**MOF vs rdf**

**Часть 1**

<https://github.com/bpmbpm/doc/blob/main/METAMODEL/SIRIUS/SIRIUSBPM.md>

**Часть 2**

Введение в разработку предметно-ориентированных языков (DSL) с помощью EMFText

<https://habr.com/ru/companies/cit/articles/270483/>

Это какое то непонятное колдунство. А можно MOF представить в rdf? Или есть принципиальная несводимость? Тогда это точно колдунство. Или практическая несводимость, типа того, что любой пример MOF с маппингами описывается ну не меньше чем за миллион триплетов? Тогда тоже непонятно, почему так :-)

Когда речь идет о выработке правил (маппинга) - непонятно, как они формализуются? В моем представлении, формализация => сериализация в код (регламентированную последовательность символов).

Иначе где гарантия что мы, а главное, компьютер, понимаем стрелочку и прямоугольник на диаграмме одинаково?

Описательный обычный текст не подходит в качестве сериализации - ибо он не однозначен.

++

Есть два основных подхода к моделированию:

1) Подход MOF. Он описан в стандартах OMG, на нём основано много языков моделирования типа UML, BPMN, SysML, ... Например здесь https://www.omg.org/spec/BPMN есть спецификация описывающая BPMN на человеческом английском языке. И есть формальное описание BPMN на языке MOF https://www.omg.org/spec/BPMN/20100501/BPMN20.cmof Т.е. это не какая-то нишевая штука, а вполне распространенная. Если коротко, то MOF - это язык описания языков моделирования

2) Подход Semantic Web (RDF, OWL, ...). Подход к моделированию с акцентом на концептуальное моделирование, логический вывод фактов

Оба подхода взаимозаменяемые. Например, на языке MOF можно описаать RDF и OWL https://www.omg.org/odm/ И наоборот любую MOF, UML, BPMN и т.д. модель можно описать на языке RDF или OWL. Как это может выглядеть:

1) Например, мы на языке MOF описываем язык для моделирования процессов.

1.1) В метамодели мы определяем метаклассы: процесс, шаг процесса, последовательность выполнения шагов

1.2) Определяем атрибуты. У процесса - название. У шага - название, длительность выполнения и любые другие атрибуты, какие нужны

1.3) Определяем связи. У шага ссылка на родительский процесс. У последовательности выполнения ссылка на предыдущий и следующий шаги

2) Теперь мы можем создавать экземпляры этой метамодели. Т.е. создать модель конкретного процесса с конкретными шагами

3) Если нам нужно замапить всё это в rdf, то сначала мапим метамодель в онтологию верхнего уровня. Онтология будет вида:

процесс - это класс

шаг процесса - это класс

последовательность выполнения шагов - это класс

название - это атрибут

процесс имеет атрибут название

шаг процесса имеет атрибут название

родительский процесс - это объектное свойство (ссылка)

родительский процесс имеет целевой класс процесс

шаг процесса имеет ссылку родительский процесс

...

Т.е. мапинг достаточно тривиальный

4) Затем можем мапить конкретные модели, например:

объект #1 - это процесс

объект #1 имеет название "процесс закупок"

...

Тут тоже мапинг тривиальный

Если всё так хорошо мапится в RDF, то почему бы для всего не использовать RDF? Зачем нужны MOF, UML, SysML и т.д. Это отдельная большая тема. MOF удобнее для технических моделей, ориентированных на атрибуты, причем если важен порядок этих атрибутов. Для MOF моделей удобно генерить документы, код, преобразовывать их в другие модели. Для этого есть много готовых инструментов. Для MOF моделей обычно есть удобная графическая нотация, текстовая нотация в виде DSL. Для людей это понятная вещь, куча инструментов. Плюс эти модели обычно основаны на предположении о закрытости мира https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D0%BF%D0%BE%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5\_%D0%BE%D0%B1\_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D1%81%D1%82%D0%B8\_%D0%BC%D0%B8%D1%80%D0%B0

С другой стороны RDF модели обычно основаны на предположении об открытости мира. Поэтому они хороши для концептуального моделирования, для логического вывода. Эти модели больше ориентированы на описание фактов, а не атрибутов или структуры. В онтологии мы можем указать, что у класса есть атрибуты, но обычно не задаём порядок атрибутов, не ограничиваем этот набор атрибутов. Для концептуального моделирования это отлично, но для моделирования технических систем не всегда удобно. Плюс у RDF/OWL сложно с визуализацией, сложно с генерацией кода из этих моделей и т.д.

