Использование предметно-ориентированных языков (ПОЯ) для унификации доступа к множеству гетерогенных устройств в рамках концепции интернета вещей (IoT)

УЛИТИН БОРИС(BULITIN@HSE.RU)

БАБКИН Э.А.

САНКТ-ПЕТЕРБУРГ, 2017

Industry 4.0. Киберфизические системы [1]

План презентации

0. Введение и основные вопросы.
1. Контекст:
   - 1.1 Архитектура микросервисов и Объектная модель устройства (ОМУ);
   - 1.2 Структура и графовое представление ПОЯ;
   - 1.3 Определение и классификация модельных эволюций.
2. Реализация предлагаемого подхода:
   - 2.1 Схема внедрения;
   - 2.2 Используемые технологии.
3. Практический кейс.
4. Заключение и дальнейшее развитие исследования.

Основные определения

- ***Предметно-ориентированный язык (ПОЯ)*** – компьютерный язык, предназначенный для решения задач в конкретной предметной области (M. Fowler “Domain Specific Languages”) [2].

- ***Интернет вещей (Internet of Things, IoT)*** – объединение сенсоров, электронных устройств и их пользователей в единую сеть (чаще всего через Интернет) (K. Ashton “That ‘internet of things’ thing“) [3].

Сложности организации IoT

- Разнообразие устройств, их архитектур;
- Гетерогенность языков управления устройствами;
- Синхронизация доступа к устройствам;
- Динамический контекст существования системы:
  - Изменения в устройствах,
  - Изменения в требованиях пользователей к системе и интерфейсам взаимодействия с ней.

Отсутствие единых стандартов

Основные вопросы

1. Каким образом организовать работу с разными устройствами в IoT?
2. Как унифицировать процедуру взаимодействия устройств между собой и с пользователем?
3. Как согласовать изменения в потребностях и компетенциях пользователей с изменениями в интерфейсе взаимодействия с системой?

Основные вопросы. Ответы

1. Каким образом организовать работу с разными устройствами в IoT?

Ответ: Архитектура микросервисов

2. Как унифицировать процедуру взаимодействия устройств между собой и с пользователем?

Ответ: Использование ПОЯ внешнего уровня

Архитектура микросервисов[1][4]

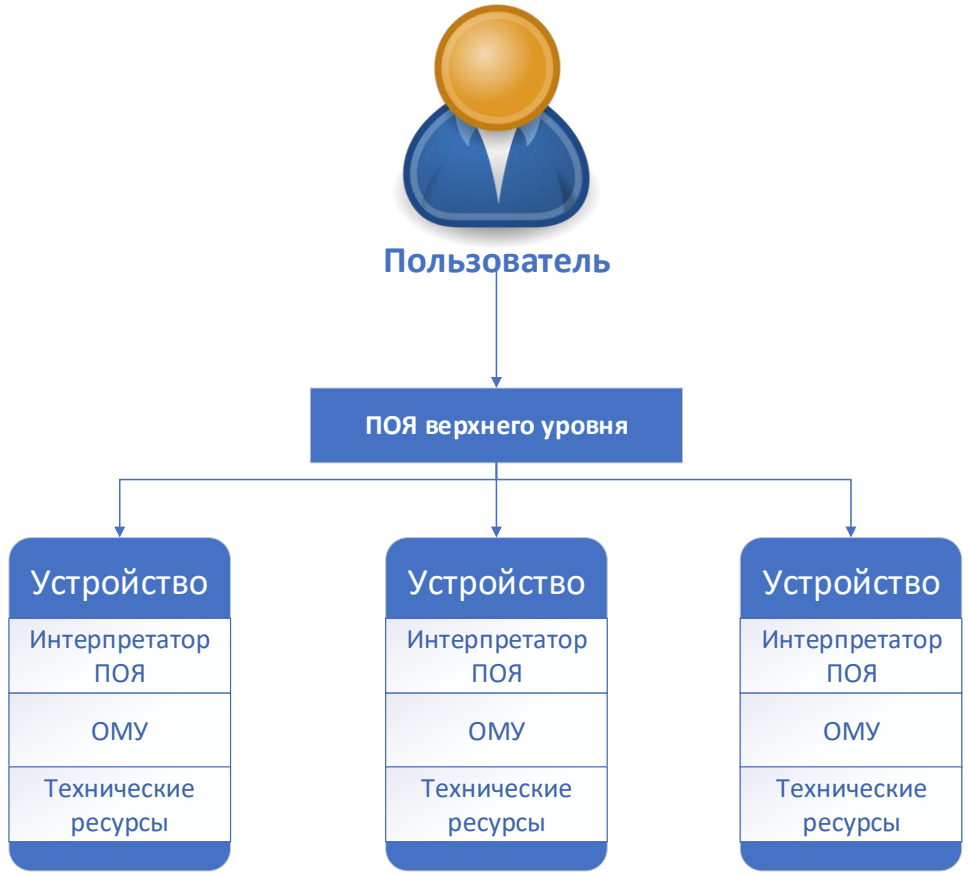
- Процесс взаимодействия с устройством разбивается на этапы,
- За каждый этап отвечает свой собственный программный компонент (сервис),
- Весь процесс взаимодействия с устройством = цепочка сервисов, каждый из которых независим от другого,
- За взаимодействие с разными устройствами отвечают разные сервисы.

