Р10-2014-53

И. А. Морковников

ОБРАБОТКА КОНФИГУРАЦИОННЫХ ФАЙЛОВ Sonix+

Морковников И. А. Р10-2014-53
Обработка конфигурационных файлов Sonix+

Описываются особенности работы с конфигурационными файлами в программном комплексе Sonix+. В работе перечислены недостатки комплекса, связанные с обработкой данных, и указаны пути их преодоления. Во время проведения работы была формализована структура конфигурационных файлов и разработана модель, обеспечивающая их редактирование. В документе содержатся основные алгоритмы, связанные с обработкой данных, и описание дополнительных функций, необходимых разработчикам.

Работа выполнена в Лаборатории нейтронной физики им. И. М. Франка ОИЯИ.

Сообщение Объединенного института ядерных исследований. Дубна, 2014

Morkovnikow I. A. P10-2014-53
Configuration Files Processing in Sonix+

The work is dedicated to configuration files processing in software package Sonix+. The work contains a list of software data processing drawbacks and their solutions. The structure of configuration files was formalized and the model providing editing of configurations files was developed during the work. The document contains logic of functions related with data processing and description of additional functions for developers.

The investigation has been performed at the Frank Laboratory of Neutron Physics, JINR.

Communication of the Joint Institute for Nuclear Research. Dubna, 2014

1. Sonix+

Sonix+ — программный комплекс для управления экспериментами на реакторе IBR2 [1]. Комплекс составляют следующие уровни:

• аппаратный — компоненты управления спектрометром (драйвера, серверы),

• программный — управляющие модули,

• пользовательский — высокоуровневые приложения для пользователей.

Конфигурация программного комплекса унифицирована. Настройка осуществляется программой ConfigEditor [2]. Для составления заданий на эксперимент в комплекс встроен интерпретатор языка программирования Python [3].

1.1. Конфигурация Sonix+. При создании Sonix+ предпочтение отдавалось гибкости комплекса, что повлияло на конфигурацию. В настройках комплекса устанавливаются:

• модули для загрузки,

• параметры модулей,

• параметры устройств (аппаратный уровень),

• взаимодействие комплекса с интерпретатором Python.

Для настройки комплекса используются три конфигурационных файла с разными форматами и назначением:

• настройка загрузки (папка с модулями комплекса) — содержит список модулей, подключаемых при загрузке комплекса;

• параметры модулей и устройств (папка 'configuration') — скрипт, на языке Python, содержит словарь 'config';

• взаимодействие с Python (папка py_src) — скрипт, с функциями для обращения к модулям Sonix+, содержит список устройств, их идентификаторы.

Далее будет описана работа с файлом настройки параметров модулей и устройств.

1.2. Работа с конфигурацией комплекса. Отличительной особенностью конфигурационного файла является использование метаданных, для сериализации/десериализации* данных, межпроцессного взаимодействия и взаимодействия между языками Python и C++ [5].

*Процесс перевода структуры данных в последовательность битов и восстановление начального состояния структуры данных из битовой последовательности [4].

Структура конфигурации задается разработчиками в модулях программного комплекса. Модуль добавляет свои данные в файл конфигурации. Для редактирования конфигурации применяется ConfigEditor со следующими функциями:

• отображения структуры данных в виде дерева,

• добавления, удаления и копирования объектов структуры конфигурации,

• изменения данных с проверкой типов данных.

Основные недостатки:

1) документация — отсутствует формализация структуры конфигурации;

2) для настройки Sonix+ необходимы знания, которыми обладают только разработчики программного комплекса;

3) нет возможности замены или удаления метаданных при изменении структуры конфигурации (метаданные удаляются в текстовом редакторе);

4) в ConfigEditor нет отдельной группировки для отображения устройств.

Для устранения данных недостатков необходимо модернизировать или переделать с нуля ConfigEditor с целью добавления:

• автоматической или полуавтоматической обработки метаданных,

• отображения списка используемых устройств.

2. СТРУКТУРА КОНФИГУРАЦИИ МОДУЛЕЙ Sonix+

В первую очередь необходимо преобразовать данные из текстового формата в структуру для обработки программой (провести синтаксический анализ).

Синтаксический анализ проводится с помощью формальной грамматики, описывающей конфигурационный файл. Грамматика будет записана в виде РБНФ (расширенная форма Бэкуса–Наура [6, 7]).

Формализация структуры состоит из следующих этапов:

1) выделение базовой грамматики структуры конфигурации для абстрагирования от синтаксиса файла и создание на ее основе дерева файла конфигурации;

2) создание описания типов данных, используемых для построения конфигурации;

3) выделение правил, описывающих структуру конфигурационного файла;

4) локализация грамматик для оптимизации работы с данными.

