Коляденко Юлия Юрьевна, *доктор технических наук, профессор, кафедра телекоммуникационных систем, Харьковский национальный институт радиоэлектроники, Украина.*
Белоусова Екатерина Эдуардовна, *аспирант, кафедра телекоммуникационных систем, Харьковский национальный институт радиоэлектроники, Украина,* ***e-mail: katrinmj@mail.ru****.*

Коляденко Юлія Юріївна, *доктор технічних наук, професор, кафедра телекомунікаційних систем, Харківський національний інститут радіоелектроніки, Україна.*

Білоусова Катерина Едуардівна, *аспірант, кафедра телекомунікаційних систем, Харківський національний інститут радіоелектроніки, Україна.*

Kolyadenko Yulia, *Kharkiv National University of Radio Electronics, Ukraine.*
Belousova Katerina*, Kharkiv National University of Radio Electronics, Ukraine,* ***e-mail: katrinmj@mail.ru***

УДК 004.75
DOI: 10.15587/2312-8372.2016.66441

Грудзинський Ю. Є.,
Марков Р. В.

ВИБІР ПРОТОКОЛУ СЕРІАЛІЗАЦІЇ ПРИ РОЗРОБЦІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В даній статті розглянуто сучасні протоколи серіалізації даних XML, JSON, упакування в бінарний вигляд, Protobuf та представлення даних у вигляді рядків. Проведено аналіз даних способів серіалізації даних для подальшого використання в розробці програмного забезпечення. Описано основні переваги та недоліки вище вказаних протоколів серіалізації. Зроблено висновки про доцільність використання кожного з них.

Ключові слова: *протокол, XML, JSON, Protobuf, серіалізація, парсінг, пакування, бінарний.*

1. Вступ

На сьогоднішній день доволі актуальною задачею при створенні програмних засобів сучасних автоматизованих систем є задача забезпечення перетворення різноманітних внутрішніх об'єктів системи в ефективний, наглядний та надійний вид, що забезпечить швидкий обмін даними між елементами системи, які знаходяться у різних місцях.

Серіалізація — це процес переведення будь-якої структури даних в послідовність бітів. Зворотною до операції серіалізації є операція десеріалізації (структуризації) — відновлення початкового стану структури даних з бітової послідовності. Ця послідовність може бути як бінарним представленням цих даних, так і текстовим [1]. Для серіалізації даних обміну може використовуватись декілька протоколів. У даній роботі проведено аналіз різноманітних протоколів серіалізації даних та зроблено висновки про доцільність їх використання у різних сферах програмної індустрії.

2. Аналіз літературних джерел і постановка проблеми

На сьогодні таких протоколів безліч, але найбільш популярними із них є XML, JSON, Protobuf, представлення даних у вигляді рядків та бінарне перетворення [1]. Кожен із розробників відповідного протоколу говорить лише про переваги та не дає ніякої інформації на рахунок недоліків свого протоколу у порівнянні із іншими [2–7]. Тому на практиці, при розробці будь-якого програмного забезпечення, постає проблема із вибором конкретного протоколу, що буде використовуватись при написанні програмного забезпечення.

Багато статей описують переваги та недоліки деяких популярних протоколів серіалізації даних при їх порівнянні з іншими, але зазвичай ці порівняння проводяться при розробці конкретного програмного забезпечення, для якого було заздалегідь вибрано той чи інший протокол для порівняння [8, 9], що не є коректним.

Вирішити проблему вибору протоколу серіалізації для розробки довільного програмного забезпечення можна лише після аналізу основних переваг та недоліків існуючих протоколів та визначення рекомендованих сфер їх застосування, після чого у програміста значно зменшиться кількість питань по вибору протоколу для конкретного випадку.

У роботах іноземних авторів зроблена спроба виконати такі порівняння, але тільки між окремими протоколами [10], чи з менш поширеними у нас протоколами BSON, XML with .NET [11]. Дана ж робота порівнює найбільш поширені на вітчизняних теренах протоколи серіалізації, з якими стикається найбільша кількість вітчизняних розробників програмових засобів.

3. Об'єкт, мета та задачі дослідження

Об'єктом дослідження цієї статті є протоколи серіалізації даних для переведення довільної структури інформаційних об'єктів програмного забезпечення у послідовність бітів, призначених для подальшого зберігання інформації та обміну нею між самими об'єктами автоматизованої системи, чи по каналах зв'язку.