Архитектура микросервисов[2][5]

- ***Преимущества:***
  - Взаимодействие с каждым устройством организовано оптимально,
  - Каждое устройство обрабатывается независимо от других;
- ***Недостатки:***
  - Необходимость согласования сервисов, ответственных за взаимодействие с одним устройством,
  - Множественные дублирования сервисов (отличаются только реализации, но суть самого сервиса – одна и та же),
  - Сложная процедура синхронизации сведений, полученных от разных устройств.

Возможное решение

- Предметно-ориентированный язык IoT верхнего уровня,
- Объектная модель устройства (ОМУ), которая отражает структуру устройства и допустимый функционал по работе с ним,
- Интерпретатор, преобразующий команды ПОЯ в команды, содержащиеся в ОМУ.



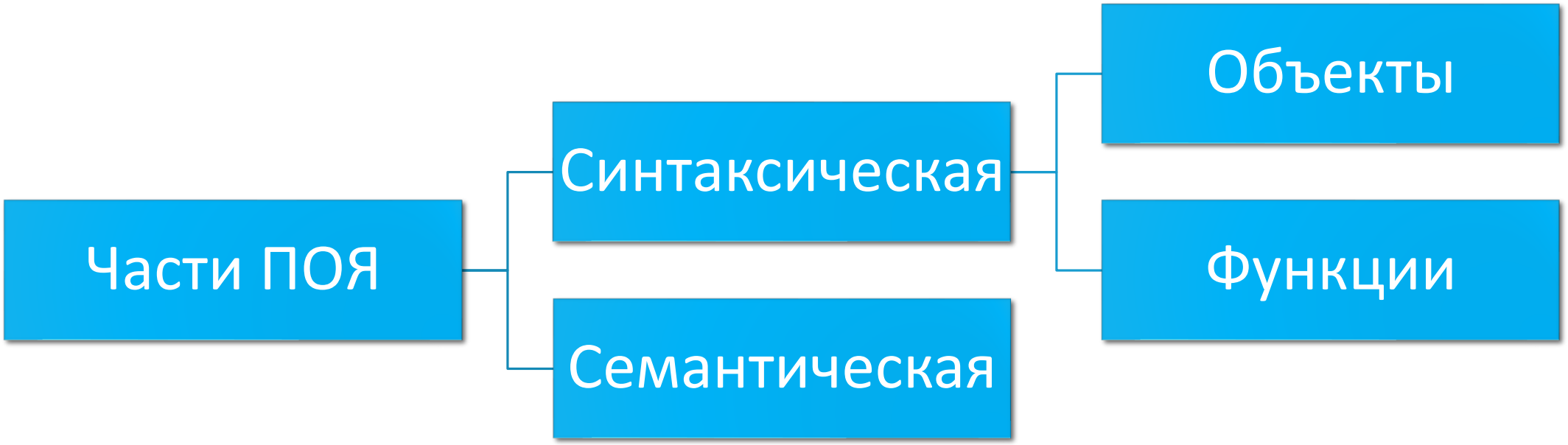
Объектная модель устройства (ОМУ)

Представляет собой тройку $(Comp, Rel, Rest)$ [6]:

- $Comp = \{Prop, Opp\}$ – множество компонентов устройства;
- Rel – множество связей между компонентами;
- $Rest$ множество ограничений (как на компоненты, так и на связи).