2.1. Описание базовой грамматики. Первоначально необходимо указать базовые элементы, которые будут задействованы при дальнейшем описании типов данных: letter (заглавные и строчные латинские буквы), digit (числа от 0 до 9), char (буквы, цифры и дополнительные символы):

$$
\begin{aligned}
\text{letter} = {} & \text{'a'}|\text{'b'}|\text{'c'}|\text{'d'}|\text{'e'}|\text{'f'}|\text{'g'}|\text{'h'}|\text{'i'}|\text{'j'}|\text{'k'}|\text{'l'}|\text{'m'}| \\
& \text{'n'}|\text{'o'}|\text{'p'}|\text{'q'}|\text{'r'}|\text{'s'}|\text{'t'}|\text{'u'}|\text{'v'}|\text{'w'}|\text{'x'}|\text{'y'}|\text{'z'}| \\
& \text{'A'}|\text{'B'}|\text{'C'}|\text{'D'}|\text{'E'}|\text{'F'}|\text{'G'}|\text{'H'}|\text{'I'}|\text{'J'}|\text{'K'}|\text{'L'}|\text{'M'}| \\
& \text{'N'}|\text{'O'}|\text{'P'}|\text{'Q'}|\text{'R'}|\text{'S'}|\text{'T'}|\text{'U'}|\text{'V'}|\text{'W'}|\text{'X'}|\text{'Y'}|\text{'Z'}; \\
\text{digit} = {} & \text{'0'}|\text{'1'}|\text{'2'}|\text{'3'}|\text{'4'}|\text{'5'}|\text{'6'}|\text{'7'}|\text{'8'}|\text{'9'}; \\
\text{char} = {} & \text{letter}|\text{digit}|\text{' '}|\text{'/'}|\text{'\_'}|\text{'.'}|\text{'<'}|\text{'>'};
\end{aligned}
\tag{2.1}
$$

Согласно описанию работы со структурами данных, в конфигурационном файле используются четыре базовых типа — Bool, Int, Double и String*:

$$
\begin{aligned}
&\text{Bool} = \text{'0'}|\text{'1'}; \\
&\text{Int} = [\text{'-'}], \{\text{digit}\}\text{-}; \\
&\text{String} = \text{''}, \{\text{char}\}, \text{''}; \\
&\text{Double} = [\text{'-'}], \{\text{digit}\}\text{-}, \text{'.'}, \{\text{digit}\}\text{-};
\end{aligned}
\tag{2.2}
$$

В качестве базовой грамматики можно использовать грамматику, описывающую базовые элементы конфигурационного файла (словари и списки), схожую с форматом JSON** [8], состоящую из следующих элементов:

• значение (Item) — может быть как базовым типом (Bool, Int, String, Double), так и списком (List) или словарем (Dict);

• словарь (Dict) — неупорядоченное множество пар (отделяемых запятыми), ключ:значение (ключ — String, значение — Item), заключенное в фигурные скобки;

• массив (List) — упорядоченное множество значений в квадратных скобках, где значения разделяются запятыми;

• SimpleItem — данные с простыми типами обозначены:

$$
\begin{aligned}
&\text{SimpleItem} = \text{Int}|\text{Bool}|\text{String}|\text{Double}; \\
&\text{Item} = \text{SimpleItem}|\text{List}|\text{Dict}; \\
&\text{List} = \text{'['}, \text{Item}, \{\text{','}, \text{Item}\}, \text{']'}; \\
&\text{DictItem} = \text{String,':',Item}; \\
&\text{Dict} = \text{'\{'}, \text{DictItem}, \{\text{','}, \text{DictItem}\},\text{'\}'};
\end{aligned}
\tag{2.3}
$$

Анализ структуры конфигурации основан на библиотеке json*** [9] языка Python. К парсеру добавлена логика для построения дерева с данными с помощью PyQt [10].

---

*В описании не учитывается размер типов. У Double не учитывается экспоненциальный тип записи.

**Разница заключается в надписи 'config = ', использовании двойных кавычек вместо одинарных, наличии символов пробела и новой строки. Далее в описании будет использоваться формат JSON.

***Конфигурация анализируется парсером JSON и обрабатывается на основании формальной грамматики. Из альтернативных решений — создание или генерирование (напр., Bison [11]) синтаксического анализатора.

Элементы дерева отображаются следующим образом:

1) словари (Dict) — ’ключ:Dictionary’,

2) списки (List) — ’:List’ со вложенными элементами,

3) простые данные (SimpleItem) — ’строка: Value’.

Данное представление позволяет скрывать/показывать дочерние элементы. Отображаемые объекты подписываются. Это позволяет исследовать базовые элементы структуры. Из недостатков — отсутствие отображения группировки элементов.

В дереве конфигурации есть две группы данных: конфигурация рис. 1 и структура конфигурации (словарь с ключом ”idl”) рис. 2.

