DOI: 10.15514/ISPRAS-2021-33(5)-1

Формальные правила продукции объектной нотации для данных, определяемых EXPRESS схемой

[1,2,3] *В.А. Семенов, ORCID: 0000-0002-8766-8454 <sem@ispras.ru>*
[1] *С.В. Аришин, ORCID: 0000-0001-6128-7082 <arishin@ispras.ru>*
[4] *Г.В. Семенов, ORCID: 0000-0003-4725-7666 <georgii.v.semenov@gmail.com>*
[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский физико-технический институт, 141701, Россия, Московская область, г. Долгопрудный, Институтский пер., 9*
[3] *Национальный исследовательский университет «Высшая школа экономики», 101978, Россия, г. Москва, ул. Мясницкая, д. 20*
[4] *Национальный исследовательский университет «ИТМО», 197101, Россия, г. Санкт-Петербург, Кронверкский проспект, д. 49, литер А*

Аннотация. В последнее время системы управления данными об изделии (PDM) широко используются в сложных междисциплинарных проектах в различных промышленных областях. PDM-системы позволяют группам проектировщиков, инженеров и менеджеров удаленно общаться в сети, обмениваться и совместно пользоваться общей информацией об изделии. Для интеграции приложений CAD/CAM/CAE с PDM-системами и обеспечения их совместимости было разработано и используется специальное семейство стандартов STEP (ISO 10303). Частью этого семейства является объектно-ориентированный язык моделирования EXPRESS, предназначенный для формального описания схем данных, а также форматы файлов для хранения и передачи данных об изделии, управляемых этими схемами. Это формат кодирования данных открытым текстом SPF и STEP-XML. В настоящее время, с развитием и широким внедрением веб-технологий, формат JSON становится все более популярным благодаря тому, что он подходит для задач обмена и хранения объектно-ориентированных данных, а также благодаря его простому и удобному для анализа синтаксису. В статье исследуется возможность применения формата JSON для представления, хранения и однозначной интерпретации данных об изделии. В предположении, что данные определяются информационными схемами на языке EXPRESS, в статье предложены и описаны формальные правила продукции объектной нотации JSON. Приводятся примеры, иллюстрирующие предложенные правила. Также обсуждаются результаты проведенных вычислительных экспериментов, которые подтверждают преимущества использования формата JSON по сравнению с SPF и STEP-XML и служат основанием для его широкого применения при интеграции программных приложений.

Ключевые слова: объектно-ориентированное моделирование; системная интеграция; интероперабельность; STEP; EXPRESS; JSON

Для цитирования: Семенов В.А., Аришин С.В., Семенов Г.В. Формальные правила продукции объектной нотации для данных, определяемых EXPRESS схемой. Труды ИСП РАН, том 33, вып. 5, 2021 г., стр. 7-24. DOI: 10.15514/ISPRAS–2021–33(5)–1

Formal Rules to Produce Object Notation for EXPRESS Schema-Driven Data

[1,2,3] *V.A. Semenov, ORCID: 0000-0002-8766-8454 <sem@ispras.ru>*
[1] *S.V. Arishin, ORCID: 0000-0001-6128-7082 <arishin@ispras.ru>*
[4] *G.V. Semenov, ORCID: 0000-0003-4725-7666 <georgii.v.semenov@gmail.com>*
[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Moscow Institute of Physics and Technology, 9, Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russia*
[3] *HSE University, 20, Myasnitskaya, Moscow, 101978, Russia*
[4] *ITMO University, Kronverksky pr. 49, bldg. A, St. Petersburg, 197101, Russia*

Abstract. Recently, product data management systems (PDM) are widely used to conduct complex multidisciplinary projects in various industrial domains. The PDM systems enable teams of designers, engineers, and managers to remotely communicate on a network, exchange and share common product information. To integrate CAD/CAM/CAE applications with the PDM systems and ensure their interoperability, a dedicated family of standards STEP (ISO 10303) has been developed and employed. The STEP defines an object-oriented language EXPRESS to formally specify information schemas as well as file formats to store and transfer product data driven by these schemas. These are clear text encoding format SPF and STEP-XML. Nowadays, with the development and widespread adoption of Web technologies, the JSON language is getting increasingly popular due to it being apropos for the tasks of object-oriented data exchange and storage, as well as its simple, easy to parse syntax. The paper explores the topic of the suitability of the JSON language for the unambiguous representation, storage and interpretation of product data. Under the assumption that the product data can be described by arbitrary information schemas in EXPRESS, formal rules for the producing JSON notation are proposed and presented. Explanatory examples are provided to illustrate the proposed rules. The results of computational experiments conducted confirm the advantages of the JSON format compared to SPF and STEP-XML, and motivate its widespread adoption when integrating software applications.

Keywords: object-oriented modeling; system integration; interoperability; STEP; EXPRESS; JSON

For citation: Semenov V.A., Arishin S.V., Semenov G.V. Formal rules to produce object notation for EXPRESS schema-driven data. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 5, 2021, pp. 7-24 (in Russian). DOI: 10.15514/ISPRAS–2021–33(5)–1

1. Введение

В последние десятилетия системы управления данными об изделии (Product data management, PDM) широко используются для реализации сложных междисциплинарных проектов в таких отраслях, как аэрокосмическая, оборонная, автомобильная промышленности, судостроение, электроника и строительство. Данные системы позволяют группам проектировщиков, инженеров и менеджеров удаленно взаимодействовать в сети, обмениваться и совместно пользоваться общей информацией об изделии, полученной с помощью приложений автоматизированного проектирования, производства и инженерного анализа (CAD/CAM/CAE). Для интеграции программных приложений и обеспечения их интероперабельности был разработан специальный международный стандарт представления и обмена данными об изделии (STEP) [1]. Его цель состоит в том, чтобы обеспечить общий для отдельных отраслей способ описания данных об изделии на протяжении всего жизненного цикла от проектирования до анализа, производства, контроля качества, инспекции, поддержки и эксплуатации. Благодаря своему нейтральному характеру такое описание подходит не только для организации файлового обмена данными, но и многопользовательской работы с использованием архивов и баз данных. Примечательно, что

многие другие отраслевые стандарты, такие как P-LIB (ISO 13584), CIS/2 (CIMSteel Integration Format), POSC/CAESAR (ISO 15926), PSL (ISO 18629) и IFC (ISO 16739), разработанные для аналогичных целей, заимствуют методологию и фундаментальные части STEP.