В общем это два равнозначных подхода, которые плюс минус один в один мапятся друг в друга

++

rdf - удобнее для компьютера, , диаграммы - для человека. Я и сам удивляюсь, почему эти два мира не так сильно связаны, как ожидалось бы. Это и причина того, что я в чате, подобном этому.

Самый выбешивающий кейс для меня был в том, что диаграммы SBGN для системной биологии не семантизированы (= переведены в rdf). В результате чего их можно использовать для коммуникации биолог-биолог, но никак для моделирования.

Можно это объяснить большим разрывом, между программистами, которые понимают, что такое сериализация, автоматический логический вывод и т.д.. И экспертами любой другой области, которые понимают компьютер не больше Пушкина.

OWL для RDF совсем не обязателен. Можно обойтись SHACL и моделировать закрытый мир.

С генерацией в код не вижу принципиальных сложностей, так как AST любого языка - есть граф, а значит расписывается через rdf триплеты.

Более того, есть подозрение, что через небольшой набор sparql запросов, можно имитировать лиспообразное вычисление eval-apply и на малом числе базовых примитивов на графе (типа свойств cond, apply, eval) вывести функции для примитивов списков car, cdr и далее развить полноценный лисп с макросами. Который уже в тысячу строк способен воспроизвести интерпретатор-компилятор любого другого языка.

<https://t.me/semanticengine/1867>

MOF - это не обязательно диаграммы. Там есть четкая граница между моделью (смысловым, содержательным описанием объекта) и представлениями этой модели. Модель может быть сериализована разными способами: XML, JSON, произвольный DSL. У модели может быть много разных представлений: диаграмма, таблица, дерево, текст

Да, я не спорю, что можно всё сделать через триплеты. Просто для MOF много готовых инструментов

Например, есть язык запросов к моделям https://www.omg.org/spec/OCL/2.4/PDF (аналог SPARQL)

Язык преобразования моделей https://www.omg.org/spec/QVT/1.3/PDF (например, на нём можно описать преобразование логической модели классов в физическую реляционную модель данных)

Язык преобразования моделей в текст https://www.omg.org/spec/QVT/1.3/PDF

Инструмент для создания графических редакторов моделей https://eclipse.dev/sirius/

Инструмент для описания текстовой нотации для моделей https://eclipse.dev/Xtext/

И много всего другого

Тот же Xtext позволяет достаточно легко описать грамматику языка и замапить её на MOF-метамодель. И мы почти даром получаем парсер для этого языка в AST. Только в случае Xtext AST будет представлена не в виде RDF графа, а в виде MOF-модели. И так же мы получаем кодогенератор для этого языка, который преобразует MOF-модель в текст. Я описывал это более детально в этих статьях (только вместо Xtext на EMFText, что по смыслу то же самое):

https://habr.com/ru/companies/cit/articles/270483/

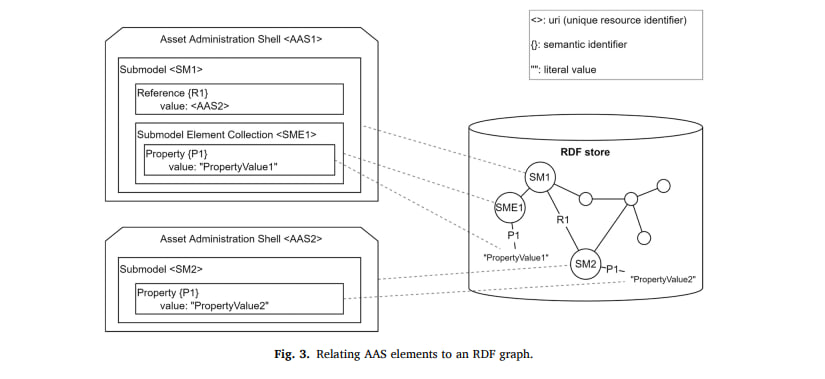
https://habr.com/ru/companies/cit/articles/271945/

