Нагорных М. Э., Быков А.Н., Чернышев С.А. Кодогенерация на основе формируемой онтологии предметной области // Международный журнал информационных технологий и энергоэффективности. – 2020. – Т. 5 № 4(18) с. 15-20

Международный журнал информационных технологий и энергоэффективности

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

УДК 004.058:004.93

# КОДОГЕНЕРАЦИЯ НА ОСНОВЕ ФОРМИРУЕМОЙ ОНТОЛОГИИ ПРЕДМЕТНОЙ ОБЛАСТИ

**[1] Нагорных М. Э., [2] Быков А.Н., [3,4] Чернышев С. А.**

*[1,2] Санкт-Петербургский государственный университет аэрокосмического приборостроения, Россия (190000, г. Санкт-Петербург, ул. Большая Морская, 67, лит. А), e-mail: [1] nagornykh_max@mail.ru, [2] alexey_bykovoff@mail.ru*

*[3] Санкт-Петербургский государственный университет промышленных технологий и дизайна, Россия (191186, г. Санкт-Петербург, ул. Большая Морская, 18)*

*[4] Санкт-Петербургский государственный экономический университет, Россия (191023, г. Санкт-Петербург, ул. Садовая, 21), e-mail: chernyshev.s.a@bk.ru*

**В статье рассмотрены и проанализированы различные способы и технологии формирования онтологии предметной области, такие как: ER-модель, RDF, OWL и XML-схема. XML-схема более подходит при создании системы кодогенерации поскольку позволяет избежать внесение некорректных или излишних данных посредством механизма валидации XML-документа, заданной XML-схеме. Также рассмотрен процесс кодогенерации на основе полученной онтологии.**

Ключевые слова: онтология, кодогенерация, XML-схема, Python

# CODE GENERATION BASED ON THE GENERATED ONTOLOGY OF THE SUBJECT AREA

**[1]Nagornykh M.E., [2] Bykov A.N., [3,4] Chernyshev S.A.**

*[1,2] Saint Petersburg State University of Aerospace Instrumentation, Russia (190000, Saint Petersburg, Bolshaya Morskaya st., 67, letter A), e-mail: [1] nagornykh_max@mail.ru, [2] alexey_bykovoff@mail.ru*

*[3] Saint Petersburg State University of Industrial Technology and Design, Russia (191186, Saint-Petersburg, Bolshaya Morskaya st., 18)*

*[4] Saint Petersburg State University of Economics, Russia (191023, Saint Petersburg, Sadovaya st., 21), e-mail: chernyshev.s.a@bk.ru*

**The article discusses and analyzes various methods and technologies for forming a domain ontology, such as: ER-model, RDF, OWL and XML-schema. The XML schema is more suitable when creating a codogeneration system because it allows you to avoid entering incorrect or unnecessary data through the XML validation mechanism specified in the XML schema. Also, we reviewed the codogeneration process based on the ontology obtained.**

Keywords: ontology, code generation, XML schema, Python.

**Введение**

В реалиях нашего времени происходит активное развитие IT-сферы. Системы становятся больше и сложнее, из-за чего возникает необходимость автоматизации определенных процессов: систем поддержки принятия решений, онлайн-ботов консультантов, транспортных систем и многих других, строящихся согласно онтологии предметной области. Такой подход позволяет повысить эффективность и мобильность работы, а также упростить процесс взаимодействия между различными элементами разрабатываемых систем.

Онтология — это попытка полного описания области знаний с помощью ее абстрактного представления.[1] Онтология определяется с помощью методов, которые позволяют четко определить ее семантику и содержание. Это дает возможность вычислительным системам корректно определять поведение в различных ситуациях.

Структурно, онтология состоит из классов, объектов-классов и их взаимосвязей, которые, благодаря их методу представления, легко воспринимаются пользователем.

Таким образом, можно сделать вывод о том, способ представления предметной области при помощи онтологии дает возможность сохранить точность используемых понятий, а также их взаимоотношений друг с другом. Это является одним из главных критериев при автоматизации механизмов поддержки принятия решений

Процесс формирования описания предметной области для больших систем очень трудоемкий. А еще более проблемным процессом является описание этой области на понятном компьютеру языке, что и послужило толчком для проводимого исследования – поиск способа упрощения процесса написания кода программных продуктов, в которых для описания предметной области используется онтологический подход.

**1. Способы описания онтологии предметной области**

В настоящее время, существует множество стандартов для формирования онтологии предметной области, среди которых можно выделить: ER-модель, RDF, OWL, XML-схема.