Стандарт STEP организован в виде нескольких частей, каждая из которых опубликована отдельно. Наибольший интерес в контексте обсуждаемой задачи представляют три из них: части 11, 21 и 28, поэтому в настоящей работе подробное рассмотрение стандарта ограничивается именно данными частями. Объектно-ориентированный язык моделирования данных EXPRESS (11 часть стандарта) определяет базовый метод описания [2,3]. Часть 21 определяет формат для записи данных в файл открытым текстом, также известный как STEP-файл или SPF [4]. Часть 28 использует язык разметки XML как для представления EXPRESS схем, так и непосредственно данных, описываемых схемами [5]. Данный формат известен как STEP-XML.

В настоящее время с развитием веб-технологий язык объектной нотации JavaScript (JSON) становится все более популярным, поскольку он подходит для задач обмена и хранения объектно-ориентированных данных [6]. JSON – это облегченный формат обмена данными в стиле "ключ-значение", который не зависит от языка программирования, при этом просто анализируется и генерируется приложениями. В то же время он легко читаем для человека. Применительно к данным об изделии эти свойства приближают его к формату SPF. В то же время JSON широко используется в качестве формата обмена данными между различными веб-приложениями, в частности, AJAX [7]. Примечательно, что JSON используется в веб-приложениях наряду с XML. Однако при использовании XML наблюдается значительное снижение производительности веб-сервисов из-за низкой эффективности чтения и анализа данных. На основе измерения показателей производительности, таких как количество отправленных объектов, среднее время передачи одного объекта, загрузка процессора и памяти, было показано, что обработка данных в JSON формате происходит значительно быстрее и обладает преимуществом над XML [8].

Несмотря на предпочтительность использования JSON по сравнению с XML для разработки веб-приложений, а также его удобочитаемости, близком к SPF, до сих пор не проводилось системных исследований, каким образом данные об изделии могут быть представлены, сохранены и однозначно проинтерпретированы с помощью объектной нотации. Целью данной статьи было восполнить данный пробел. В разд. 2 проведен краткий обзор языка EXPRESS, в котором основной акцент делается на категоризацию базовых типов данных. Пример схемы данных, описанной на EXPRESS, представлен в разд. 3. Здесь же приводится набор данных, использующий указанную схему, и представленный альтернативными способами в форматах SPF и JSON. Формальные правила для продукции объектной нотации JSON для данных, определяемых произвольной EXPRESS схемой, обобщены, систематизированы и проиллюстрированы в разд. 4. Разд. 5 посвящен вычислительным экспериментам и сравнительному анализу предложенного формата, основанного на JSON, со стандартными SPF и STEP-XML. В выводах определены перспективы стандартизации и широкого применения формата при интеграции индустриальных программных приложений, прежде всего, в области информационного моделирования зданий и сооружений (Building Information Modeling).

2. Краткий обзор языка моделирования данных EXPRESS

EXPRESS – это декларативный язык информационного моделирования, предусматривающий формальные правила описания объектов реального мира, их характеристик и ограничений. Многие конструктивные принципы языка EXPRESS были заимствованы из Ada, Algol, C, C++, Eiffel, Euler, Icon, Modula-2, Pascal, PL/I, SQL, ER и UML. Некоторые оригинальные возможности были специально привнесены для того, чтобы сделать EXPRESS более

подходящим для задач информационного моделирования. Также было разработано и стандартизировано графическое представление для подмножества EXPRESS-конструкций, называемое EXPRESS-G.

Основной конструкцией языка EXPRESS является схема, поэтому все определения появляются в контексте объявления схемы. Схема включает в себя определения типов данных и ограничений, заданных для их экземпляров. Типы данных подразделяются на следующие категории: простые типы, агрегированные типы (различные виды коллекций), перечисляемые типы, выборки, а также переопределенные типы данных, в том числе объектные типы (или сущности). Эти категории показаны на рис. 1. Предопределенные типы данных, предусмотренные EXPRESS, выделены жирным курсивом. Конструкции для пользовательских типов показаны обычным шрифтом. Стрелки указывают отношения обобщения/специализации между типами данных. В последней редакции стандарта EXPRESS для объектных типов может быть определено динамическое поведение как результат реакции на соответствующие события. Данные возможности языка не существенны для обсуждаемого круга задач и поэтому дальнейшее рассмотрение стандарта будет ограничено его первоначальной версией [3].

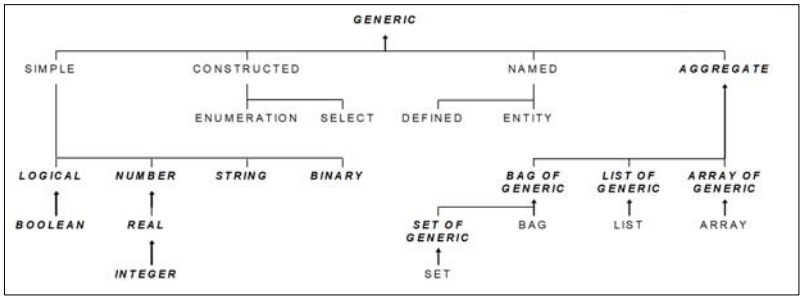

Рис. 1. Классификация типов данных языка EXPRESS
Fig. 1. Datatype categories available in EXPRESS language

Простые типы определяют атомарные единицы данных в EXPRESS. Простыми типами являются: вещественный (REAL), целочисленный (INTEGER), числовой (NUMBER), строковый (STRING), логический (LOGICAL), булев (BOOLEAN) и двоичный (BINARY).

Перечисляемые типы и выборки объявляются с использованием ключевых слов ENUMERATION и SELECT соответственно. Множеством значений ENUMERATION является множество имен, которые называются элементами перечисления. Два разных типа данных ENUMERATION могут содержать один и тот же элемент перечисления. В этом случае ссылки на элементы перечисления должны снабжаться идентификатором типа данных, чтобы однозначно интерпретироваться. Множеством значений типа данных SELECT является объединение множеств значений всех именованных типов данных, указанных в его определении. Тем самым, тип данных SELECT является обобщением каждого из указанных именованных типов данных.

Агрегации в EXPRESS описывают различные виды коллекций. Они задаются путем указания базового типа их элементов. Количество элементов может варьироваться и быть ограничено спецификацией. В EXPRESS представлены четыре основные типа агрегаций, определяемых с помощью следующих ключевых слов.

- BAG определяет неупорядоченную коллекцию, в которой разрешены повторяющиеся элементы.
- SET определяет неупорядоченную коллекцию уникальных элементов.
- LIST определяет упорядоченную коллекцию элементов, к которым можно получить доступ по их положению в последовательности. Объявление LIST может быть дополнено

с помощью ограничения UNIQUE, что подразумевает последовательность уникальных элементов.

- ARRAY определяет упорядоченную коллекцию фиксированного размера с индексированным доступом к элементам. Количество элементов в коллекции ARRAY может фиксироваться как конкретным значением, так и переменной.