2.2. Конфигурация. Исследование структуры, созданной с помощью базовой грамматики рис. 1, позволило выделить несколько закономерностей для локализации грамматики:

1) у всех словарей (кроме словаря с ключом ”__list__”) есть ключи ”type” и ”value”;

2) формат значения ключа ”value” зависит от значения ключа ”type”;

3) отдельным полем можно выделить словарь с ключами ”value” и ”type” (со значениями ”int”, ”bool”, ”double”, ”__string__”);

4) у словарей, содержащих словари и списки, ”type” отвечает за название структуры, описывающей словарь (приставка __list__ указывает, что вложен список);

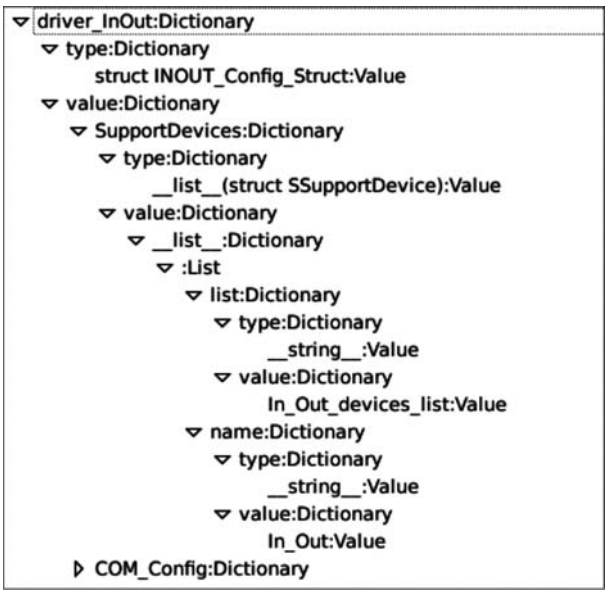

Рис. 1. Дерево конфигурации

5) в списках могут содержаться только словари;

6) есть списки с перечислением данных в одном из форматов (2.2), тип которых принимает значение int, bool, double или __string__ с приставкой __list__;

7) списки вложены в словари с ключом "__list__".

Для определения ключей словаря к уже указанным типам (2.2) добавим Key, опишем поле (DataItem) и проведем перечисление данных (DataList), у которых форматы данных зависят от типа:

```
      Key = '"', {char-' '|'.'|'/'}-,'"';
 DataItem = Key,':{"type":"int","value":',Int,'}'|
            Key,':{"type":"bool","value":',Bool,'}'|
            Key,':{"type":"double","value":',Double,'}'|
            Key,':{"type":"__string__","value":',String,'}'                    (2.4)
 DataList = '{"type":"__list__(int)","value":{"__list__":[',Int,{',',Int},']}}'|
            '{"type":"__list__(bool)","value":{"__list__":[',Bool,
                                                  {',',Bool},']}}'|
            '{"type":"__list__(double)","value":{"__list__":[',Double,
                                                  {',',Double},']}}'|
            '{"type":"__list__(__string__)","value":{"__list__":[',String,
                                                  {',',String},']}}'
```

С учетом правил построения конфигурационного файла, описанных выше, можно более строго определить часть элементов (2.3) в контексте файла конфигурации:

```
Item = DataItem |Dict;
List = '{"__list__":[',Item, {',', Item},']}';
Dict = Key,':', DataList |                                                    (2.5)
       '{"type":"__list__(struct ', Key, ')","value":', List, '}' |
       '{"type":"struct ', Key, '","value":{', Item, { ',', Item}, '}}';
```

2.3. Структуры данных. Описание данных (рис. 2) конфигурационного файла располагается в корневом словаре под ключом "idl" и обладает следующими особенностями:

1) словарь "idl" состоит из списка словарей с полем "type" со значением "__list__(struct IDL_struct)";

2) у словарей есть два идентичных поля "unit_type" и "type_name", в которых хранится название структуры, а также поля "type" (со значением "class") и "fields" (со списком полей описываемой структуры и полем "type", равным "__list__(struct IDL_field)");

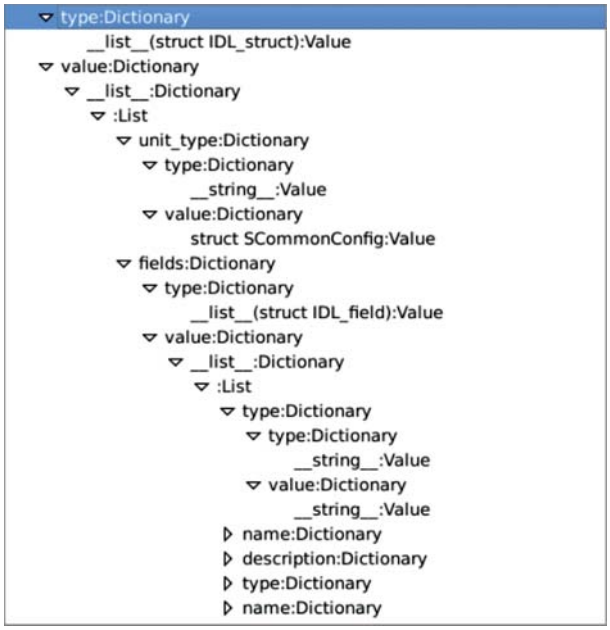

Рис. 2. Дерево метаданных конфигурации

3) в списке полей структуры каждый элемент состоит из трех элементов (по структуре аналогичных DataItem (2.4)) с ключами ”name” (название поля), ”type” (формат данных в поле) и ”description” (описание поля).

У всех элементов структуры жестко задан тип и набор дочерних элементов, что позволяет сделать детальное описание каждого элемента.