Существует графовое представление ОМУ

Структура ПОЯ



Графовое представление ПОЯ

ПОЯ может быть описан следующим образом:

$$V = Set \bigcup_{i=1}^{|Set|} Attr_i \bigcup_{i=1}^{|Set|} Opp_i \bigcup_{i=1}^{|Set|} SRest_i \bigcup Rel \bigcup_{i=1}^{|Set|} RRest_i$$

$$E = ESA \bigcup ESO \bigcup ESR \bigcup ERR \bigcup ESRR$$

Похожее на ОМУ графовое представление ПОЯ:

$$(Obj, Rel, \{Rest, Opp\})$$

Выводы

- Можно организовать управление устройствами в рамках IoT посредством ПОЯ внешнего уровня,
- Командам такого ПОЯ в соответствие ставится конкретная функция устройства, описанная в ОМУ,
- Преобразование осуществляется интерпретатором на основании графовых трансформаций ПОЯ и ОМУ.

Каким образом организовать трансформации между командами ПОЯ и командами ОМУ

Определение эволюции модели

С формальной точки зрения любая модель есть пара (E, R), где E – множество сущностей модели [7]:

$$e_i = \{attr_{i_1}, attr_{i_2}, \dots, attr_{i_M}\}, M \in \mathbb{N},, i = 1, N$$

и R – множество отношений/связей между ними.

Эволюция модели представляет собой процесс изменения ее структуры

Классификация модельных эволюций

- ***Вертикальная:***

  Подразумевает изменение уровня абстракции (перспективы) модели:

  $$(E^1, R^1) \textit{ является результатом вертикальной эволюции модели } (E, R) \textit{ если } |E| \neq |E^1| \textit{ и } |R| \neq |R^1|$$

- ***Горизонтальная:***

  Подразумевает фиксацию уровня абстракции модели с внесением изменений в ее детали (множество атрибутов сущностей, отношения между сущностями):

  $$(E^1, R^1) \textit{ является результатом горизонтальной эволюции модели } (E, R) \textit{ если } |E| = |E^1| \textit{ и } \exists\, r_i \in R, r_j \in R^1 : r_i \notin R^1 \vee r_j \notin R \textit{ и}$$

  $$\exists\, e_i \in E, e_i^1 \in E^1 : Attr_i \cap Attr_i^1 = Attr_i \wedge |Attr_i| \neq |Attr_i^1|$$

Выводы[1]

- ***В случае Вертикальной эволюции:***

  Необходимо **выровнять** множества **сущностей** и **отношений** между ними.

  Для этого необходимо определить 4 общих правила трансформации: 2 для переноса сущностей между моделями, и 2 для переноса отношений между ними.

- ***В случае Горизонтальной эволюции :***

  Необходимо выровнять списки атрибутов сущностей моделей и связи между сущностями.