Агрегации языка EXPRESS одномерны. Структуры данных, которые обычно имеют несколько измерений, могут быть представлены агрегацией, базовым типом которой является другая агрегация. Такой подход позволяет представить любое количество измерений. Например, матрица линейной алгебры может быть определена как массив массивов вещественных чисел.

Переопределенные типы данных, объявленные с помощью ключевого слова TYPE, включают типы данных с именем и с ограничениями. Множеством значений ограниченного переопределенного типа данных является подмножество значений базового типа, удовлетворяющих логическое ограничение WHERE.

Обобщенные типы данных обычно используются при определении параметров функций и процедур EXPRESS схемы. Тип данных GENERIC является обобщением всех типов данных. Тип данных AGGREGATE является обобщением всех агрегированных типов данных BAG, SET, LIST и ARRAY и их специализаций.

Объектные типы данных объявляются с помощью ключевого слова ENTITY. Объектный тип определяется в терминах атрибутов, каждый из которых характеризуется именем и множеством значений. Атрибуты могут принимать значения из указанного множества с учетом ограничений, определяемых индивидуально для отдельного атрибута или для комбинации атрибутов. Обычно значения атрибутов явно предоставляются реализацией при конструировании объектных экземпляров. Исключение делается для атрибутов, объявленных с ключевым словом OPTIONAL. Если атрибут не имеет значения, оно считается неопределенным (?). Ключевое слово DERIVED указывает, что значение атрибута должно быть вычислено предписанным способом. Ключевое слово INVERSE указывает, что значением атрибута является множество объектов, ассоциированных с данным объектом в определенной атрибутивной роли.

EXPRESS позволяет специфицировать объектные типы с помощью отношений подтипа/супертипа. В этом случае подтип наследует свойства (т. е. атрибуты, поведение, ограничения) своего супертипа. Атрибут, объявленный в супертипе, может быть повторно объявлен в подтипе с уточнением множества его значений. Например, атрибут типа данных NUMBER в супертипе может быть изменен на REAL или на тип данных INTEGER в подтипе, необязательный атрибут – на обязательный атрибут, явный атрибут – на вычисляемый атрибут, атрибут типа неупорядоченной коллекции – на атрибут упорядоченной коллекции.

Сложное наследование реализуется с помощью ограничений ONEOF, ANDOR и AND, которые устанавливают отношения между прямыми подтипами. Ограничение ONEOF устанавливает, что перечисленные в его списке подтипы являются взаимоисключающими. Каждый конструируемый объект должен иметь строго один из перечисленных подтипов и не может быть принадлежать любому другому подтипу (например, в результате конструирования составных экземпляров). Если подтипы не являются взаимоисключающими и объекты могут быть экземплярами более чем одного из перечисленных подтипов, то отношения между подтипами устанавливаются с помощью ограничения ANDOR. Если объекты могут быть подразделены на несколько групп взаимоисключающих подтипов (т. е. на несколько групп ONEOF), причем объекты каждой такой группы конструируются как составные экземпляры нескольких подтипов, то отношения между ними устанавливаются с помощью ограничения AND.

Наконец, при определении объектных типов обычно уточняется множество допустимых значений. С этой целью используются специальные ограничения, каждое из которых устанавливает одно из следующих свойств объекта:

- допустимое количество элементов для атрибута агрегатного типа;
- зависимость между значениями атрибутов или ограничение значений атрибута для данного объекта, называемые правилами домена WHERE;
- условие уникальности значений атрибутов на экстенте объектного типа (для всех объектов данного типа), называемое правилом уникальности UNIQUE;
- ограничение количества объектов, участвующих в обратном атрибуте данного объекта INVERSE;
- зависимость между экстентами нескольких объектных типов, которая определяется как глобальное правило RULE.

Рассмотренных выше принципов и конструкций языка EXPRESS достаточно для дальнейшего изложения. Более подробную информацию об языке можно почерпнуть непосредственно из стандартов [2,3].

3. Пример EXPRESS схемы и набора данных

В качестве примера рассмотрим схему ActorResource, фрагмент спецификации которой представлен в листинге 1. Диаграмма EXPRESS-G для данной схемы показана на рис. 2. Схема определяет общую предметную область для ключевых объектных сущностей Person, Organization, Address, PostalAddress, TelecomAddress, OrganizationRelationship и некоторых вспомогательных типов данных, таких как выборка ActorSelect, перечисление AddressTypeEnum, строковый тип Label и переопределенный строковый тип ActorRole.

Address объявлен как абстрактный супертип для объектных типов PostalAddress и TelecomAddress. Это означает, что Address конструируется только через его подтипы. PostalAddress и TelecomAddress наследуют как атрибуты, так и ограничения их общего супертипа Address. Это явные атрибуты Purpose, UserDefinedPupose, обратные атрибуты OfPerson, OfOrganization и правило WR1. Согласно правилу, значение атрибута UserDefinedPupose не может быть неопределенным, если атрибут Purpose принимает элемент перечисления USERDEFINED. Кроме того, PostalAddress определяет атрибут AddressLines, а TelecomAddress – атрибуты TelephoneNumbers, FacsimileNumbers, ElectronicMailAddresses, WWWUrls и собственное правило WR1.

```
TYPE ActorSelect = SELECT (Organization, Person);
  END_TYPE;
  TYPE AddressTypeEnum = ENUMERATION OF (OFFICE, HOME, USERDEFINED);
  END_TYPE;
  TYPE Label = STRING(255);
  END_TYPE;
  TYPE ActorRole = Label;
  END_TYPE;

  ENTITY Address
    ABSTRACT SUPERTYPE OF (ONEOF(PostalAddress, TelecomAddress));
      Purpose            : AddressTypeEnum;
      UserDefinedPurpose : OPTIONAL STRING;
    INVERSE
      OfPerson           : SET OF Person FOR Addresses;
      OfOrganization     : SET OF Organization FOR Addresses;
    WHERE
      WR1 : (Purpose <> AddressTypeEnum.USERDEFINED) OR
            ((Purpose = AddressTypeEnum.USERDEFINED) AND
```

```
              EXISTS(UserDefinedPurpose));
  END_ENTITY;

  ENTITY PostalAddress
    SUBTYPE OF(Address);
      AddressLines     : LIST [1:?] OF Label;
  END_ENTITY;

  ENTITY TelecomAddress
    SUBTYPE OF(Address);
      TelephoneNumbers        : OPTIONAL LIST [1:?] OF Label;
      FacsimileNumbers        : OPTIONAL LIST [1:?] OF Label;
      ElectronicMailAddresses : OPTIONAL LIST [1:?] OF Label;
      WWWUrls                 : OPTIONAL LIST [1:?] OF Label;
    WHERE
      WR1 : EXISTS (TelephoneNumbers) OR EXISTS (FacsimileNumbers) OR
            EXISTS (ElectronicMailAddresses) OR EXISTS (WWWUrls);
  END_ENTITY;

  ENTITY Organization;
      Id          : INTEGER;
      Name        : Label;
      Description : OPTIONAL STRING;
      Roles       : LIST [1:?] OF UNIQUE ActorRole;
      Addresses   : LIST [1:?] OF UNIQUE Address;
    INVERSE
      IsRelatedBy : SET OF OrganizationRelationship FOR
RelatedOrganizations;
      Relates     : SET OF OrganizationRelationship FOR
RelatingOrganization;
      Engages     : SET OF Person FOR EngagedIn;
    UNIQUE
      UR1 : Id;
    WHERE
      WR1 : SIZEOF( QUERY( temp <* Engages |
            'Director' in temp.Roles ) ) = 1;
  END_ENTITY;

  ENTITY OrganizationRelationship;
      Name                 : Label;
      Description          : OPTIONAL STRING;
      RelatingOrganization: Organization;
      RelatedOrganizations: SET [1:?] OF Organization;
    WHERE
      WR1 : SIZEOF( [RelatingOrganization] * RelatedOrganizations ) = 0;
  END_ENTITY;

  ENTITY Person;
      Id           : INTEGER;
       FamilyName   : OPTIONAL Label;
       GivenName    : OPTIONAL Label;
       MiddleNames  : OPTIONAL LIST [1:?] OF Label;
       PrefixTitles : OPTIONAL LIST [1:?] OF Label;
       SuffixTitles : OPTIONAL LIST [1:?] OF Label;
       Roles        : LIST [0:?] OF UNIQUE ActorRole;
       Addresses    : LIST [1:?] OF UNIQUE Address;
       EngagedIn    : SET OF Organization;
     UNIQUE
```



```
       UR1 : Id;
     WHERE
       WR1 : EXISTS(FamilyName) OR EXISTS(GivenName);
   END_ENTITY;

END_SCHEMA;
```