Тут несколько статей (кроме последних трех) про метамодели, MOF и т.д.:

https://habr.com/ru/users/Ares\_ekb/articles/

Всё то же самое можно делать и для RDF просто это параллельное пространство моделирования https://www.sfu.ca/~dgasevic/papers/The\_Tao\_of\_Modeling\_Spaces.pdf со своим подходом, инструментарием

++



Да, это правда интересно,

Потому что я например никогда вообще не погружался в тему RDF, и по этому для меня вообще тоже было открытием насколько это разные вообще парадигмы мышления и подходов.

Я нашел несколько источников описывающих кейсы интероперабельности, и того что их можно эффективно совмещать,

скорее всего надо детальней разбирать эту работу:

https://publications.tno.nl/publication/34641154/LyQGGI/rongen-2023-modelling.pdf

Но в целом надо четко и очень хорошо представлять для чего и как.

Я пока для себя не смог как то адекватно сформулировать или найти какую то теоретическую основу которая бы примеряла эти две парадигмы, настолько они разные.

В целом накидал на форуме (https://forum.openmetamodel.org/d/56-kak-svyazany-mof-i-rdf)еще несколько примеров подобных этим, чтобы показать насколько разные подходы к мышлению в MDA и RDF

· MOF:

o Представьте, что вы моделируете библиотеку. В MOF вы бы описали её как набор классов: Книга, Автор, Читатель. У каждого класса есть атрибуты (например, у книги есть название, автор, год издания) и связи (например, книга написана автором).

· RDF:

o В RDF вы бы описали библиотеку как набор утверждений: "Книга X написана Автором Y", "Читатель Z взял Книгу X".

RDF подчеркивает связи и их значение, позволяя делать выводы и интегрировать данные из разных источников.

++

Тут у меня вопрос о разнице в самих подходах к мышлению:

- парадигме описания триплетами, когда я описываю по нарастающей отношения

- идее которая стоит за тем что мы тут обрисовали как MOF-like подход,

где сначала метамодель должна быть строго определена.

В MOF как будто мы должны все определить заранее и типизировать

В RDF подходе (с коим я детально только вчера познакомился) мы как будто сфокусированы на постепенном развитии и расширение модели данных.

Т.е. там разница в самом характере подходов к формализации.

Т.е. нечто большее чем просто разница в форматах хранения

++++++++++++++++++

Для примера реализовал VAD на нашем движке

Здесь метамодель VAD: https://metamodel.dev/metamodel/vad/dev/vad

В ней основная сущность - это цепочка добавленной стоимости (ValueAddedChain)

Она состоит из сущностей и отношений

Виды сущностей: процесс, участник процесса, ресурс, ключевой показатель, программная система

Виды отношений: обобщение, композиция, поток управления, разные виды участия, ...

Так же в метамодели описываются атрибуты сущностей и отношений

В принципе метамодель в MOF - это полный аналог онтологии предметной области (domain ontology, domain-specific ontology)