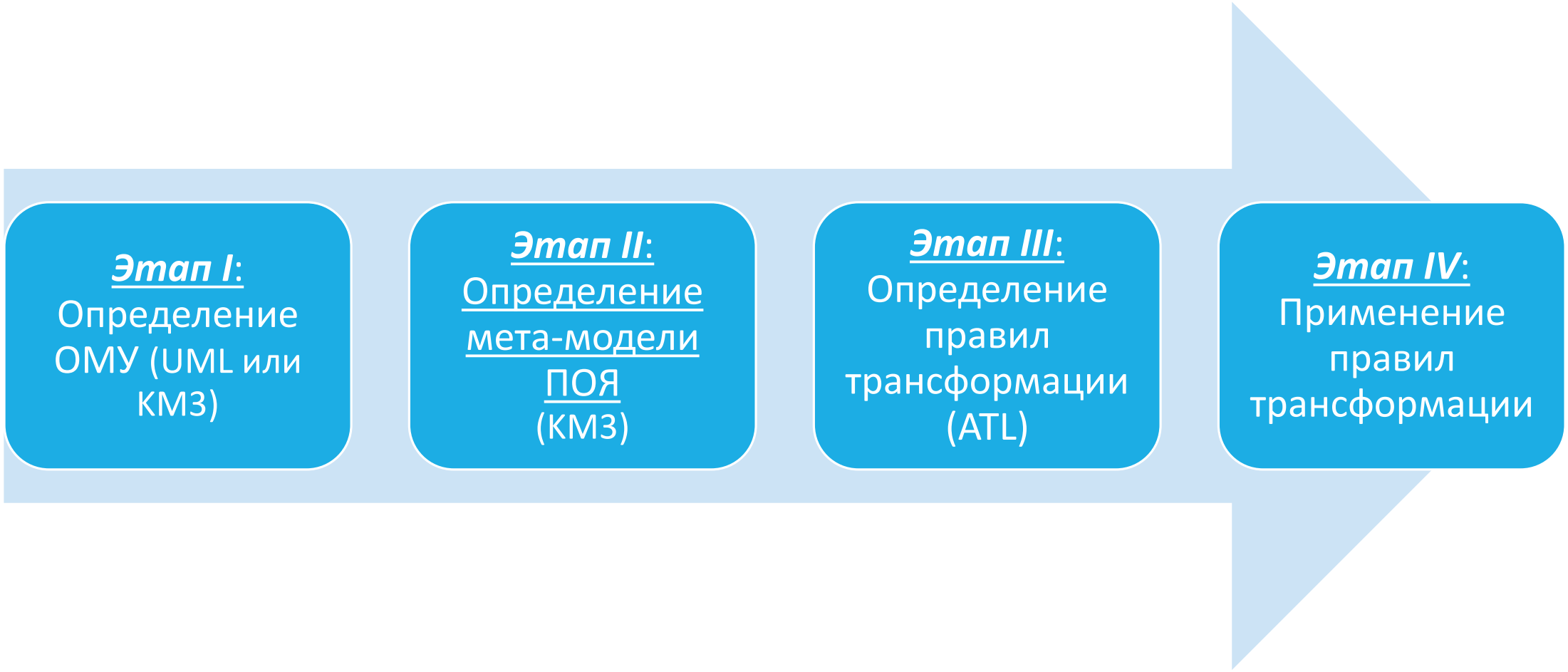
  Для этого необходимо определить 4 общих правила трансформации: 2 для переноса атрибутов сущностей между моделями, и 2 для переноса отношений между ними.

Выводы^2

- Таким образом, необходимые нам 6 правил трансформации можно объединить в 3 группы:
  - ***Правила синхронизации сущностей:***
    - добавление;
    - удаление;
  - ***Правила синхронизации атрибутов сущностей:***