Листинг 1. Спецификация схемы ActorResource на языке EXPRESS
Listing 1. Specification of ActorResource schema in EXPRESS language

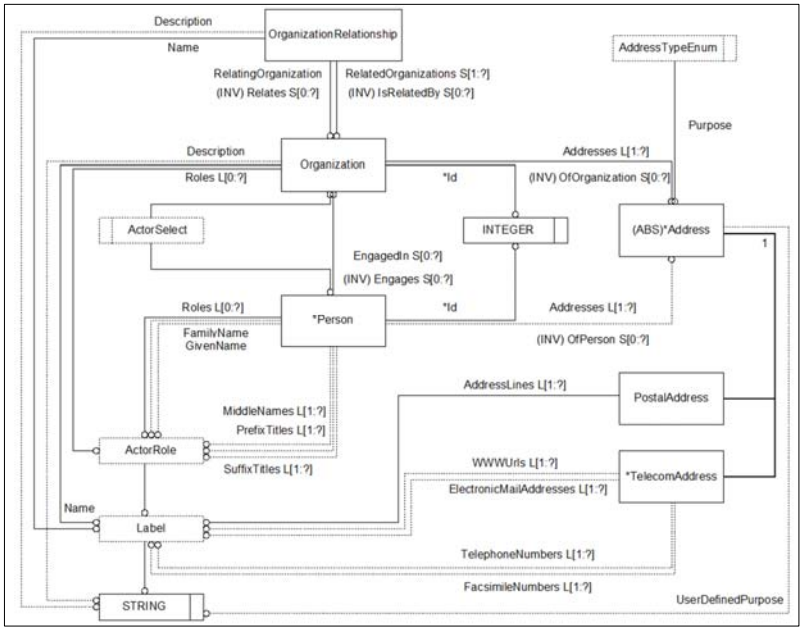

Рис. 2. Графическое представление схемы ActorResource на языке EXPRESS-G
Fig. 2. Graphical representation of ActorResource schema in EXPRESS-G language

Объектные сущности Person и Organization определяют собственные наборы явных и обратных атрибутов, а также задают ограничения на них в виде соответствующих правил. Объектный тип OrganizationRelationship определен в схеме для того, чтобы устанавливать квалифицированные атрибутивные отношения между объектами Organization. Для задания отношений «один ко многим» используются атрибуты RelatingOrganization и RelatedOrganization, первый из которых имеет тип Organization, а второй определен как непустое множество объектов типа Organization. Кроме того, используя обратные атрибуты IsRelatedBy, Relates, Engages объекта типа Organization, появляется возможность определить, с какими именно объектными экземплярами и в каких отношениях он участвует.

Для краткости опустим дальнейшие пояснения, поскольку оставшаяся часть спецификации схемы использует аналогичные конструкции.

В качестве примера рассмотрим набор данных, определяемых описанной выше EXPRESS схемой и представленных в файловом формате ASCII SPF (листинг 2). Файл организован как набор строк, каждая из которых представляет один из экземпляров объектного типа и передается в составе структуры обмена. Объектный тип экземпляра должен быть определен в EXPRESS схеме, указанной в заголовке файла. В нашем примере указанная схема – это ActorResource, а экземпляры – объекты соответствующих типов Organization, Person, PostalAddress, TelecomAddress и OrganizationRelationship, определяемых данной схемой. Каждый экземпляр представлен в структуре обмена точно один раз и имеет уникальный

идентификатор. Порядок экземпляров в структуре обмена не имеет значения. На экземпляры можно ссылаться с помощью идентификаторов из любого места структуры обмена.

```
#11 = Organization(1203, 'Automobile Inc.', 'In cars we trust.',
     ('Supply Chain Manager', 'Executive Manager', 'Sells Manager'),
     (#31, #34));
#12 = Organization(1204, 'Wheels Inc.', 'We do wheels.',
     ('Executive Manager', 'Sells Manager'), (#32, #35));
#13 = Organization(1205, 'Motor Inc.', 'Motors are essential.',
     ('Executive Manager', 'Sells Manager'), (#33, #36));

#31 = PostalAddress(.OFFICE., $, ('9292 Automobile Dr.', 'Mc Lean',
     'VA 22101'));
#32 = PostalAddress(.USERDEFINED., 'Sells Department',
     ('1172 Wheel Avenue', 'Fresno', 'CA 93711'));
#33 = PostalAddress(.USERDEFINED., 'Sells Department',
     ('1119 Motor Road', 'Reno', 'NV 89501'));
#34 = TelecomAddress(.OFFICE., $, ('678-762-2354', '678-762-2355'),
     $, ('automobile@cars.com'), ('http://www.cars.com/automobile'));
#35 = TelecomAddress(.USERDEFINED., 'Product Information',
     ('775-201-8669', '775-761-2384'), $, ('wheels@cars.com'),
     ('http://www.cars.com/wheels'));
#36 = TelecomAddress(.USERDEFINED., 'Product Information',
     ('609-639-9256', '201-213-0598'), $, ('motor@cars.com'),
     ('http://www.cars.com/motor'));

#51 = OrganizationRelationship('Consumer', $, #11, (#12, #13));
#52 = OrganizationRelationship('Supplier', $, #12, (#11));
#53 = OrganizationRelationship('Supplier', $, #13, (#11));

#61 = Person(901, 'Pringle', 'Andrew', $, ('Mr'), $,
     ('Supply Chain Manager', 'Executive Manager'), (#34), (#11));
#62 = Person(902, 'Martinez', 'Bill', $, ('Mr'), $,
     ('Executive Manager', 'Sells Manager'), (#35), (#12));
#63 = Person(903, 'Ackley', 'Chris', $, ('Mr'), $,
     ('Executive Manager', 'Sells Manager'), (#36), (#13));
```