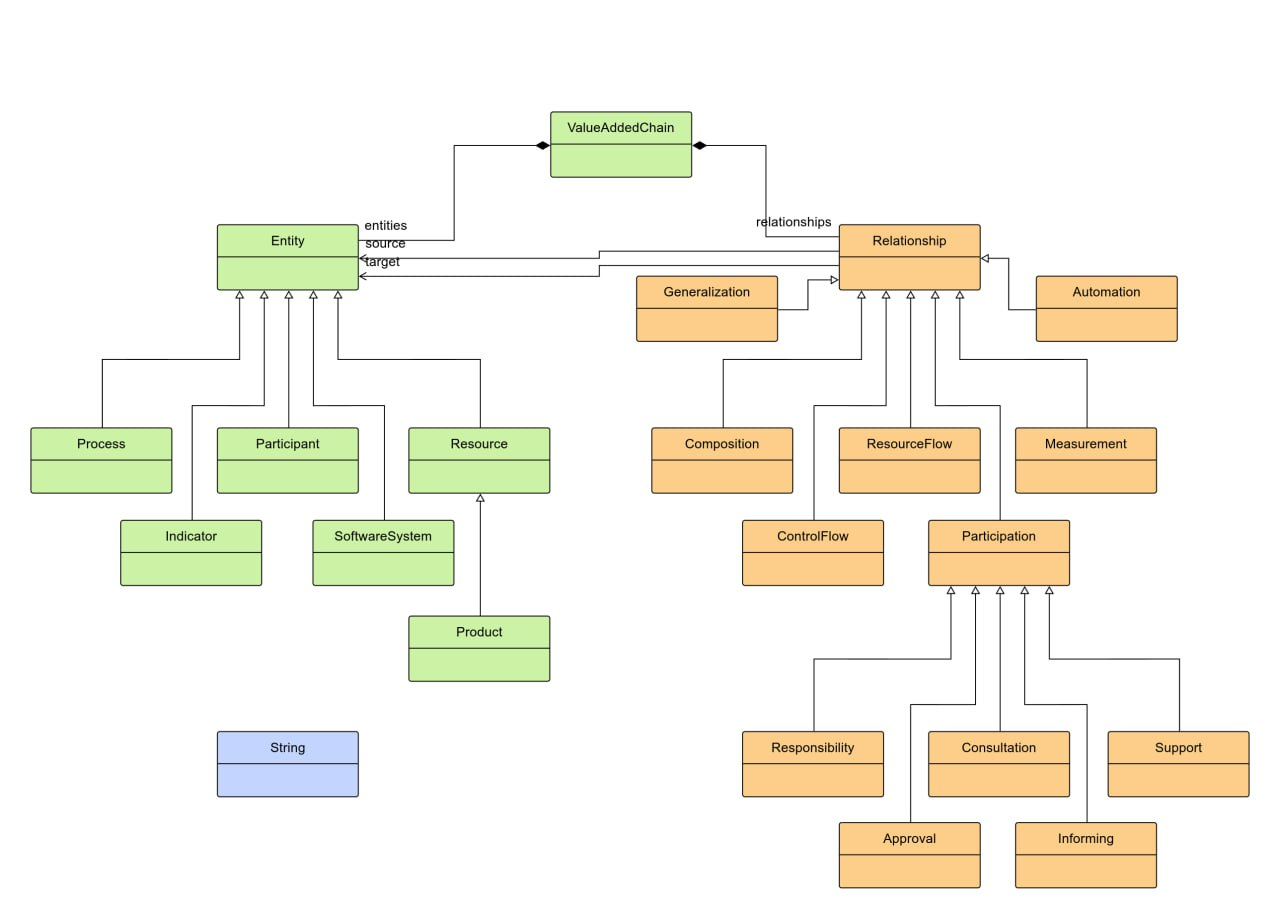
Затем на основе этой метамодели создаётся модель https://metamodel.dev/metamodel/vad/dev/sample

Причём модель и диаграмма - это две разные вещи

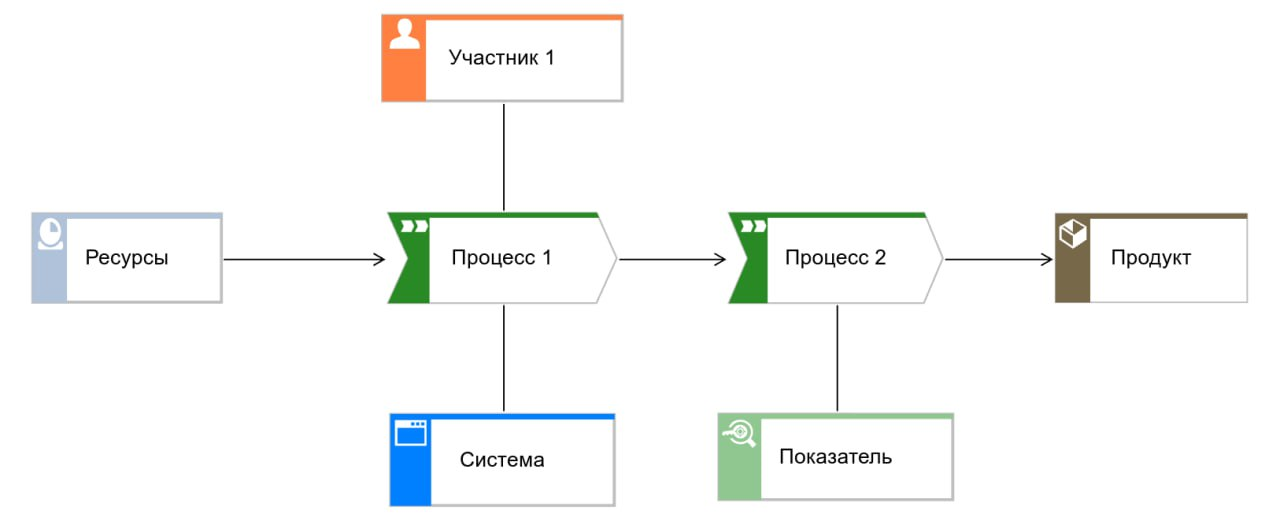
Модель содержит объекты (процесс 1, процесс 2, ...), их атрибуты и связи. Это аналог онтологии с экземплярами (наверное instance ontology)