    - добавление;
    - удаление;
  - ***Правила синхронизации отношений между сущностями:***
    - добавление;
    - удаление;

Реализация предлагаемого подхода

Схема внедрения



Определение ATL-правила

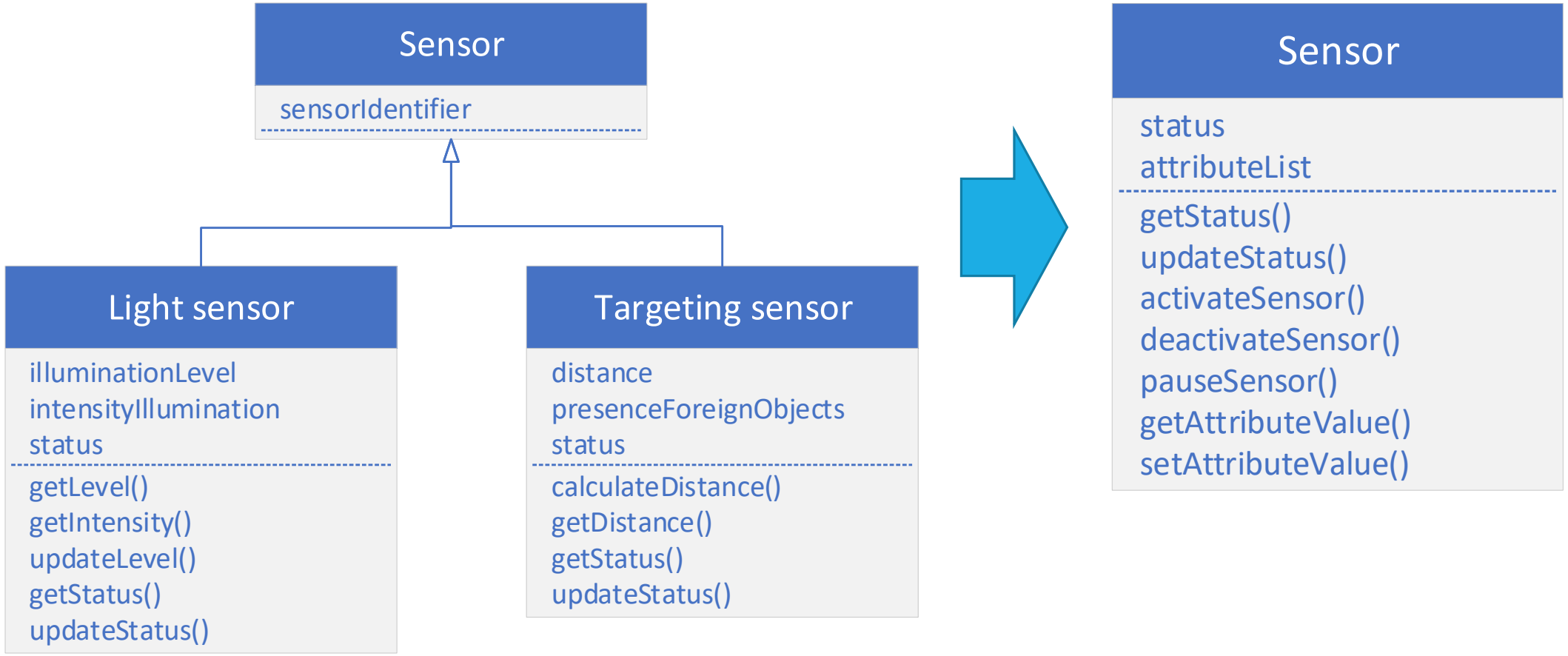
rule *rule_name* {
from *type_of_the_element_of_original_model* (*conditions_expression*)
to *action* or *type_of__the_element_of_target_model*
do { *additional_actions* } }