Листинг 2. Пример набора данных, определенных схемой ActorResource и представленных в формате SPF

Listing 2. Sample dataset driven by ActorResource schema and represented in SPF format

Порядок параметров в строке экземпляра такой же, как порядок определения соответствующих атрибутов в объявлении объектной сущности. При этом атрибуты, определенные в супертипах, предшествуют атрибутам подтипов. Значение каждого параметра должно соответствовать типу данных атрибута.

В представленном наборе данных экземпляр PostalAddress с уникальным идентификатором #31 имеет следующие параметры: первый параметр «.OFFICE.» – значение атрибута Purpose, объявленного в супертипе Address, второй параметр «$» – неопределенное значение атрибута UserDefinedPurpose, также объявленного в супертипе Address, третий параметр «('9292 Automobile Dr.', 'Mc Lean', 'VA 22101 ')» – это значение явного атрибута AddressLines, объявленного в объекте PostalAddress как список строк. Здесь стоит указать, что в структуре обмена присутствуют только хранимые значения явных атрибутов, а производные и обратные атрибуты, объявленные с помощью ключевых слов DERIVED и INVERSE, игнорируются. Предполагается, что они всегда могут быть вычислены приложением по заданным явным атрибутам и поэтому нет необходимости включать их в структуру обмена.

4. Правила продукции объектной нотации JSON

JSON – это текстовый формат для представления и сериализации объектно-ориентированных данных, основанный на языке программирования ECMAScript, который в свою очередь является расширением JavaScript [6]. Данные в этом формате описываются с использованием четырех примитивных типов (строки, числа, логические значения и null) и двух структурированных типов (объекты и массивы). Объект – это неупорядоченная коллекция из нуля или более пар (членов) ключ / значение, в которых ключи представляются строками, а значения – любыми предусмотренными типами данных. Массив – это упорядоченная последовательность из нуля или более значений. Строка – это последовательность из нуля или более символов Unicode.

Данный раздел посвящен обобщению и систематизации формальных правил, посредством которых типы данных, описанные на языке EXPRESS, могут быть сопоставлены с типами данных JSON. Такое отображение должно определять непротиворечивый способ представления, хранения и интерпретации данных об изделии в формате JSON наряду с форматами SPF и STEP-XML, определяемыми действующими международными информационными стандартами.

Первые два правила носят общий характер и применяются независимо от пользовательских типов данных EXPRESS. Первое правило регулирует представление опциональных атрибутов, объявленных в схеме с использованием ключевого слова OPTIONAL. Такие атрибуты не обязаны иметь значений в объектных экземплярах. Незаданные значения кодируются в JSON значением "null", которое соответствует знаку доллара «$» в формате SPF. Заданные значения должны быть представлены в соответствии с правилами, представленными в следующих подразделах.

Второе правило касается уточнения типов данных EXPRESS. Уточнение необходимо в случаях, когда значение атрибута не может быть однозначно интерпретировано без явного указания типа данных. Обычно такие случаи возникают для атрибутов выборочного типа SELECT. Уточнение типа данных также необходимо в составных экземплярах объектных типов, объявленных с ограничениями подтипа/супертипа ONEOF, ANDOR, AND. Чтобы избежать неправильной интерпретации, простые экземпляры объектных типов должны быть снабжены именем типа данных. Для определенности и читаемости все имена типов данных указываются с использованием только строчных букв.

```
SCHEMA IllustrationResource;

TYPE Url = STRING; END_TYPE;
TYPE Png = BINARY; END_TYPE;
TYPE Pixels = LIST[1:?] OF INTEGER; END_TYPE;
TYPE Image = SELECT (Url, Png, Pixels); END_TYPE;

ENTITY Title;
  text : STRING;
END_ENTITY;

ENTITY Picture;
  figure : Image;
END_ENTITY;

ENTITY Illustration SUBTYPE OF (Title AND Picture);
END_ENTITY;

END_SCHEMA;


[
```

```
        {
           "_oid": "#1",
           "type": "Picture",
           "figure": {
              "type": "Pixels",
              "value": [0,255,255,128,128]
           }
        },
        {
           "_oid": "#2",
           "type": "Illustration",
           "_prototype":
           [
              {
              "type": "Title",
              "text": "Book"
              },
              {
              "type": "Picture",
              "figure": {
                  "type": "Url",
                  "value": "http://www.cars.com/automobile/picture.jpg"
              }
           ]
        }
     ]
```

Листинг 3. Пример уточнения типа данных EXPRESS в формате JSON
Listing 3. An example of EXPRESS datatype qualification in JSON format

Проиллюстрируем данное правило, используя схему IllustrationResource и набор данных, представленные в листинге 3. Схема определяет основные объектные сущности Title, Picture и Illustration, а также несколько вспомогательных типов данных. Тип данных Image является выборкой и объявлен как обобщение типов данных Url, Png, Pixels, каждый из которых соответствует одному альтернативному способу представления изображений. Поэтому, когда конструируется экземпляр Picture и инициируется его атрибут «figure», присвоенное значение должно дополняться именем типа Url, Png или Pixels. В приведенном наборе данных экземпляр типа Picture с идентификатором «# 1» определен как имеющий атрибут «figure», инициированный заданным списком целых чисел и квалифицированный как «Pixels». Как видно из примера, в таких случаях значения атрибутов должны быть представлены как вложенные объекты JSON с собственными членами «type», «value».

Схема IllustrationResource определяет также сложный объектный тип «Illustration», являющийся подтипом «Title» и «Picture». Это означает, что экземпляры типа Illustration состоят из простых экземпляров соответствующих супертипов. В приведенном примере экземпляр типа Illustration с идентификатором «# 2» определен как состоящий из экземпляров типов Title и Picture. Экземпляры супертипов должны быть представлены как объекты JSON и квалифицированы с помощью члена «type». Объекты включаются во внешний массив JSON, который определяется как член «_prototype» сложного экземпляра. Простым экземплярам в данном случае идентификаторы не присваиваются.