В списке полей структуры элемент ”type” является описанием для эквивалентного объекта в конфигурации файла. Известны все используемые значения данного поля:

$$\begin{aligned} \text{IdlType} = {} & \text{'''int'''} \mid \text{'''bool'''} \mid \text{'''double'''} \mid \text{'''\_\_string\_\_'''} \mid \\ & \text{'''struct ', Key, ''''} \mid \text{'''\_\_list\_\_(struct ', Key,')'''} \end{aligned} \tag{2.6}$$

В полях структур используются уже описанные типы (2.2), (2.4) и (2.6). Далее идет описание строения поля и их группировки в список:

$$\begin{aligned} \text{IdlFieldItem} = {} & \text{'\{ ”type”: \{”value”:', IdlType, ',”type”:”\_\_string\_\_”\},} \\ & \text{”name”:\{”value”:', Key,',”type”:”\_\_string\_\_”\},} \\ & \text{”description”:\{”value”:', String, ',”type”:”\_\_string\_\_”\}\}'} \\ \text{IdlField} = {} & \text{'''fields”:\{ ”type”:”\_\_list\_\_(struct IDL\_field)”,} \\ & \text{”value”:\{”\_\_list\_\_”:[', IdlFieldItem, \{',', IdlFieldItem\},']\}\}'} \end{aligned} \tag{2.7}$$

Далее описаны поле с названием структуры, элементы словаря ”idl” и сам словарь:

```
IdlName = '{"value": "struct ', Key, '", "type":"__string__"}'
IdlDictItem = '{ "type_name":', IdlName, ', "unit_type":', IdlName, ',
                 "type":{"value":"class","type":"__string__"},', IdlField, '}'
IdlDict = '"idl":{ "type":"__list__(struct IDL_Struct)",
                   "value":{"__list__":[', IdlDictItem, {',', IdlDictItem}, ']}}'
```
(2.8)

3. ОБРАБОТКА КОНФИГУРАЦИИ

Структуры конфигурационного файла достаточно для проведения синтаксического анализа. Для обработки данных требуется учитывать структуру конфигурации.

Обработка конфигурации состоит из таких задач, как:

1) синтаксический анализ конфигурационного файла;

2) создание структуры, учитывающей связи между данными в конфигурации;

3) создание пользовательского представления для обработки данных;

4) удаление неиспользуемых структур из метаданных;

5) генерация текстового файла с конфигурацией.

3.1. Структура представления конфигурационного файла. Обработка конфигурации состоит из нескольких этапов, на каждом из которых создается представление для дальнейшей обработки данных (рис. 3):

- парсинг текстового файла,
- создание модели представления данных,
- отображение данных пользователям.

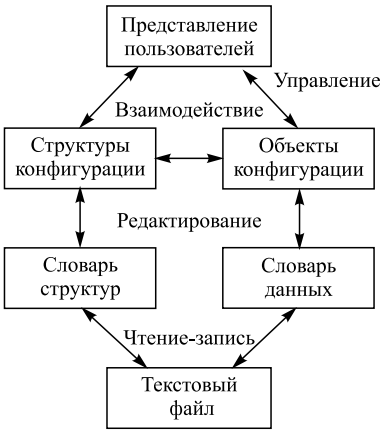

Рис. 3. Уровни обработки конфигурации

У каждого представления свое назначение:

• текстовый файл — набор входных данных и результат обработки,

• словарь — хранение данных в программе,

• объекты* — обработка связей между данными,

• представление для пользователей — отображение данных.

3.2. Объекты структур данных. Структуры данных в словаре Python располагаются под ключом "idl". Они обладают следующим набором атрибутов (2.8): "unit_type" и "type_name" (названия структуры идентичны), "type" и "fields" (список полей описываемой структуры).

У каждого поля есть три атрибута type, name и description (2.7). В поле хранится описание дочерней структуры либо поля с данными.

Обработка структур данных заключается в проверке и удалении неиспользуемых структур, которые делятся на три типа:

• структуры без объектов — структуры, на которые не ссылаются объекты;

• дублирующие структуры — структуры с одинаковыми названиями;

• идентичные структуры — дублирующие структуры с идентичными полями.

Если нет объектов, ссылающихся на структуру, значит, она не используется. При ее удалении дочерняя структура может стать структурой без объектов (в случае, когда на нее ссылается только удаленная структура).

В конфигурации может находиться несколько структур с идентичными названиями и разными атрибутами. Возможно использование только одной из таких структур.

Идентификацией объектов должен заниматься разработчик. Целью программы является выделение определенных объектов и удаление тех, на которые укажет разработчик.

Для идентификации структуры объект должен содержать название структуры (достаточно значения Key объекта IdlName (2.8)) и список полей со значениями.

Поиск структур без объектов основан на подсчете ссылок на родительские элементы и объекты структуры.

Обработка дублирующих структур (рис. 4) осуществляется с помощью словаря (struct_links), где ключ — название структуры, а значение — список структур. Программа проверяет количество элементов у всех ключей словаря. Если в списке больше одного элемента, все элементы списка помечаются как дублирующие (duplicate).

---

*Структура конфигурации представлена в виде ориентированного графа (объекты — вершины, ссылки — ребра). В зависимости от целей ее можно представить в виде древовидной структуры, или графа.