Проведені дослідження ставили за *мету* визначити особливості існуючих протоколів серіалізації, їх переваги та недоліки, а також можливі області застосування програмного забезпечення, написаного з використанням даних протоколів.

Для досягнення поставленої мети вирішувались наступні задачі:

— визначити існуючі на сьогодні популярні протоколи серіалізації даних;

— встановити основні переваги та недоліки зазначених протоколів;

— визначити можливі області застосування програмного забезпечення, написаного з використанням даних протоколів.

4. Результати досліджень протоколів серіалізації для розробки програмного забезпечення

Коли перед програмістом постає завдання серіалізації даних, то у нього є кілька варіантів, що залежать від призначення майбутнього програмного забезпечення.

У випадку, якщо програмне забезпечення буде оперувати мінімальним об'ємом даних та є потреба у тому, щоб серіалізовані дані могли бути прочитані людиною, то найпростіший і досить популярний спосіб — це представити всі дані у вигляді рядків. У цьому випадку на виході автори статті отримують потік ASCII-символів, який потім буде передано по мережі або записано у файл [1]. Як можна помітити із вище наведеного основним недоліком даного підходу є те, що представлення структур у вигляді рядка підійде тільки для дуже простих наборів даних. До того ж доведеться писати досить багато коду для кодування і декодування. Тому даний підхід можна рекомендувати для програмного забезпечення, що дозволяє обмінюватись такими простими даними, як текстові рядки, числові вирази та їх комбінації.

Із збільшенням об'єму даних та варіації типів, з якими буде працювати програмне забезпечення, постає нагальна потреба у відносно простому способі створення та обробки серіалізованих даних, що окрім того дозволить створювати структуровану інформацію. Все це може забезпечити інший популярний метод — запис даних в XML форматі. XML-документ являє собою звичайний текстовий файл, в якому за допомогою спеціальних маркерів створюються елементи даних, послідовність і вкладеність яких визначає структуру документа і його зміст [2].

Різноманітних бібліотек, що займаються парсингом XML-файлів, можна нарахувати безліч, що спрощує процес написання механізмів серіалізації [3]. Дані в цьому випадку представлені наочніше, ніж у випадку простих ASCII рядків, та й гнучкість тут на висоті. Багато популярних протоколів використовують цю мову розмітки в якості своєї основи, так як вона розширювана і дозволяє не сильно замислюватися про зворотну сумісність при оновленні структури даних. Але, як і у випадку з попереднім способом, читання і запис даних в XML-формат накладає великі обмеження на продуктивність. Навігація по дереву вимагає високої потужності процесора, та й код, все-таки, теж доведеться трохи дописати, щоб все працювало так, як задумано. Ще один мінус, який необхідно врахувати — це розмір одержуваних даних. При досить великих обсягах інформації або поганих мережевих з'єднаннях використовувати XML не надто доцільно.

Із вище сказаного можна зробити висновок, що даний підхід дозволить оперувати значно більшим об'ємом даних, у порівнянні із перетворенням в рядок, але все-таки рекомендується для розробки програмного забезпечення, що не вимагає високої продуктивності та не передбачає обміну великими масивами даних.

Отже, розробникам все-таки необхідний протокол серіалізації даних, що буде мати простий формат та забезпечувати роботу із достатнім об'ємом даних, що в свою чергу не буде впливати на продуктивність. В якості такого можна розглянути сучасний протокол серіалізації даних, що має назву JSON. JavaScript Object Notation — простий формат обміну даними, зручний для читання і написання як людиною, так і комп'ютером. Він заснований на підмножині мови програмування JavaScript, визначеної в стандарті ECMA-262 3rd Edition — December 1999. JSON — текстовий формат, повністю незалежний від мови реалізації, але він використовує угоди, знайомі програмістам С-подібних мов, таких як С, С++, С#, Java, JavaScript, Perl, Python і багатьох інших [4]. За допомогою простого синтаксису стає можливим легко зберігати як прості числа і рядки, так і масиви, об'єкти, використовуючи при цьому не що інше як текст. Так само можна пов'язувати об'єкти і масиви, що дозволяє створювати складні структури даних [4].

JSON має ряд переваг:

— компактніший ніж XML;

— зрозумілий для людей і легко зчитується комп'ютером;

— його легко можна перетворити в програмні формати: числові значення, рядки, булевий формат, масиви і асоціативні масиви;

— майже всі програмні мови мають функції, що дозволяють зчитувати і створювати json формат даних.