А диаграмма содержит информацию о том с помощью каких значков эти объекты отображаются, информацию о размере фигур, их координатах и т.д.

Т.е. здесь четкое разделение между знанием, семантикой (модель) и формой их представления (диаграмма). Просто знания можно представлять не только в виде RDF графов, но и в виде MOF моделей



Продублирую здесь картинки. Это метамодель VAD



Это пример представления модели VAD в виде диаграммы, но вообще её можно представить и другими способами (таблица, дерево, просто граф без значков, текст)

завел если что отдельную ветку на форуме (https://forum.openmetamodel.org/d/57-primer-sozdaniya-metamodeli-vad) про этот новоиспеченный кейс чтобы если что тут не флудить если кто захочет задать отдельные уточняющие вопросы или разбор

Насчет примеров практического внедрения Сириуса. Вообще кроме Сириуса там целый стек технологий. Базовая технология - это Eclipse Modeling Framework (EMF), по сути это реализация стандарта OMG MOF. EMF позволяет создавать метамодели, затем создавать модели соответствующие этим метамоделям. На базе EMF построено много инструментов моделирования: Rational Software Architect, Ansys SCADE, Eclipse Papyrus, кстати 1C: EDT, ...

Но в EMF нет ничего про диаграммные представления моделей. Есть низкоуровневые движки типа Eclipse Graphical Modeling Framework (GMF). Eclipse Sirius более высокоуровневый движок для создания диаграммных редакторов моделей. Кроме диаграммных представлений у моделей могут быть текстовые представления, например, на базе https://eclipse.dev/Xtext/

Почему всё крутится вокруг Eclipse? Просто так исторически сложилось, что там собралось сообщество людей, которые создавали разные проекты в области модельно-ориентированной разработки, на базе Eclipse реализовано много инструментов моделирования и разных IDE https://en.wikipedia.org/wiki/List\_of\_Eclipse-based\_software В принципе в рамках Eclipse реализованы многие стандарты OMG в области моделирования

Насчет самого Sirius. Здесь много примеров https://eclipse.dev/sirius/gallery.html Наверное самый мощный из них это https://mbse-capella.org/ Но в основном это примеры десктопных инструментов моделирования. Последние несколько лет они активно движутся в сторону веба. Один из проектов это как-раз https://eclipse.dev/sirius/sirius-web.html Другой проект https://eclipse.dev/emfcloud/ Кстати по ссылке есть интересный пример, можно пролистать вниз до "Coffee Editor NG" и там параллельно показаны два представления одной и той же модели (в виде диаграммы и в виде текста), можно редактировать любое из них, при этом обновится и второе. Это к вопросу, что там чёткое разделение между моделью (семантикой) и разными формами её представления

Наши примеры использования этого подхода, EMF, Sirius... Мы делаем аналог ARIS https://цифровоепредприятие.рф/bpms/ (сайт почему-то очень долго открывается, это сайт заказчика) - это десктопный вариант. И ещё делаем веб-вариант https://advalange.ru/arus Тут так совпало, что и вы и мы делаем ARIS, просто разные инструменты и подходы. Делаем ещё несколько инструментов моделирования. Но это на основной работе, а в этом проекте https://metamodel.dev/ мы пытаемся сделать что-то прямо противоположное, максимально простое и облегченное

<https://t.me/semanticengine/1882>

В RDF все связи равнозначные и чтобы построить дерево нужно 1) выбрать какие связи для этого будут использоваться (общее-частное, часть-целое, ...) и 2) в каком порядке будут идти объекты на одном уровне (по алфавиту, дате добавления, в порядке выполнения функций в процессе и т.д.)