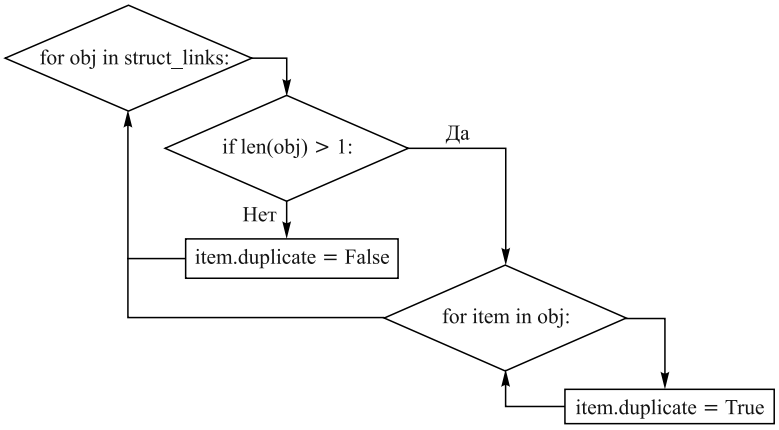

Рис. 4. Поиск дублирующих структур

Идентичные структуры, в отличие от дублирующих структур, содержат поля с идентичными значениями. Это позволяет провести проверку атрибутов дублирующих структур и удалить повторяющиеся данные (в результате останется одна структура с необходимым набором полей).

При проверке структур (рис. 5) проводится их линковка (structure_linking), после чего структуры проверяются на наличие объектов* (check_objects) (рис. 6). Для проверки структур используются списки родительских структур (parents) и объектов (objects), ссылающихся на структуру.

При линковке алгоритм перебирает (рис. 7) все структуры из словаря (struct_links) по очереди два раза — в первый раз очищает списки с родительскими структурами, после чего проверяет все поля структуры рис. 8 и устанавливает родителей для дочерних структур.

При удалении одной структуры без объектов может появиться несколько таких же структур. Для определения всех структур без объектов найденные структуры (с флагом no_objects) не используются. Алгоритм поиска структур без объектов повторяется до тех пор, пока не будет найдено ни одной структуры без объектов (позволяет отобразить пользователю сразу все структуры без объектов).

Для отображения и изменения данных, а также поддержания связей между объектами объект структуры должен содержать следующую информацию:

1) ключ — поиск данных;

2) имя и список полей — идентификация и отображение структуры;

*При наличии дублирующих структур может использоваться только одна из них. При линковке и проверке в качестве дочерней используется первая в списке дублирующая структура. Однако для всех дублирующих структур устанавливаются связи с дочерними элементами.

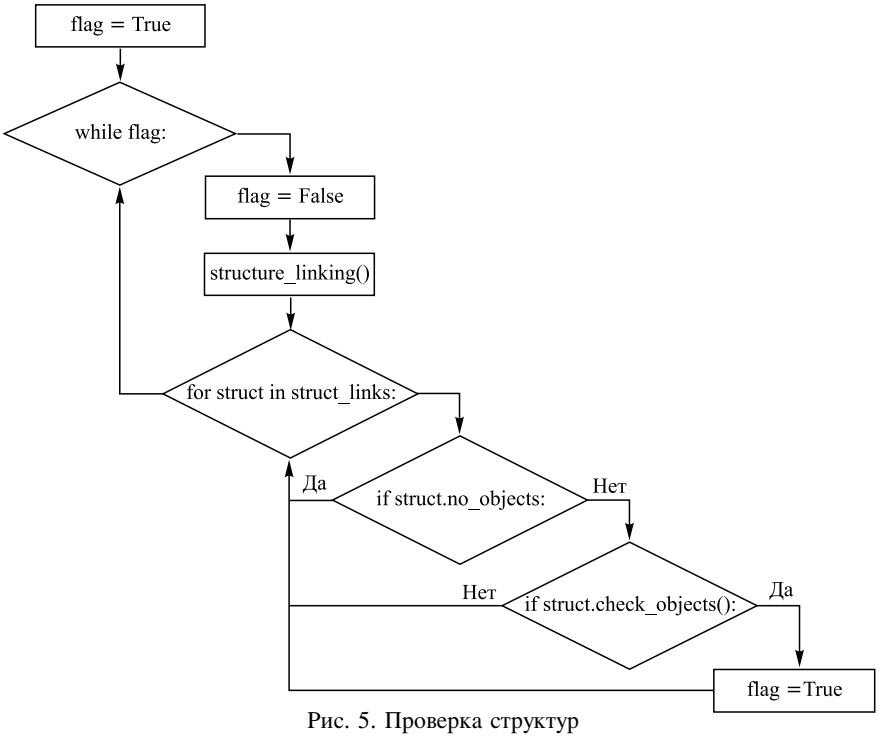

Рис. 5. Проверка структур

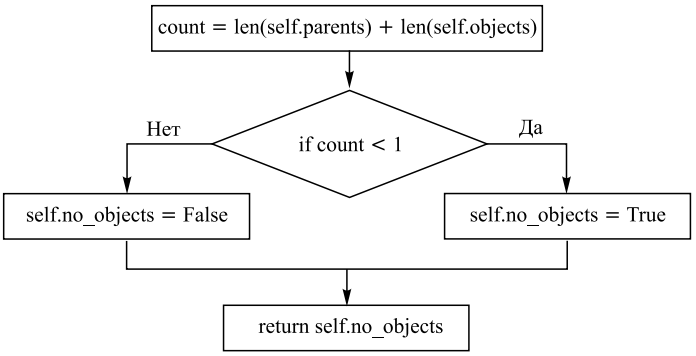

Рис. 6. Проверка структур на наличие объектов

3) ссылка на область словаря с данными — для удаления структуры;

