вызван произвольно из любого места, находится в третьем порядке потому что это единственное место которое близко к точке до и после изменений. Этот модуль относится к Second Black Postulate - Расширяемая система, сжимаемое содержание

Примеры: render, text_render, digit_render, graphics_render

C> **Reactor** relate to `Second Black Postulate` :

Создает новое действие или сущность на основе взаимодействия двух или больше сущностей. Находится в модуле connector по той же причине что и модуль render. Этот модуль относится к Second Black Postulate - Расширяемая система, сжимаемое содержание

Примеры: uran_238_reactor, h2_reactor, player_enemy_reactor, collision_reactor

B > **Counter** relate to `First White Postulate` :

Выполняет все вычисления, сортировку, преобразования и манипуляции с данными строго математически. Это позволяет избавить систему от лишних специализированных служб и сосредоточиться на высокопроизводительных операциях. Основное отличие от сервисного подхода - явное название, которое объясняет, что мы уходим от сервисной архитектуры, в которой применяем сервис для специализированных разного рода сущностей. Проще рассматривать сущности строго математически как определенный тип данных, без необходимости создавать отдельные сущности.

При добавлении новой счетчик-структуры, следует хорошо подумать, не делает ли она то, что может сделать math_counter или sort_counter. Название счетчика должно определять группу алгоритмов определенной классификации, а не узкое направление. Этот модуль относится к First White Postulate - Сжимающаяся система, сжимаемое содержание

Примеры: math_counter, sort_counter, physics_counter, chemistry_counter, encrypt_decrypt_counter

A M C B > **Refiner** relate to `Second Black Postulate` :

Перерабатывает данные из одного вида в другой, сжимает для экономии места, шифрует их и дешифрует, перегруппирует, разделяет на части, проводит любые операции с данными, которые являются ни строго математическим или физическим, имеет потенциал вложенности математических методов создавая пути нестандартной переработки или/и перегруппировки, способы переработки неизменные, описаны строго математически, комбинирование этих способов и методов произвольный процесс. Работает из любого слоя. Этот модуль относится к Second Black Postulate - Расширяемая система, сжимаемое содержание

Примеры: data_refiner, wood_refiner, steel_refiner

Architecture Example

```
use std::collections::HashMap;

struct Entity {
    data: Vec<u8>,
    width: usize,
    height: usize,
    description: String,
}

struct EntityGenerator {
    next_id: usize,
}

impl EntityGenerator {
    fn new() -> Self {
        EntityGenerator { next_id: 0 }
    }

    fn generate(&mut self, width: usize, height: usize) -> Entity {
        let data = vec![0; width * height];
        let entity = Entity {
            data,
            width,
            height,
            description: String::new(),
        };
        self.next_id += 1;
        entity
    }
}

struct EntityInterpreter {
    entities: HashMap<usize, Entity>,
}

impl EntityInterpreter {
    fn new() -> Self {
        EntityInterpreter { entities: HashMap::new() }
    }

    fn interpret(&mut self, entity: &mut Entity) {
        let id = self.entities.len() + 1;
        if entity.width == entity.height {
            entity.description = String::from("square");
        } else {
            entity.description = String::from("rectangle");
        }
        self.entities.insert(id, entity.clone());
        println!("Interpreted entity with ID: {}", id);
    }
}
```

```
struct Connector {
    entities: Vec<Entity>,
    math_counter: MathCounter,
}

impl Connector {
    fn new(math_counter: MathCounter) -> Self {
        Connector {
            entities: Vec::new(),
            math_counter,
        }
    }

    fn connect(&mut self, entity: Entity) {
        self.entities.push(entity);
        self.math_counter.calculate(10.0, 5.0, |a, b| a + b);
        println!("Result: {}", self.math_counter.result);
    }
}

struct Manipulator {
    interpreter: EntityInterpreter,
    connector: Connector,
}

impl Manipulator {
    fn new(interpreter: EntityInterpreter, connector: Connector) -> Self {
        Manipulator {
            interpreter,
            connector,
        }
    }

    fn manipulate(&mut self, entity: &mut Entity) {
        self.interpreter.interpret(entity);
        self.connector.connect(entity.clone());
    }
    fn resize(&mut self, entity: &mut Entity, new_width: usize, new_height: usize) {
        entity.data = vec![0; new_width * new_height];
        entity.width = new_width;
        entity.height = new_height;
        self.interpreter.interpret(entity);
        self.connector.connect(entity.clone());
    }
}

struct MathCounter {
    result: f64,
}
```

```
impl MathCounter {
    fn new() -> Self {
        MathCounter { result: 0.0 }
    }

    fn calculate(&mut self, a: f64, b: f64, op: fn(f64, f64) -> f64) {
        self.result = op(a, b);
    }
}

fn main() {
    let mut generator = EntityGenerator::new();
    let mut interpreter = EntityInterpreter::new();
    let mut math_counter = MathCounter::new();
    let mut connector = Connector::new(math_counter);

    let mut manipulator = Manipulator::new(interpreter, connector);

    let mut square_entity = generator.generate(10, 10);
    manipulator.manipulate(&mut square_entity);
    println!("Entity description: {}", square_entity.description);

    manipulator.resize(&mut square_entity, 10, 15);
    println!("Entity description: {}", square_entity.description);
}
```