Стоит отметить, что порядок следования атрибутов в любом объекте JSON не имеет значения, однако рекомендуется первыми устанавливать атрибуты «_oid», «type» и «_prototype», а для следующих использовать порядок, в котором определены атрибуты сущности в исходной схеме. В подобном предположении удается оптимизировать обработку данных об изделии, представленных в формате JSON в соответствии с предложенными правилами.

4.1 Правила представления простых, конструируемых и агрегатных типов EXPRESS

Для простых типов данных EXPRESS определим следующие правила их представления в нотации JSON.

- Типу данных NUMBER соответствует числовой тип данных JSON. Представление числа аналогично тому, которое используется в большинстве языков программирования. Число представляется по десятичному основанию с использованием соответствующих цифр. Оно содержит целочисленный компонент, которому может предшествовать необязательный знак минус “-“, за которым может следовать дробная часть и / или экспоненциальная часть.
- Тип данных INTEGER рассматривается как целочисленное подмножество для числового типа данных JSON, а его экземпляры представляются как последовательности из одной или нескольких цифр без начальных нулей, которой может предшествовать знак минус “-“.
- Типу данных BOOLEAN соответствует логический тип данных JSON.
- Типу данных LOGICAL соответствует предопределенное перечисление строк "true", "false" и "unknown".
- Типу данных STRING соответствует строковый тип JSON, который определяется соглашениями, принятыми для семейства языков программирования C. Строка начинается и заканчивается кавычками. Все символы Unicode могут быть помещены в кавычки, за исключением символов, которые должны быть экранированы: кавычки, обратная косая черта и управляющие символы (от U + 0000 до U + 001F);
- Типу данных BINARY соответствует строковый тип JSON, значения которого представлены в закодированном виде в соответствии со стандартом Base64 [9].

Перечислимый тип данных ENUMERATION, объявленный в EXPRESS схеме, должен быть представлен как строка JSON, соответствующая одному из имен элементов перечисления. Тип данных SELECT отображается в типы данных JSON таким же образом, как и типы данных, перечисленные в его выражении. Значения этих типов данных, включая вложенные выборки, кодируются в соответствии с указанными выше правилами и должны быть снабжены полем «type» для уточнения типов выбранных значений.

Агрегатные типы данных BAG, SET, LIST, ARRAY представляются как массивы JSON. Допустимо использование неоднородных элементов разных типов данных. В массиве JSON каждый элемент должен быть закодирован в соответствии с его типом данных, объявленным в схеме EXPRESS. Порядок элементов в кодировке должен поддерживаться для упорядоченных агрегаций LIST и ARRAY. Если агрегация пуста, то она должна быть представлена как пустой массив JSON, а не как неопределенное значение, обозначенное ключевым словом "null".

Наконец, значение атрибута объектного типа ENTITY, являющееся ассоциацией на соответствующий объект данного типа, должно быть представлено в виде строки JSON, содержащей идентификатор объектного экземпляра. Напомним, что каждый объектный экземпляр представлен в структуре обмена ровно один раз.

4.2 Правила представления объектных типов EXPRESS

Структура обмена данными об изделии в формате JSON представляется как массив объектов, каждый из которых соответствует экземпляру объектного типа EXPRESS. Каждый объект представляется в виде набора пар ключ/значение.

Представление объекта должно содержать следующие элементы.

- Строковый идентификатор объекта с ключом «_oid». Его значение должно быть

уникальным в пределах структуры обмена. В зависимости от целевого приложения можно требовать и глобальную уникальность значений идентификаторов. Данный атрибут является обязательным для всех экземпляров объектных типов EXPRESS, не являющихся простыми экземплярами сложных объектных типов. Предшествующий символ подчеркивания, который используется в ключевом слове, должен предотвращать конфликты имен с возможными определениями базовой схемы. Для обеспечения согласованности с форматом SPF в качестве идентификаторов могут использоваться целочисленные значения, которым предшествует символ «#»;

- Тип объекта должен быть представлен как строковое значение JSON с ключом «type». Значение должно соответствовать имени объектного типа, определенного в базовой схеме. Данный атрибут является обязательным для всех объектов;
- Прототип объекта с ключом «_prototype» представляется как массив вложенных объектов JSON. Данный атрибут определяется только для составных экземпляров сложных объектных типов ENTITY, объявленных с использованием ограничений подтипа/супертипа ONEOF, ANDOR, AND. В таких случаях каждый простой экземпляр супертипа должен быть представлен как объект массива прототипа. Экземплярам супертипов объектные идентификаторы не назначаются. Дополнительные сведения о конструировании экземпляров сложных сущностей можно найти в стандарте языка EXPRESS [3];
- Каждый атрибут объекта соответствуют одному из явных атрибутов экземпляра объектного типа EXPRESS. В качестве ключа используется имя атрибута как оно определено в базовой схеме. Значение атрибута должно быть представлено в соответствии с правилами, описанными в предыдущем подразделе. Уточнение типа данных должно применяться во всех случаях, когда значение не может быть однозначно интерпретировано. Никаких ограничений на порядок задания атрибутов не накладывается. Обязательные атрибуты, не объявленные в определении объектного типа как OPTIONAL, обязаны быть заданы в структуре JSON объекта. Отсутствующие необязательные атрибуты считаются неустановленными. Атрибуты, объявленные в схеме как DERIVED и INVERSE, игнорируются.

4.3 Пример продукции объектной нотации на основе правил

Чтобы проиллюстрировать предложенные формальные правила, воспользуемся схемой ActorResource и набором данных из разд. 3.

```
[
   {
      "_oid":"#11",
      "type":"Organization",
      "id":1203,
      "name":"Automobile Inc.",
      "description":"In cars we trust.",
      "roles":["Supply Chain Manager", "Executive Manager",
               "Sells Manager"],
      "addresses":["#31", "#34"]
   },
{
      "_oid":"#12",
      "type":"Organization",
      "id":1203,
      "name":"Automobile Inc.",
      "description":"In cars we trust.",
      "roles":["Supply Chain Manager", "Executive Manager",
               "Sells Manager"],
      "addresses":["#31", "#34"]
```



```
      },
   {
      "_oid":"#13",
      "type":"Organization",
      "id":1203,
      "name":"Automobile Inc.",
      "description":"In cars we trust.",
      "roles":["Supply Chain Manager", "Executive Manager",
               "Sells Manager"],
      "addresses":["#31", "#34"]
      },
      {
         "_oid":"#61",
         "type":"Person",
         "id":901,
         "familyName":"Pringle",
         "givenName":"Andrew",
         "middleNames":null,
         "prefixTitles":["Mr"],
         "suffixTitles":null,
         "roles": ["Supply Chain Manager", "Executive Manager"],
         "addresses":["#34"],
         "engagedIn":["#11"]
      },
      {
         "_oid":"#51",
         "type":"OrganizationRelationship",
         "name":"Consumer",
         "description":null,
         "relatingOrganization":"#11",
         "relatedOrganizations":["#12","#13"]
      },
      {
         "_oid":"#31",
         "type":"PostalAddress",
         "purpose":"OFFICE",
         "userDefinedPurpose":null,
         "addressLines":["9292 Automobile Dr.", "Mc Lean", "VA 22101"]
      },
      {
         "_oid":"#34",
         "type":"TelecomAddress",
         "purpose":"OFFICE",
         "userDefinedPurpose":null,
         "telephoneNumbers":["678-762-2354", "678-762-2355"],
         "facsimileNumbers":null,
         "electronicMailAddresses": ["automobile@cars.com"],
         "WWWUrls":["http://www.cars.com/automobile"]
      }
]
```