В MOF немного иначе. Там есть три фиксированных типа связей: 1) общее-частное (для наследования классов) 2) часть-целое (определяют какие объекты в какие объекты вкладываются, например, VAD-модель может состоять из групп процессов, которые могут содержать вложенные группы процессов, те содержат уже сами процессы, ...) и 3) просто ссылки

Это дерево у нас строится в соответствии со связями часть-целое в метамодели. Например, в C4 программная система состоит из контейнеров, контейнеры состоят из компонентов. Дерево таким образом и строится. Ещё могут быть группы программных систем, группы контейнеров, группы компонентов. Там замороченная метамодель:

https://metamodel.dev/metamodel/c4/dev/c4/hierarchy

https://metamodel.dev/metamodel/c4/dev/c4

Структура дерева получается такая:

Архитектура

- Группа программных систем

-- Программная система

--- Группа контейнеров

---- Контейнер

----- Группа компонентов

------ Компонент

Насчет генерации кода. Например, у нас есть такой сценарий пока не до конца завершенный:

1) Рисую модель классов (она описывает предметную область без погружения в технические детали, модель описывается в терминах - класс, атрибут класса, отношение наследования между классами, ссылка на класс)

2) Из модели классов генерю описание модели в виде документа

3) Генерю Java код

4) Преобразую модель классов в реляционную модель данных (модель в терминах - база данных, схема, таблица, столбец, первичный ключ, внешний ключ)

5) Из реляционной модели генерю SQL-скрипты

Это такой типовой сценарий использования модельно-ориентированного подхода. Он у нас реализован в десктопном моделере. В каком-то более простом виде реализуем его в веб моделере

При использовании MOF и всех этих инструментов это типовые задачи: 1) преобразовать модель из одной нотации в другую (перевести модель с одного языка на другой) 2) преобразовать модель в документ 3) в код 4) наоборот распарсить код и получить модель

MOF модели изначально древовидные, поэтому в навигаторе просто отображается содержимое модели, там нет каких-то специальных преобразований. Есть пример преобразования ER-моделей с наследованием в реляционные модели

На входе такая модель: https://github.com/AresEkb/dm/blob/main/org.example.dm.sample/sample.dm (она состоит сущностей, атрибутов и отношений)

На выходе: https://github.com/AresEkb/dm/blob/main/org.example.dm.sample/sample.rdm (модель содержит описание таблиц, столбцов, ключей)

А здесь описано преобразование 1-ой модели во 2-ую: https://github.com/AresEkb/dm/blob/main/org.example.dm.qvto/transforms/DmToRdm.qvto

Например со строки

mapping DM::Entity::toTable() : RDM::Table

начинаются правила преобразования ER-сущности в таблицу

потом дальше мапинг атрибутов в столбцы и т.д.

Я думаю для treeview нет смысла заморачиваться с кодогенерацией, преобразованием моделей и т.д. В нашем моделере он просто реализован вручную. Потому что для любых моделей он будет одинаковый, больше времени уйдет на создание кодогенератора

В разных Workflow Engine не может генериться совсем произвольный код. Например, можно описать процесс: описать функции в этом процессе, отметить какие функции выполняются вручную, какие автоматически и т.д. Затем в Workflow Engine для ручных функций сгенерится формочка для ввода данных пользователем. А для автоматических функций сгенерится код, который например, вычисляет арифметические выражения, отправляет письма и т.д. Но сгенерить таким образом компьютерную игру, Matlab или что-то сложное не получится