4) списки объектов и структур, ссылающихся на структуру, — поиск структур без объектов;

5) флаги состояний — определяет, является ли структура дублирующей или без объектов.

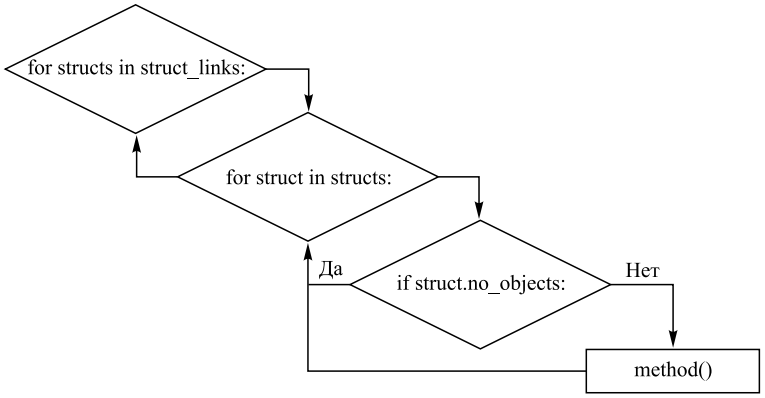

Рис. 7. Обход структур

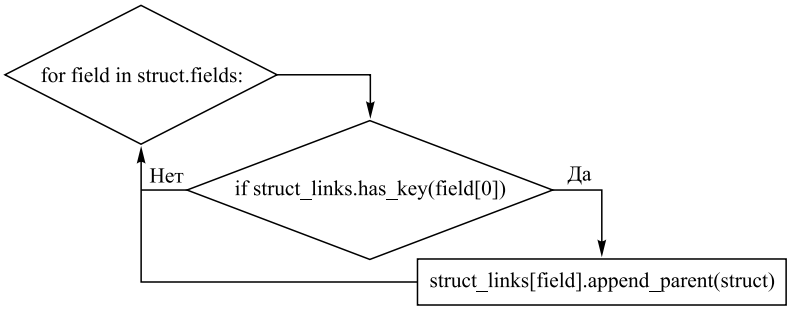

Рис. 8. Линкование структур

3.3. Представление данных. Словарь (Dict) — общая структура (2.5), содержащая остальные (DataItem — обычное поле, DataList — список с данными, List — список) типы данных. Все типы данных схожи со словарем, но в их строении есть отличия.

Данные были разделены на две группы:

• поля с информацией — DataItem и значения DataList,

• объекты — Dict, List и DataList (рис. 9).

Поля с информацией используются для хранения информации об объекте и находятся в нем. Свойства DataItem и значения DataList схожи. У них есть атрибуты типа, значения, и имени. Разница заключается в представлении данных.

DataItem (2.4) представлен в виде словаря с ключом, который используется как атрибут имени, полем type — тип и value — значение.

Значения DataList (2.4) представлены в виде списка. Для определения типа используется тип DataList. Значение определяется значением элемента списка, а в качестве имени можно использовать номер элемента.

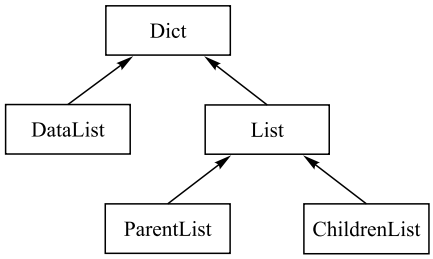

Рис. 9. Объекты данных

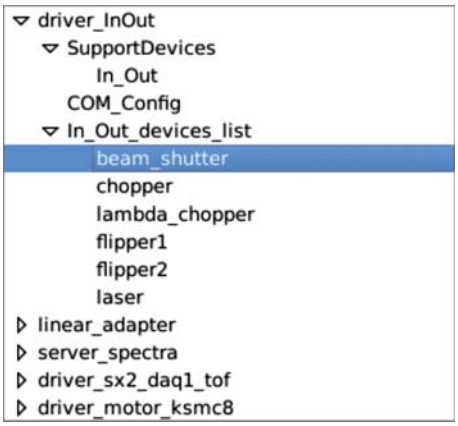

Рис. 10. Дерево данных

В результате поля с данными можно представить с помощью атрибутов тип, имя и значение. При этом, в зависимости от типа поля, данные будут обрабатываться по-разному.

Объекты используются для построения структуры конфигурации. В качестве основного элемента можно выделить Dict (2.5). Данный элемент при небольшой модификации поиска и обработки данных можно представить как List или DataList.