ER-модель позволяет выделять ключевые объекты, их связи и соответствующие им ограничения. Кроме того, ER-модель обеспечивает независимость модели от способа реализации. Однако, ER-модель является узконаправленной, она решает маленький спектр задач и, преимущественно, используется для баз данных.

Другой стандарт - RDF. Он позволяет работать с метаданными, обеспечивать компьютер семантической информацией и обрабатывать эту информацию автоматически. Все объекты в нем - отдельные сущности, поэтому этап определения объектов и их отношений выполняется с меньшим приоритетом.[2] Однако RDF имеет неудобный синтаксис и крайне громоздкую структуру, что может вызывать неудобство при создании больших систем.

Язык OWL повышает функциональные возможности RDF, обеспечивая при этом совместимость с другими приложениями.[3] Однако, системы, созданные при помощи OWL крайне неудобны при проектировании больших структур и медлительны при работе с ними.[4]

Еще один вариант стандарта для формирования онтологии предметной области - XML-схема. Она формально описывает, что может содержать данный XML-документ, точно так же, как схема базы данных описывает данные, которые могут содержаться в ней: структура таблицы, типы данных, ограничения и другие.

Схема XML определяет форму или структуру XML-документа, а также правила для содержимого данных и семантику, например, какие поля может содержать элемент, какие под элементы он может содержать и сколько элементов может присутствовать. Она также может описывать тип и значения, которые могут быть помещены в каждый элемент или атрибут. Ограничения данных XML называются фасетами и включают такие правила, как минимальная и максимальная длина. [5]

Несмотря на то, что все перечисленные методы имеют ряд своих преимуществ, и отлично подходят при web-проектировании, стоит выделить, что XML-схема более подходит при создании системы кодогенерации для систем поддержки принятия решений, так как она позволяет избежать избыточность данных и более быстродейственна, благодаря особому виду представления данных - дереву знаний. Кроме того, она позволяет избежать внесение некорректных или излишних данных вследствие валидации XML-документа, заданной XML-схеме.

В общем, кодогенерация — это процесс конвертации какой-либо модели данных в исполняемый код автоматически. Данный процесс значительно оптимизирует и ускоряет процесс создания информационной системы. А главное - она исключает возможность появления неточности в полученном коде.[6]

В различных механизмах кодогенерации этот процесс происходит по-разному. Отличаются, как входные данные, так и сам процесс конвертации.

**2. Кодогенерация на основе XML-схемы**

Выбранный способ - использование XML-формата, использует в качестве входных данных XML-структуру, которая проходит валидацию, согласно заданной XML-схеме.[7]

Пример листинга используемой XML-схемы приведен ниже:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="Actor" type = "ActorFile"/>
    <xs:complexType name = 'ActorFile'>
      <xs:sequence>
          <xs:element name="Description" type="xs:string"/>
          <xs:element name="States" type="StateT"/>
          <xs:element name="Characteristics" type="CharacteristicsT"/>
      </xs:sequence>
        <xs:attribute name="name" type ="xs:string"/>
    </xs:complexType>
<!--States-->
    <xs:complexType name = 'StateT'>
        <xs:sequence>
            <xs:element name="State" type="StateFile" minOccurs="0"
             maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name = 'StateFile'>
            <xs:attribute name="name" type ="xs:string"/>
            <xs:attribute name="extra_options" type ="xs:string"/>
            <xs:attribute name="description" type ="xs:string"/>
    </xs:complexType>
<!--Characteristics-->
```

```
    <xs:complexType name = 'CharacteristicsT'>
        <xs:sequence>
            <xs:element name="Characteristic" type="CharacteristicsFile"
             minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name = 'CharacteristicsFile'>
            <xs:attribute name="name" type ="xs:string"/>
            <xs:attribute name="type" type ="xs:string"/>
            <xs:attribute name="default" type ="xs:string"/>
            <xs:attribute name="description" type ="xs:string"/>
    </xs:complexType>
</xs:schema>
```

Как можно наблюдать по первому листингу кода, XML-схема описывает модель объекта с его внутренними параметрами, как: имя, описание, состояние и характеристики. После того, как данные характеристики задаются, формируется XML-файл описания получившегося объекта. Кроме того, следует отметить, что получившийся файл полностью соответствует заданной XML схеме:

```
<Actor name="Car">
    <Description>Human vehicle</Description>
    <States>
        <State name="Go" extra_options="forward, back" description="Car
         rides" />
        <State name="Stop" extra_options="on the handbrake, on transfer"
         description="Car stands" />
    </States>
    <Characteristics>
        <Characteristic name="Color" type="Text Sequence Type"
         default="Red" description="Car color" />
        <Characteristic name="Speed" type="Numeric Type" default="90"
         description="Car speed" />
    </Characteristics>
</Actor>
```

На следующем шаге происходит кодогенерация. Для примера рассматривается использование языка программирования Python. Процесс кодогенерации начинается с импортирования встроенного модуля enum, позволяющего создавать перечисления для описания внутренних состояний класса, которые содержаться в XML-файле внутри узла «States» (Состояния).

Далее идет создание класса, название которого соответствует, атрибуту «name» внутри узла «Actor». Также указывается и описание класса в виде docstring-строки, которое соответствует узлу «Description». Следующий этап - инициализация переменных класса и их описание. Это действие выполняется в функции «init» и отражает в себе значения атрибутов узла «Characteristics». По такой же схеме создается производный класс, наследуемый от класса «Enum», который хранит в себе возможные состояния и их описание из узла «States».

Получившийся в результате кодогенерации листинг кода на языке программирования Python приведен ниже:

```
from enum import Enum

class Car:
    '''Human vehicle'''

    def init(self):
        self.Color ='Red'  # Car color
        self.Speed = 90  # Car speed


    class State(Enum):
        Go = 1  # Car rides
        Stop = 2  # Car stands
```

Сгенерированный класс описывает заданные характеристики и состояния объекта согласно его XML-файлу. При этом сам процесс генерации кода можно расширять с помощью добавления новых правил в XML-схему.

**Заключение**

Анализируя работу созданного кодогенератора, можно сделать вывод, что, благодаря данному способу описания возможно описать любую онтологию любой предметной области. Полученный код легко воспринимается человеком, не знакомым с проектом работы, поэтому дальнейшая разработка становится проще. А достаточно простым расширением XML-схемы можно увеличить границы описания предметной области. Кроме того, валидация полученного файла обеспечивает минимизацию ошибки и целостность данных.

Благодаря такому подходу можно ускорить создание информационных систем и добиться высокого уровня автоматизации трудоемких процессов проектирования.

**Список литературы**

1. Лапшин В. А. Онтологии в компьютерных системах. — М.: Научный мир, 2010.
2. James Hendler, Dean Allemang, Fabien Gandon. Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL. — 2-е изд., M.: Association for Computing Machinery and Morgan & Claypool Publishers, 2011.
3. Lee W. Lacy. OWL: Representing Information Using the Web Ontology Language. — M.: Trafford Publishing (uk) Ltd, 2006
4. Michael Uschold. Demystifying OWL for the Enterprise (Synthesis Lectures on Semantic Web: Theory and Technology). — M.: Morgan & Claypool Publishers, 2018
5. XML Documentation// Материалы с сайта https://www.w3.org [Электронный ресурс]: – Режим доступа: https://www.w3.org/TR/xmlschema-0/, свободный. – Загл. с экрана. – Яз. Англ. Дата доступа: 13 октября 2020
6. Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. Компиляторы: принципы, технологии и инструментарий — 2-е изд., М.: Вильямс, 2008.
7. Python Documentation// Материалы с сайта https://www.python.org [Электронный ресурс]: – Режим доступа: https://www.python.org/doc/, свободный. – Загл. с экрана. – Яз. Англ. Дата доступа: 9 сентября 2020

**References**

1. Lapshin V. A. Ontologies in computer systems. — M.: Scientific world, 2010.
2. James Hendler, Dean Allemang, Fabien Gandon. Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL. — 2nd ed., M.: Association for Computing Machinery and Morgan & Claypool Publishers, 2011.
3. Lee W. Lacy. OWL: Representing Information Using the Web Ontology Language. — M.: Trafford Publishing (uk) Ltd, 2006
4. Michael Uschold. Demystifying OWL for the Enterprise (Synthesis Lectures on Semantic Web: Theory and Technology). — M.: Morgan & Claypool Publishers, 2018
5. XML Documentation. Available at: https://www.w3.org/TR/xmlschema-0/ (accessed 13 October 2020)
6. Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. Compilers: Principles, Techniques and Tools. — 2nd ed., M.: Williams, 2008.
7. Python Documentation. Available at: https://www.python.org/doc/ (accessed 9 September 2020)