Кодогенерация подходит для генерации типового кода. Например, для модели данных сгенерить SQL-скрипты ( тут пример такого шаблона https://github.com/AresEkb/dm/blob/main/org.example.rdm.acceleo/src/org/example/rdm/acceleo/rdmToSql.mtl ) или Java-классы. Или например в самолетах есть вычислительные модули, которые обрабатывают данные с датчиков и есть индикаторные панели в кабине пилотов. В этих индикаторных панелях код достаточно типовой, его можно описать в модели и сгенерить код, который будет отображать разные виджеты на приборной панели. Код обмена данными между индикаторными панелями и вычислителем тоже типовой, описывается в модели и генерится

Обычно кодогенерация используется следующим образом:

1) Мы видим, что есть много однотипного кода. И написание этого кода - сплошная рутина. Например, формочки в Workflow Engine абсолютно типовые. SQL скрипты для создания базы данных типовые. Интеграционный код в самолетах типовой и т.д.

2) Вычленяем из этого кода нетиповую часть. Например, формочки в целом одинаковые, но набор полей в каждой форме свой. Определяем основные термины, которые стоят за этими элементами: процесс, шаг процесса, ручной шаг процесса, автоматический шаг процесса, последовательность выполнения шагов, данные которые указываются пользователем при выполнении шага, данные которые пользователь должен видеть при выполнении шага, ветвления в процессе и т.д.

3) Затем ищем готовую нотацию моделирования например BPMN, которая позволяет описывать то что нам нужно. Или UML Activity или EPC или ещё что-то. Если подходящей нотации нет, то можем создать свой язык моделирования

4) Описываем преобразование из моделей в код

5) Пользователи создают модели, генерят код

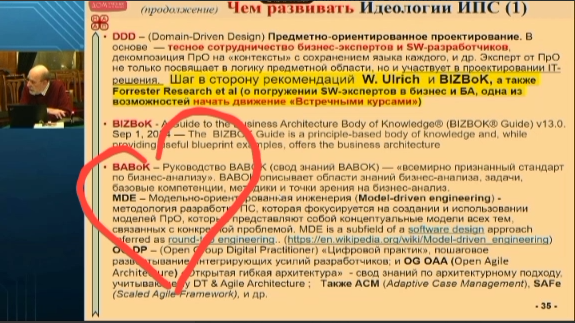
Такой подход имеет смысл только если действительно много типового кода. Если нужно создать уникальный компонент типа treeview, который получает специфические данные с сервера по специфическому API, то проще реализовать его вручную. Можно взять готовые компоненты типа https://mui.com/x/react-tree-view/ но всё равно совсем без кода не получится

У нас в навигаторе обычно фиксированная структура дерева для моделей. Например, в древе на верхнем уровне могут быть каталоги "Процессы", "Организационная структура", "Ресурсы", "Информационные системы", ... В процессах подкаталоги "Процессы управления", "Основные процессы", "Поддерживающие процессы" и т.д. Затем модели процессов, в них объект и связи

Была идея, чтобы пользователи могли сами выбирать как должно строиться это дерево: по каталогам, по детализирующим связям между моделями, по подразделениям, направлениям деятельности, типам объектов, ... Но эта идея не зашла, слишком сложно. Например, тем кто привык работать с ARIS удобнее всё раскладывать по папкам, и альтернативные деревья выглядят сложно и непонятно

Плюс в навигаторе привычно управлять доступом. Например, создать каталог с моделями для служебного пользования и ограничить к нему доступ. Это понятная полная аналогия с каталогами и файлами в файловой системе. А если вся модель - один большой граф, который в навигаторе можно показывать как угодно, то это может быть сложно для пользователей. Хотя как дополнительное представление - вполне норм, типа найти все функции в процессах построить для них иерархию. Или построить матрицу RACI

семинар ассоциации НААП «Комплексный анализ и архитектурное проектирование предприятий, информационных систем и программных продуктов» доклад Е.З. Зиндера "ИДЕОЛОГИЯ ИНЖИНИРИНГА ПРОГРАММНЫХ СИСТЕМ: ИСТОКИ, СОСТОЯНИЕ, ПЕРСПЕКТИВЫ".