У объекта Dict есть набор полей и дочерних элементов, на которых строится структура конфигурации, а также ссылка на используемую структуру (для проверки структуры объекта). Методы обработки данных определяются с помощью флагов, которые определяют, является ли данный объект Dict, List или DataList.

Для удаления или создания объекта, а также изменения полей с данными используется ссылка на область словаря с данными, отвечающего за изменяемый объект.

Поиск данных в словаре осуществляется с помощью ключа 'key', а для отображения данных используется атрибут 'name' (не совпадает с атрибутом key только у объекта List).

Для типа объекта используются две формы:

• сокращенная — только название структуры,

• полная — как в хранилище данных.

Сокращенная форма используется для отображения пользователям и поиска структур в списке. Полная форма необходима для сравнения типа полей структуры с типом объекта.

List можно условно разделить на ParentList (содержит список объектов Dict и DataList) и ChildrenList — Dict или DataList. ParentList (рис. 9), в отличие от Dict, может содержать только дочерние элементы. В качестве названия для ChildrenList (так как у данного элемента есть только номер в списке) используется поле с именем 'name', если его нет, выводится 'Name not found'.

3.4. Целостность структуры данных. Для обеспечения целостности структуры данных необходимо выполнять определенные действия при работе с объектной моделью:

1) рекурсивно удалять все ссылки на удаляемые объекты и их дочерние элементы;

2) при удалении объекта удалять соответствующие элементы из хранилища данных;

3) при удалении объекта из списка переопределять ключи для остальных элементов.

При удалении структуры без объектов достаточно удалить объект из модели и словаря со структурами. Этого достаточно, так как невозможно удалить структуру с объектами (т. е. структуру, на которую ссылается другой объект).

Так как все структуры хранятся в списке, потребуется переопределить ключи (атрибуты key) для всех структур. В противном случае произойдет смещение структур после удаленной.

Удаление дублирующей структуры отличается только тем, что все дочерние объекты ссылаются на первую дублирующую структуру. Поэтому при ее удалении достаточно передать все ссылки на объекты и родительские структуры следующей структуре в списке.

Для удаления объекта необходимо удалить ссылку на объект у родительского элемента и структуры, а также информацию об объекте из хранилища данных. Кроме того, необходимо рекурсивно удалить ссылки на дочерние объекты у всех структур.

4. ОБРАБОТКА МОДЕЛИ ДАННЫХ

Модель, получившаяся в результате обработки конфигурационного файла, является основой для работы с конфигурацией. Она позволяет выполнять такие операции, как отображение списка устройств, объединение файлов.

Основным недостатком является то, что при добавлении новой функции требуется изменять все уровни модели.

4.1. Обработка устройств. Кроме отображения объектов конфигурации, требуется отдельно отображать список всех устройств в файле.

При выполнении данной задачи учитывалось, что все устройства являются объектами конфигурации, у которых могут быть дочерние элементы. В результате список устройств отображается в виде дерева.

Кроме того, устройства являются объектами, схожими по составу с объектами конфигурации и копирующими их данные. Также потребовалось добавить взаимодействие между данными группами объектов, что в значительной степени повлияло на структуру модели обработки данных рис. 11.

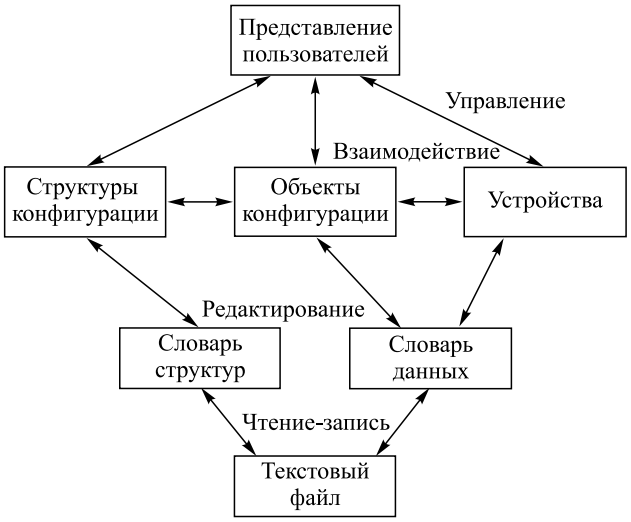

Рис. 11. Уровни конфигурации

Здесь структура будет рассматриваться как ориентированный граф, так как при поиске устройств идет тесное взаимодействие структурной и объектной моделей.

Ключевой особенностью модели конфигурации является возможность получить все необходимые данные при наличии правил хранения информации.

Информация об устройствах содержится в объекте корневого элемента SupportDevices, который содержит названия объектов со списками устройств в своем параметре list. Для поиска всех устройств в конфигурации необходимо найти все объекты SupportDevices.

Объекты SupportDevices основаны на структуре SSupportDevice. Данная структура содержит список со всеми объектами, ссылающимися на нее

(п. 3.2). При этом следует учитывать, что часть объектов представлена списками (SupportDevices) с интересующими нас объектами, поэтому их не нужно учитывать. Нас интересуют объекты, хранящие названия списков устройств в объекте (SupportList).