JSON складається із таких основних частин:

— колекція пар ключ/значення. У різних мовах ця концепція реалізована як об'єкт, запис, структура, словник, хеш, іменований список або асоціативний масив;

— упорядкований список значень. У більшості мов це реалізовано як масив, вектор, список або послідовність.

Це універсальні структури даних. Майже всі сучасні мови програмування підтримують їх в будь-якій формі. Логічно припустити, що формат даних, незалежний від мови програмування, повинен бути заснований на цих структурах.

Отже, протокол JSON може бути використаний для розробки програмного забезпечення, що працює із досить великим об'ємом даних. В ньому майже відсутні обмеження для програміста по вибору мови програмування, на якій буде закодовано алгоритм. Окрім того продуктивність такого забезпечення (в плані швидкості серіалізації даних та їх обміну) буде досить високою.

Слід зазначити, що однією з основних вимог до протоколу серіалізації, який може використовуватися для розробки сучасного програмного забезпечення великих інформаційних систем є наявність в ньому високорівневих засобів та інструментів для роботи. Окрім того кінцевий розмір серіалізованих даних повинен бути мінімальний, оскільки робота сучасного програмного забезпечення пов'язана з великою інформаційною завантаженістю мережі. Велике значення також має цілісність та захищеність даних.

Вище наведені вимоги може виконати протокол Protobuf від Google. Protocol Buffers — це спеціальний метод кодування структур даних, який дозволяє швидко

і без проблем серіалізувати все що завгодно. Protocol Buffers має офіційну підтримку від Google для таких мов, як C++, Java і Python. Ця підтримка виражається в наявності компілятора для спеціальної мови, що описує структури даних. Структура даних визначається заголовними файлами *.proto. Компілятор генерує з них об'єкти на різних мовах програмування, що забезпечує властивість кросплатформенності [5–8, 11, 12].

Такий підхід, з одного боку, дозволяє замінити будь-який компонент системи на його реалізацію на іншій мові, а з іншого, дозволяє зручно підтримувати різні версії об'єктів, додавати нові поля і змінювати існуючі. Google розробив Protobuf для використання в своїх внутрішніх службах. Це бінарний формат кодування, який дозволяє задати схему для даних, що серіалізуються, з використанням спеціальних специфікацій.

Protobuf дозволяє описувати прості структури даних на спеціальній мові, яка потім компілюється в класи, що представляють ці структури. Разом з класами йде оптимізований код їх серіалізації в компактний формат представлення. А найкраще те, що доступ до даних максимально спрощено, оскільки для доступу до кожного поля у класі є відповідні методи «get» і «set», а для серіалізації об'єкта в масив байтів або потік введення/виведення потрібно зробити всього один виклик функції [10, 11, 13, 14].

Protocol Buffers дозволяє перевести дані і об'єкти в програмі в бінарний вид і безпечно переслати їх по мережі або зберегти у файл для подальшого використання. Основною перевагою Protocol Buffers є простота використання. Досить зробити метаопис об'єкта, після чого виходить готовий код об'єкта і його серіалізація. Користувачеві потрібно просто використовувати отриманий код. Стандартні засоби Protocol Buffers дають можливість не тільки виконувати плоску (послідовну) серіалізацію, а й серіалізацію в потік, що дозволяє відразу вивести дані в файл, мережу або кудись ще і дозволяє підтримувати нормальні C++ потоки типу std::ostream і std::istream.

5. Обговорення результатів дослідження протоколів серіалізації для розробки програмного забезпечення

Результати проведених досліджень властивостей використання окремих протоколів серіалізації для наочності зведено до табл. 1.

Таблиця 1

Переваги та недоліки використання різних протоколів серіалізації

Критерій/Протокол	Бінарний вигляд	Представлення у вигляді рядків	XML	JSON	Protobuf
Найменший розмір серіалізованих даних	так	ні	ні	частково	так
Читабельність серіалізованих даних	ні	так	так	так	ні
Необхідність написання додаткового коду	так	так	так	так	ні
Передача будь-яких типів даних	ні	ні	так	так	так
Використання додаткових бібліотек	ні	ні	так	так	так
Компактність коду	ні	ні	ні	так	так