Пример:

rule OWLClassWithParents2DSLClassWithParents {
from a: OWL!OWLClass (a.parent->exists())
to b: DSL!DSLClass (b.name = a.name)
do { b.parent.name = a.parent.name } }

Практический кейс

Пример ОМУ и мета-модели ПОЯ

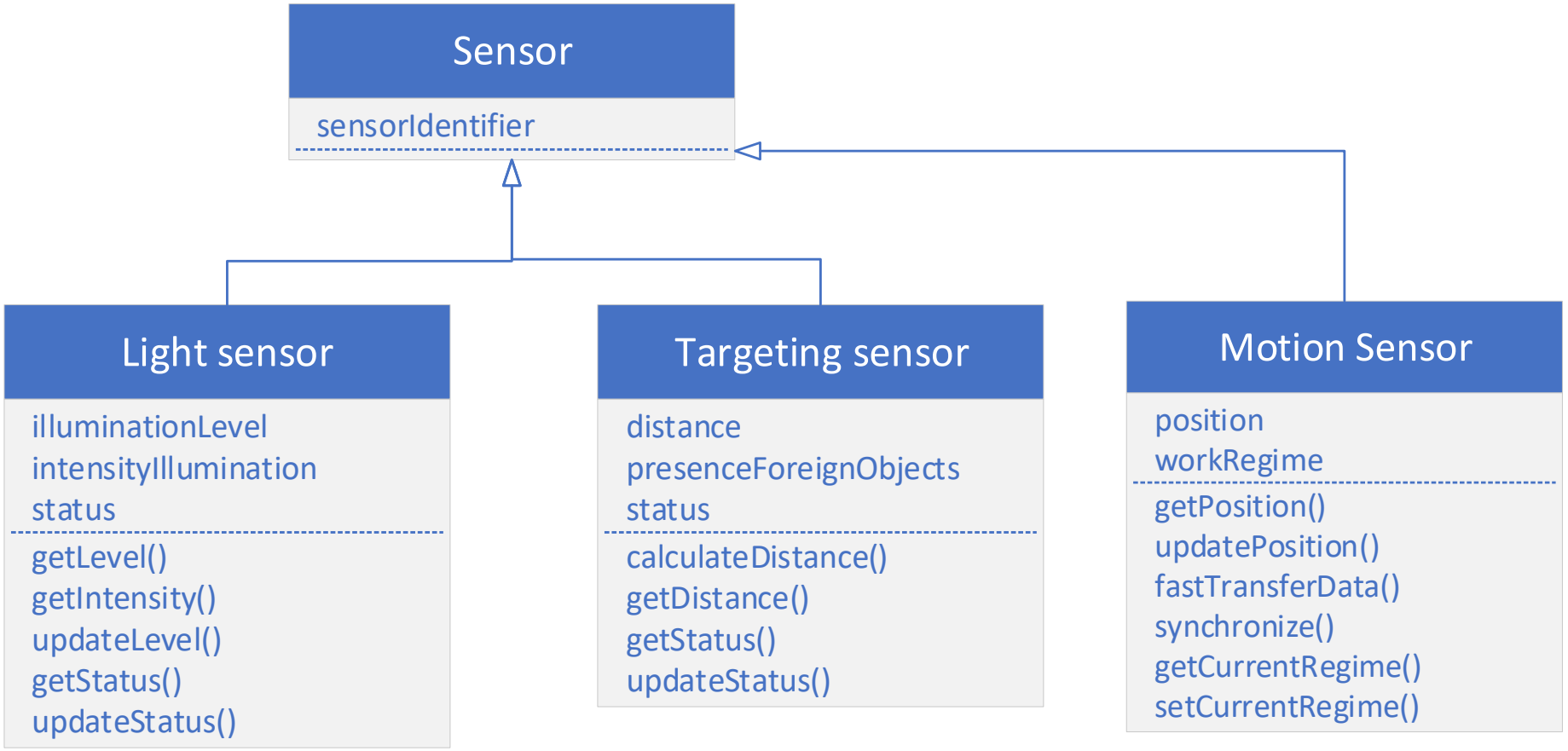


Сценарий эволюции

- В множество устройств добавлено новое, поддерживающее обновленный протокол подключения и расширенное множество функций.

- Следовательно, добавлена новая сущность в ОМУ, которая отсутствует в ПОЯ.

Сценарий эволюции



Сценарий эволюции – реализация

```
rule    DOMClass2DSLClass {
from    a: DOM!DOMClass (not exists (select b| b.isTypeOf (DSL!DSLClass) and a.name =
b.name))
to      b: DSL!DSLClass,   c: DSL!Constructor
do      {   analyseAndApplyAttributes();   }  }
rule    DOMClassAttributes2DSLClassAttributes {
from    a: DOM! DOMClass! DOMObjectProperty ( not exists (select b|
        b.isTypeOf (DSL!DSLClass!DSLObjectProperty) and a.name = b.name))
to      b: DSL!DSLClass!DSLObjectProperty (b.owner.name = a.owner.name),
        c: DSL!Constructor!DSLObjectProperty (b.owner.name = c.owner.name)   }
```

Выводы

Предлагаемый подход:

- позволяет эффективно организовать взаимодействие между разными устройствами в рамках IoT,
- обеспечивает гибкость в плане возможности внесения изменения в ПОЯ без изменения концептуальной схемы устройств и обратное изменение схемы устройств,
- позволяет осуществлять развитие ПОЯ на уровень пользователя.

Заключение[1]

- ПОЯ является гибким и удобным средством организации работы в рамках IoT;
- Для того, чтобы разрешить проблему стандартизации и гетерогенности в рамках IoT могут быть использованы графовые представления ОМУ и ПОЯ;
- Предлагаемый подход позволяет упростить организацию взаимодействий (как между устройствами, так и между пользователем и устройствами) в рамках IoT;
- В отличие от существующих решений(ex.: Cleenewerck et al. “Component-Based DSL Development”), предлагаемый подход может быть применен к любому ПОЯ и позволяет изменять и расширять ПОЯ без его повторного создания из-за изменения ОМУ.

Заключение[2]

Следующие шаги:

- Расширение предлагаемого подхода за счет добавления поддержки дополнительных компонентов в ОМУ и ПОЯ (роли пользователей и привилегии);
- Организация эволюции в режиме реального времени,
- Внедрение функционала по изменению ПОЯ на уровне пользовательских запросов.

- Перенос предлагаемого подхода на другие предметные области (ПО) – анализ применения онтологий в качестве представления концептуальной схемы ПО.

Список использованной литературы

1. Berger, U., Andersen, L. “Industry 4.0 and Smart Production”, 2017.
2. Fowler, M. “Domain specific languages”, 2010.
3. K. Ashton “That ‘internet of things’ thing“, 2009.
4. Fowler, M., Lewis, J. “Microservices”, 2016.
5. Parsons, R. “Evolutionary Architecture & Micro-Services”, 2015.
6. Sneps-Sneppe, M., Namiot, D. “On Web-based Domain-Specific Language for Internet of Things”, 2015.
7. Pereira, M., Fonseca, J., Henriques, P. “Ontological approach for DSL development”, 2016.

Спасибо за внимание!