Листинг 4. Набор данных, определяемый схемой ActorResource и представленный в формате JSON
Listing 4. Sample dataset driven by ActorResource schema and encoded in JSON format

В листинге 4 приведен JSON-документ с тем же набором данных, который был представлен на рис. 2 в формате SPF. В структуре обмена определены те же объекты типов Organization, OrganizationRelationship, Person, PostalAddress и TelecomAddress с теми же значениями атрибутов. Для сопоставления наборов данных идентификаторы объектов были специально назначены такими же как в SPF-файле, хотя правила допускают более общие способы.

Важно отметить, что схема ActorResource была использована для наглядности и компактности представления данных. Она является частью спецификации на языке EXPRESS международного индустриального стандарта Industry Foundation Classes (IFC) [10], который определяет около тысячи объектных типов данных и несколько тысяч вспомогательных типов для представления данных об архитектурно-строительных проектах. Очевидно, что без обобщения и систематизации формальных правил отображения типов данных из EXPRESS в JSON и уточнения способов представления данных об изделии, их обработка и интерпретация разными приложениями не представлялись бы возможными.

5. Валидация формальных правил

Предложенные правила представления данных об изделии были апробированы в ходе разработки программного приложения, предназначенного для их сериализации в формат JSON и трехмерной визуализации. С помощью приложения была выполнена серия вычислительных экспериментов и проведен сравнительный анализ с альтернативными способами представления данных об изделии в стандартных форматах SPF и STEP-XML.

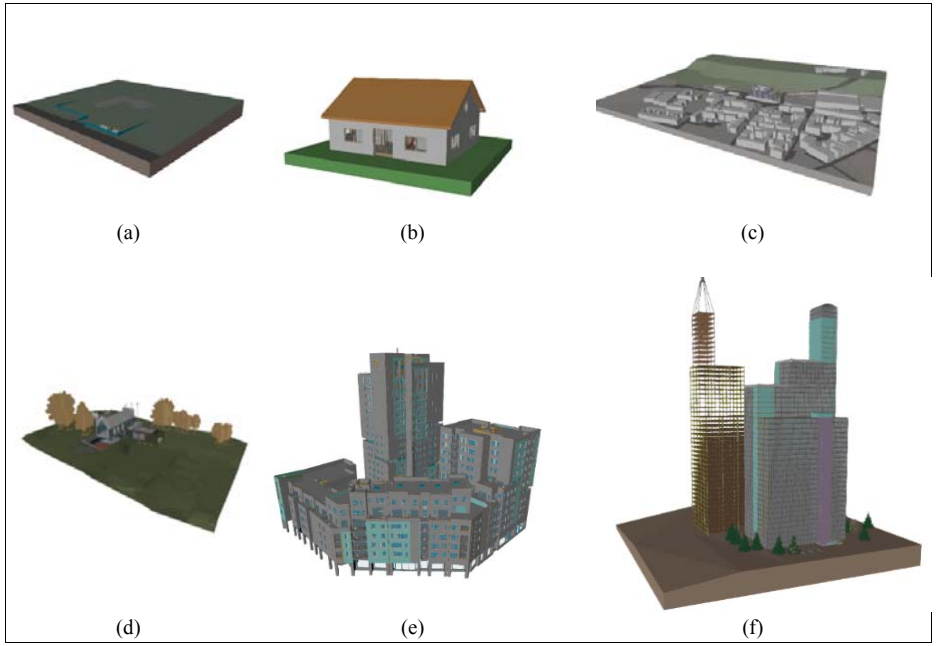

Рис. 3. Визуализация IFC-данных, использованных в вычислительных экспериментах
Fig. 3. Visualization of IFC data used in computational experiments

В качестве тестовых наборов данных для вычислительных экспериментов были использованы IFC данные разной сложности (рис. 3). Данные имелись в открытом доступе и были получены путем экспорта из Revit 2019 в формат SPF (применительно к IFC схеме часто называемом IFC форматом). Эксперименты проводились на компьютере с процессором Intel Core i7-7700 CPU, тактовая частота 3.60 ГГц, объем ОЗУ 32 Гб.

В первом эксперименте сравнивался размер файлов в указанных форматах (табл. 1) и в их архивированных вариантах (табл. 2). Видно, что размер файлов в формате JSON в среднем в два раза меньше, чем в формате STEP-XML, и приблизительно в 2 раза больше, чем в формате SPF. Архивирование нивелирует разницу между размерами файлов, однако общая тенденция не меняется: SPF файлы компактнее, чем JSON, а JSON файлы компактнее, чем XML.

Табл. 1. Размер исходных IFC данных, представленных в SPF, JSON и STEP-XML форматах
Table. 1. The size of original IFC data represented in SPF, JSON и STEP-XML formats

Набор IFC данных	SPF Kb	JSON Kb (% от размера SPF)	STEP-XML Kb (% от размера SPF)
a	952	2195 (231%)	4375 (460%)
b	2516	6054 (241%)	11556 (459%)
c	26274	56613 (215%)	113199 (431%)
d	40816	89794 (220%)	175689 (430%)
e	93160	237564 (255%)	379806 (408%)
f	173307	430514 (248%)	746400 (431%)

Табл. 2. Размер IFC данных после zip-архивирования
Table. 2. The size of IFC data after zip-archiving

Набор IFC данных	SPF zip Kb (% сжатия)	JSON zip Kb (% сжатия)	STEP-XML zip Kb (% сжатия)
a	175 (82%)	202 (91%)	237 (95%)
b	429 (83%)	545 (91%)	646 (94%)
c	4230 (84%)	4635 (92%)	5643 (95%)
d	6903 (83%)	7587 (92%)	8906 (95%)
e	16095 (83%)	19920 (92%)	23225 (94%)
f	30084 (83%)	35252 (92%)	39860 (95%)

Во втором эксперименте сравнивалось процессорное время, требуемое для считывания IFC данных в разных форматах в оперативную память компьютера. Для этого использовались парсеры JSON и XML файлов в составе среды Qt версии 5.15.1. Результаты в табл. 3 свидетельствуют о существенном ускорении работы при использовании формата JSON по сравнению с STEP-XML.