Закінчення табл. 1

Критерій/Протокол	Бінарний вигляд	Представлення у вигляді рядків	XML	JSON	Protobuf
Гнучкість коду	ні	ні	так	так	так
Висока продуктивність коду	так	ні	ні	частково	так
Можливість роботи із великим об'ємом даних	ні	ні	ні	частково	так
Кросплатформеність	ні	ні	так	так	так
Наявність високорівневих засобів для роботи	ні	ні	частково	частково	так
Захищеність та цілісність серіалізованих даних	ні	ні	ні	ні	так
Простота використання	ні	ні	ні	частково	так

Дослідження, що були проведені авторами, у такому вигляді зроблено вперше і їх результати були використані при розробці програмного забезпечення комунікаційного модуля нової вітчизняної SCADA-системи. У якості протоколу серіалізації було вибрано протокол Protobuf. В подальшому планується дослідити структуру та особливості використання даного протоколу для реалізації ефективного паралельного обміну інформацією між SCADA-системою та ПЛК об'єктів автоматизації, а також для організації міжсистемного обміну.

6. Висновки

У результаті проведених досліджень було зроблено наступні висновки, щодо можливостей застосування вищезазначених протоколів для розробки програмного забезпечення:

— у випадку, якщо програмне забезпечення буде оперувати мінімальним об'ємом даних, такими, як текстові рядки, числа та їх комбінації, буде невеликим за розміром, не буде змінювати структури даних у майбутньому та є потреба у тому, щоб серіалізовані дані могли бути прочитані людиною, то найпростіший і раціональний спосіб — це плоска серіалізація у вигляді рядків. При відсутності потреби у контролі серіалізованих даних людиною — цілком підійде і бінарна серіалізація;

— при необхідності серіалізації складних типів об'єктів, таких як масиви, списки, дерева та інше і системи автоматизації, що не вимагає високої продуктивності (бухгалтерські програми, MES та подібні їм) цілком підійде серіалізація за допомогою протоколу XML. Серіалізація за допомогою XML також забезпечить максимальну сумісність з існуючими системами;

— при необхідності серіалізації складних типів об'єктів в автоматизованих системах з підвищеними вимогами до продуктивності та при відсутності якихось особливих вимог до безпеки передачі даних, вимогами до кросплатформенності — оптимальним варіантом стане використання протоколу JSON. В ньому майже відсутні обмеження для програміста по вибору мови програмування, на якій буде закодовано алгоритм. Окрім того, продуктивність такого рішення (в плані швидкості серіалізації даних та їх обміну) буде досить високою;

— у випадку серіалізації складних типів об'єктів в автоматизованих системах з особливими вимогами

до продуктивності (SCADA-системи), наявними вимогами кросплатформенності та підвищенними вимогами до безпеки передачі даних (банківські системи, OLAP, ERM/CRM) кращим вибором стане серіалізація за допомогою протокола Protobuf, в якому (на відміну від інших) також присутні високорівневі засоби та інструменти для роботи. Окрім того, кінцевий розмір серіалізованих даних буде мінімальний, а також гарантується цілісність та захищеність даних.