После определения названия списка через родительский элемент (ParentItem) проводится поиск элементов с указанными именами (DeviceList).

В результате, объединив списки найденных элементов, можно получить список устройств.

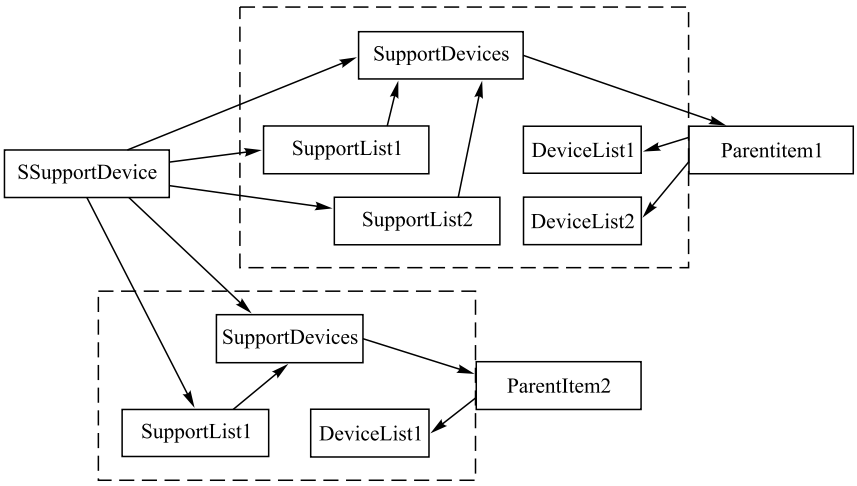

Рис. 12. Поиск списков с устройствами

4.2. Объединение файлов. Каждый модуль Sonix+ при запуске с определенными параметрами создает свой собственный модуль с конфигурацией, необходимой для себя, или изменяет уже существующий файл.

Для создания конфигурационного файла «с нуля» необходимы средства для объединения уже существующих файлов. Полезной также может оказаться возможность импортирования данных из другого конфигурационного файла.

При объединении файлов важную роль играет гибкость средств для выполнения целей. Обеспечить данную гибкость можно с помощью трех функций, это:

• Union — добавляет в открытый файл корневые элементы из указанного файла, при их совпадении сохраняются элементы из открытого файла (данную функцию удобно использовать, когда нужно добавить элементы конфигурации из другого файла, сохранив параметры уже существующих);

• Add from file — добавление/замена выбранных элементов из указанного файла, при совпадении элементов они заменяются (можно использовать,

когда необходимо добавить или заменить несколько корневых элементов из указанного файла);

• Open folder — объединение выбранных файлов в указанной папке (выполняется поочередный Union для всех выбранных файлов, нет возможности указать последовательность объединения файлов, данную функцию можно использовать, когда необходимо объединить несколько файлов, созданных Sonix+).

ЗАКЛЮЧЕНИЕ

С помощью РБНФ формализовано описание конфигурационного файла, отвечающего за настройку параметров модулей и устройств программного комплекса Sonix+.

Описана и реализована модель, позволяющая редактировать конфигурационный файл. В описании модели есть стандартные алгоритмы, поддерживаемые редактором конфигурации ConfigEditor:

1) отображение структуры данных в виде дерева;

2) добавление, удаление и копирование определенных частей структуры конфигурации;

3) изменение данных с проверкой типов данных;

4) объединение файлов.

Кроме того, в модель добавлены дополнительные алгоритмы, обеспечивающие такие функции, как:

1) поиск и удаление повторяющихся и неиспользуемых метаданных;

2) поиск и отображение списка устройств.

ЛИТЕРАТУРА

1. [Sonix+] http://sonix.jinr.ru/wiki/doku.php
2. [ConfigEditor] http://sonix.jinr.ru/wiki/doku.php?id=ru:config_editor
3. [Python] http://www.python.org/
4. [Serialization] http://en.wikipedia.org/wiki/Serialization
5. [Использование метаданных в программах на языке C++] http://www.rsdn.ru/article/cpp/cpp_metadata.xml
6. [ISO/IEC 14977] http://www.cl.cam.ac.uk/%7Emgk25/iso-14977.pdf
7. [ISO/IEC 14977 (неоф. перев.)] https://groups.google.com/forum/?hl=ru#!forum/localizations
8. [JSON] http://ru.wikipedia.org/wiki/JSON
9. [Python json] http://docs.python.org/2/library/json.html
10. [Bison] http://www.gnu.org/software/bison/
11. [PyQt] http://www.riverbankcomputing.com/software/pyqt/intro

Получено 7 июля 2014 г.

Редактор *М. И. Зарубина*

Подписано в печать 24.10.2014.
Формат 60 × 90/16. Бумага офсетная. Печать офсетная.
Усл. печ. л. 1,2. Уч.-изд. л. 1,4. Тираж 225 экз. Заказ № 58361.

Издательский отдел Объединенного института ядерных исследований
141980, г. Дубна, Московская обл., ул. Жолио-Кюри, 6.
E-mail: publish@jinr.ru
www.jinr.ru/publish/