**Часть 3**

<https://t.me/semanticengine/1906>

Мне тут случайно попалась информация о переходе на MOF 3

И вскользь я заметил что у них там стратегия в будущем на расширение по многим направлениям в том числе и про семантику -

Вот тут про отличия:

https://forum.openmetamodel.org/d/67-mof-251-i-mof-3/3

А тут про отличия семантического MOF SMOF от классического MOF и связь SMOF и RDF (Deepseek Search):

<https://forum.openmetamodel.org/d/68-mof-i-semantic-mof-deepseek-search>

а) Семантическое описание метаданных

SMOF может использовать RDF для сериализации метамоделей с семантикой. Например, преобразовать MO-модель в RDF-граф, обогатив её OWL-аннотациями.

RDF-триплеты способны хранить метаинформацию о структуре и правилах, определённых в SMOF.

б) Интеграция с онтологиями

SMOF-модели можно связать с OWL-онтологиями (построенными на RDF), чтобы:

Добавить доменно-специфичную логику (например, медицинские правила в FHIR).

Обеспечить интероперабельность между моделями из разных инструментов.

в) Использование общих технологий

Оба подхода могут использовать:

SPARQL для запросов к данным (RDF напрямую, SMOF — через преобразование в RDF).

SHACL для валидации моделей на соответствие семантическим ограничениям.



Ширшов

Но на состояние сейчас, мне не очень нравится идея разрабатывать на Java, если переключаться на платформу Eclipse.

Графовая модель мне нравится больше, чем объектная и реляционная. Уже много лет как нравится, тут сложно меня переубедить.

И как раз имеет смысл продемонстрировать ее мощь на примере создания приложения, которое смогут использовать удобно другие люди.

Вообще да, вот если возьмем RDF - это стандарт от W3C, а если возьмем MOF - то это стандарт от OMG.

Похоже на какие то параллельные ветки стандартизации, и интересно узнать, насколько сильно они с друг другом взаимодействуют. Может здесь какие то коммерческие интересы сталкиваются и не дают лучше договориться?

Никифоров

W3C и OMG просто занимались разными вещами. W3C - в основном вебом, и RDF - это в первую очередь язык описания ресурсов, а не язык моделирования процессов, данных или ещё чего-то. А OMG занимался стандартизацией в области моделирования. MOF - это язык описания языков моделирования таких как UML, BPMN, SysML, ... Но RDF и MOF пересекаются, потому что на RDF можно не только описывать веб-ресурсы, но и описывать более общие концептуальные модели, строить онтологии чего угодно. Плюс RDF и MOF модели мапятся друга на друга один в один

Я думаю отличия между RDF и MOF такие:

1) В базовых понятиях. В RDF всё рассматривается как триплеты, а уровни онтологий (upper ontology, domain ontology, ...) - это уже дополнительная вещь, люди сами могут выбирать в одном графе у них будут все эти уровни или в отдельных. В MOF базовая идея это уровни моделирования: мета-метамодель (это язык описания языков моделирования - MOF), метамодель (описание языка моделирования - UML, BPMN, SysML, ...), модель (модель конкретного процесса). В метамодели определяем метакласс, затем в модели могут быть его экземпляры. В MOF граница между уровнями строгая, в одной модели не могут смешиваться метаклассы и их экземпляры (хотя из этого правила есть исключения). Короче в основе этих подходов лежат просто разные базовые принципы, акцент на разных вещах. Ничто не мешает в RDF разбивать онтологии по уровням. И ничто не мешает в MOF использовать триплеты

2) В назначении. RDF больше предназначен для концептуального моделирования, для представления знаний, для использования в системах логического вывода. MOF больше предназначен для технических моделей. Но опять-таки можно и на MOF создавать концептуальные модели, и RDF использовать для технических моделей

3) В инструментарии. Для RDF в основном инструменты для логического вывода и наверное чего-то ещё, тут вы лучше знаете. Для MOF основной акцент на преобразовании моделей между разными нотациями (у них там эта тема с platform-independent и platform-specific models), на генерации документов и кода из моделей, на разных формах представления моделей (диаграммы, таблицы, деревья, текст). Но опять-таки ничто не мешает преобразовывать RDF модели или генерить из них что-то. Или прикрутить систему логического вывода к MOF

В общем в целом это взаимозаменяемые вещи. Просто исторически у авторов этих подходов были разные акценты, разные цели