Література

1. PROTOBUF VS. BOOST: SERIALIZATION [Электронный ресурс] // Журнал «Хакер». — 31.10.2013. — Режим доступа: \www/URL: https://xakep.ru/2013/10/31/protobuf-vs-boost-serialization/
2. Еллоіт, Р. XML [Текст] / Р. Еллоіт. — Спб.: Символ-Плюс, 2001. — 576 с.
3. Введение в JSON [Электронный ресурс]. — 11.11.2011. — Режим доступа: \www/URL: http://json.org/json-ru.html
4. Json или «Туда и Обратно» [Электронный ресурс] // Хабрахабр. — 01.08.2014. — Режим доступа: \www/URL: https://habrahabr.ru/company/naumen/blog/228279/
5. Protocol Buffers [Electronic resource] // Google Developers. — Available at: \www/URL: https://developers.google.com/protocol-buffers/. — 05.01.2016.
6. Protocol Buffer Basics: C++ [Electronic resource] // Google Developers. — 03.09.2014. — Available at: \www/URL: https://developers.google.com/protocol-buffers/docs/cpptutorial#why-use-protocol-buffers. — 05.01.2016.
7. Google Protocol Buffers in action (C++) [Electronic resource]. — 16.09.2012. — Available at: \www/URL: http://forums.4fips.com/viewtopic.php?f=3&t=807
8. JSON и XML. Что лучше? [Электронный ресурс] // Хабрахабр. —24.08.2007. — Режим доступа: \www/URL: https://habrahabr.ru/post/31225/
9. 5 Reasons to Use Protocol Buffers Instead of JSON For Your Next Service [Electronic resource] // Code Climate. — 05.06.2014. — Available at: \www/URL: http://blog.codeclimate.com/blog/2014/06/05/choose-protocol-buffers/
10. ProtoBuf.js vs JSON [Electronic resource] // GitHub, Inc. — 02.02.2015. — Available at: \www/URL:https://github.com/dcodeIO/protobuf.js/wiki/ProtoBuf.js-vs-JSON
11. COMPARING PROTOBUF, JSON, BSON, XML WITH .NET FOR FILE STREAMS [Electronic resource] // Software Engineering. — 09.01.2014. — Available at: \www/URL: http://damienbod.com/2014/01/09/comparing-protobuf-json-bson-xml-with-net-for-file-streams/
12. Если вы еще используете JSON, то Google protobuf идет к вам! [Электронный ресурс] // Дневник программиста. — 27.11.2012. — Режим доступа: \www/URL: http://knzsoft.blogspot.com/2012/11/protobuf.html
13. Сопоставление JSON и XML [Электронный ресурс] // Microsoft. — Режим доступа: \www/URL: https://msdn.microsoft.com/ru-ru/library/bb924435(v=vs.110).aspx. — 01.02.2016.
14. Как сериализовать и десериализовать данные JSON [Электронный ресурс] // Microsoft. — Режим доступа: \www/URL: https://msdn.microsoft.com/ru-ru/library/bb412179(v=vs.110).aspx. — 01.02.2016.

ВЫБОР ПРОТОКОЛА СЕРИАЛИЗАЦИИ ПРИ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В данной статье рассмотрены современные протоколы сериализации данных XML, JSON, упаковка в бинарный вид, Protobuf и представление данных в виде строк. Проведен анализ данных способов сериализации данных для дальнейшего использования в разработке программного обеспечения. Описаны основные преимущества и недостатки вышеуказанных протоколов сериализации. Сделаны выводы по целесообразности использования каждого из них.

Ключевые слова: протокол, XML, JSON, Protobuf, сериализация, парсинг, упаковка, бинарный.

Грудзинський Юліан Євгенович, *старший викладач, кафедра автоматизації теплоенергетичних процесів, Національний технічний університет України «Київський політехнічний інститут», Україна,* ***e-mail: jug@sonettele.com****.*

Марков Роман Валерійович, *кафедра автоматизації теплоенергетичних процесів, Національний технічний університет України «Київський політехнічний інститут», Україна.*

Грудзинский Юлиан Евгеньевич, *старший преподаватель, кафедра автоматизации теплоэнергетических процессов, Национальный технический университет Украины «Киевский политехнический институт», Украина.*

Марков Роман Валерьевич, *кафедра автоматизации теплоэнергетических процессов, Национальный технический университет Украины «Киевский политехнический институт», Украина.*

Grudzynskyy Yulian, *National Technical University of Ukraine «Kyiv Polytechnic Institute», Ukraine,* ***e-mail: jug@sonettele.com****.*

Markov Roman, *National Technical University of Ukraine «Kyiv Polytechnic Institute», Ukraine*

УДК 004.021+004.052.42
DOI: 10.15587/2312-8372.2016.66444

Дубинский А. Г.,
Хорольский О. А.

РАЗРАБОТКА МЕТОДА ПРОВЕРКИ БЛОК-СХЕМ МЕДИЦИНСКИХ АЛГОРИТМОВ

Обеспечение высокого качества медицинской помощи требует точной и эффективной записи медицинских алгоритмов. Предложен метод для выявления частей схем алгоритмов, которые записаны не в соответствии с государственным стандартом, что допускает их неоднозначное толкование и может стать причиной врачебных ошибок. Дан алгоритм выявления таких частей, показан пример его применения для обнаружения ошибок.

Ключевые слова: *алгоритм, стандарт, блок-схема, клиническое руководство, медицинский стандарт, клинический протокол.*

1. Введение

Под руководством Министерства здравоохранения Украины уже несколько десятилетий ведется работа по созданию полной базы медико-технологических документов по стандартизации медицинской помощи.

Новая система представлена тремя типами документов: унифицированные клинические протоколы медицинской