Табл. 3. Процессорное время считывания данных об изделии в форматах JSON и STEP-XML
Table. 3. CPU time for parsing product data in JSON and STEP-XML formats

Набор IFC данных	JSON CPU time (ms)	STEP-XML CPU time (ms)	Ускорение (JSON по отношению к STEP-XML)
a	38	146	3.84
b	129	564	4.37
c	824	2772	3.36
d	1281	4205	3.28
e	3124	9025	2.89
f	5626	31864	5.66

Таким образом, проведенные вычислительные эксперименты подтверждают преимущества использования формата JSON, порождаемого предложенными формальными правилами представления данных об изделии, по сравнению со стандартным форматом STEP-XML. Однако следует отметить, что в случаях, когда считывание данных в JSON сочетается с семантическим контролем, например, при получении данных PDM-системой и необходимой проверкой на соответствие EXPRESS схеме, выигрыш может быть нивелирован относительно высокими затратами на семантический контроль. В других случаях, когда такой контроль не требуется, например, при получении достоверных данных клиентом PDM-системы, скорость считывания оказывается важным фактором. С учетом существенного уменьшения объема пересылаемых данных при распределенной конфигурации PDM-системы преимущества представления данных об изделии в формате объектной нотации JSON становятся очевидными. Наконец, заметим, что стандартный SPF формат обеспечивает приблизительно такой же размер архивированных файлов, однако имеет другие

существенные недостатки, связанные со сложностью обработки и интерпретации данных web приложениями.

6. Заключение

Таким образом, в статье рассмотрены возможности применения формата JSON для представления данных об изделии, определяемых информационной схемой на языке EXPRESS. В статье представлены формальные правила продукции однозначной, не избыточной и независимой от программных реализаций объектной нотации JSON для данных об изделии. Приведены содержательные примеры, иллюстрирующие применение предложенных правил. Правила прошли апробацию в ходе разработки программного приложения, предназначенного для сериализации и трехмерной визуализации данных об изделии. Проведенные с его помощью вычислительные эксперименты подтвердили преимущества использования формата JSON по сравнению со стандартными форматами SPF и STEP-XML, что служит основанием для его более широкого применения при интеграции индустриальных программных приложений. Ожидается, что усилия по стандартизации предложенных правил и порождаемого ими файлового формата, прежде всего, в области информационного моделирования зданий и сооружений (Building Information Modeling), будут способствовать этому.

Отметим, что предложенные правила определяют способ представления данных об изделии, определяемых EXPRESS схемой, но не самой схемы. В некоторых работах [11] предпринимаются попытки представления IFC схемы в виде JSON документа, однако их следует признать довольно частными и ограниченными с учетом разнообразия конструкций языка EXPRESS. В частности, наличие алгебраических спецификаций для атрибутов DERIVED и разного рода правил WHERE, UNIQUE и GLOBAL не позволяет полноценно представить информационную схему в JSON и применить ее, например, для валидации данных [12]. Поэтому данные возможности не являлись предметом представленных исследований.

Список литературы / References

[1] ISO 10303. Industrial automation systems and integration — Product data representation and exchange.
[2] ISO 10303-11:2004. Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual.
[3] ISO 10303-11:1994. Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual.
[4] ISO 10303-21:2016. Industrial automation systems and integration — Product data representation and exchange — Part 21: Implementation methods: Clear text encoding of the exchange structure.
[5] ISO 10303-28:2007. Industrial automation systems and integration — Product data representation and exchange — Part 28: Implementation methods: XML representations of EXPRESS schema and data.
[6] T. Bray. The JavaScript Object Notation (JSON) data interchange format. Internet Engineering Task Force (IETF), Request for Comments: 8259, 2014.
[7] D. Peng, L. Cao, W. Xu. Using JSON for data exchanging in web service applications. Journal of Computational Information Systems, 2011, vol. 7, no. 16, pp. 7552-7569.
[8] N. Nurseitov, M. Paulson et al. Comparison of JSON and XML Data Interchange Formats.
[9] Base64. https://en.wikipedia.org/wiki/Base64
[10] ISO 16739-1:2018. Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries.
[11] K. Afsari, Charles M. Eastman, Daniel Castro-Lacouture. JavaScript Object Notation (JSON) data serialization for IFC schema in web-based BIM data exchange. Automation in Construction, 2017, vol. 77, pp. 24-51.
[12] V. Semenov, D. Ilyin, S. Morozov, O. Tarlapan. Effective consistency management for large-scale product data. // Journal of Industrial Information Integration, 2019, vol. 13, pp. 13-21.

Информация об авторах / Information about authors

Виталий Адольфович СЕМЕНОВ – доктор физико-математических наук, профессор, заведующий отделом системной интеграции и прикладных программных комплексов Института системного программирования им. В.П. Иванникова РАН с 2015 года. Сфера научных интересов: модельно-ориентированные методологии и инструменты программной инженерии для создания цифровых платформ и мультидисциплинарных программных комплексов, визуализация и компьютерная графика, технологии информационного моделирования в архитектуре и строительстве, проектное управление и календарно-сетевое планирование.

Vitaly Adolfovich SEMENOV – Doctor of Physical-Mathematical Sciences, Professor, Head of the Department of System Integration and Multi-disciplinary Applied Systems of the Ivannikov Institute for System Programming of the RAS since 2015. Research interests: model-driven methodologies and CASE toolkits for creating digital platforms and advanced applied systems, visualization and computer graphics, building information modeling, project management and scheduling.

Семен Васильевич АРИШИН – аспирант ИСП РАН. Научные интересы: системная интеграция, интероперабельность программного обеспечения и технологии информационного моделирования в архитектуре и строительстве.

Semen Vasilyevich ARISHIN – a postgraduate student of ISP RAS. Research interests: system integration, software interoperability and building information modeling.

Георгий Витальевич СЕМЕНОВ является студентом факультета информационных технологий и программирования Национального исследовательского университета «ИТМО». Его научные интересы включают технологии оптимистической репликации данных, математические методы сопоставления и слияния семантически сложных данных.

Georgii Vitalyevich SEMENOV is a student of the Information Technologies and Programming Faculty of the ITMO University. His research interests include technologies of optimistic replication of data, mathematical methods for differing and merging semantically rich datasets.