УДК 004.71

ОРЛОВА М.М.,
ШАНДИБА Д.В.

РЕАЛІЗАЦІЯ МЕРЕЖЕВИХ ПРОТОКОЛІВ ЯК НЕЗАЛЕЖНИХ ПРОЦЕСІВ

В статье рассматривается подход к реализации сетевых протоколов как независимых процессов. Использование предлагаемого метода позволяет реализовывать логику каждого сетевого протокола в стеке полностью независимо от других протоколов, что значительно облегчает применение формальных методов для решения задачи синтеза протоколов. В результате исследования сформулирована и опробована методика формального описания обобщенных моделей поведения сетевых протоколов, а также внешних механизмов их взаимодействия.

The article reviews the approach to network protocols implementation as independent processes. Usage of the proposed method allows implementation of each network protocol in the stack completely independently from other protocols, that significantly simplifies the task of formal methods application for protocols synthesis. As a result of the research there was proposed and practically tried out a methodology of formal specification of generalized models of network protocols behavior, as well as of external mechanism of their interaction.

Вступ

Формальні методи представлення мережевих протоколів є важливим напрямом розвитку автоматизації розробки програмних технологій створення та підтримки ефективних мережевих систем. Модульне представлення поведінкової логіки мережевих протоколів є однією з підзадач розв'язання якої наближає нас до вирішення загальної проблеми формального подання мережевих протоколів. Використання модульності (зокрема у вигляді загальних абстракцій, таких як кінцеві автомати) в поведінковій логіці протоколів на практиці дозволяє перенести значний об'єм описів поведінкової логіки мережевих протоколів у формалізовані форми, які, в свою чергу, надають можливості автоматизації проектування, реалізації, верифікації та контролю за роботою мережевих протоколів. Пошуки вирішення кожної із цих задач займають важливе місце в сфері мережевого програмування.

Існуючі дослідження і публікації

Загалом дослідження у сфері формального представлення мережевих протоколів виконуються досить давно і набули свого розвитку у трьох головних напрямках, загальний огляд яких наведено у [1]:

1. використання формальних методів із метою аналізу характеристик протоколів ще до їх реалізації;
2. використання формальних методів із метою контролю за роботою вже реалізованих протоколів, вирішення задач мережевої безпеки;
3. використання формальних методів із метою синтезу (реалізації) протоколів.

Розробки останньої групи, в свою чергу, виконуються двома основними шляхами: шляхом створення нових мов програмування вузького застосування (domain-specific languages), із, відповідно, розробкою специфікацій цих мов та повної інфраструктури їх підтримки [8, 9, 10], а також, рідше, шляхом модифікації (розширення) існуючих мов програмування [2]. В обох випадках практичне застосування отриманих реалізацій потребує наявності додаткових інструментів розробки (компіляторів або трансляторів), а часто також і додаткових дій, необхідних для інтеграції отриманих реалізацій у програмні продукти, що мають ці реалізації використовувати. Необхідність цього є суттєвим недоліком при практичному використанні таких методів: фактично формалізація процесу розробки протоколів вищезазначеними способами дає ресурсний виграш при реалізації правил протоколу, проте потребує додаткового часу та зусиль для реального використання цих можливостей. Тож проблема формалізації протоколів шляхом, привабливим з практичної точки зору, все ще потребує вирішення.

Окремою задачею є проблема формального представлення даних, якими оперують протоколи. На відміну від представлення поведінкової логіки, вона більше досліджена, і

результатами цих досліджень є численні мови опису даних та супутні технології, розглянуті в [9, 11].

Мета

Метою дослідження, викладеного в даній статті, є розробка моделі реалізації поведінкової логіки мережевих протоколів, яка б з одного боку могла бути безпосередньо виражена в термінах популярних сьогодні мов програмування виробничого маштабу (таких як C/C++, Java, .Net мови) без застосування додаткових інструментів, а з іншого – мала б характеристики, що дозволяли б її опис, формування (створення) та аналіз із застосуванням формальних методів.

Пропонований підхід

Традиційний підхід до реалізації поведінкової логіки мережевих протоколів (як із використанням звичайного програмування, так і формальних методів) полягає у використанні ієрархічності природи мережевих протоколів, тобто коли протоколи вищого рівня безпосередньо використовують можливості протоколів нижчого рівня. Це дозволяє уникнути дублювання коду, а також досягти вищих рівнів абстрагування при розробці наступних (вищих) рівнів протоколів. В даному випадку поняття “використовують”, як правило, означає безпосередні виклики функцій інтерфейсів протоколів нижчого рівня протоколами вищого рівня.

У більшості випадків така архітектура протоколів є повністю виправданою. Втім, вона істотно погіршує модульні властивості окремих протоколів у стеку, і цей факт набуває особливої гостроти при дослідженні способів формального представлення протоколів. В більшості існуючих підходів до формального представлення поведінкової частини протоколів безпосередній зв’язок між протоколами реалізується через інкапсульовані інтерфейси протоколів нижчого рівня у виклики підсистеми представлення протоколів. Тобто бібліотечний виклик заманюється на формальний літерал, який цей виклик закапсулює. В реалізаціях, основаних на умовно псевдо-незалежних моделях [10], процеси хоча і мають суттєві ознаки відокремленості, проте вони все одно тісно пов’язані між собою. Загальним негативним результатом використання таких підходів є “монолітність” кінцевих реалізацій. Зміна будь–якої поведінкової частини протоколу нижчого рівня потребує перегляду всієї реалізації на предмет виникнення можливих логічних помилок. У випадку існуючих способів формального представлення поведінки протоколів додається ще один суттєвий недолік: внесення можливостей протоколів нижчого рівня в інфраструктуру системи формального визначення протоколів фактично робить створювані системи асиметричними з точки зору маштабованості: інтерфейс створеного, нового протоколу не виглядає таким самим чином, як виглядає інтерфейс протоколу, внесеного в інфраструктуру системи. Це робить такі системи придатними для порівняно ефективного створення лише “фінальних” протоколів, на основі яких важко створювати протоколи вищих рівнів тим самим шляхом, яким було створено даний протокол. Наприклад, у [8] викладено успішну методику створення мережевих протоколів транспортного рівня, яка мало придатна для протоколів прикладного і вищого рівнів через відсутність інтерфейсної сумісності між отримуваними реалізаціями.

Ідеальною моделлю представлення поведінкової логіки мережевого протоколу є абсолютна незалежність рівнів протоколу. Це вирішувало б існуючі проблеми, проте така модель є або принципово недосяжною (при збереженні ідеї повторного використання коду), або недоцільною, внаслідок втрати переваги від повторного використання коду.

Для поєднання з одного боку переваг ідеальної моделі для легкості формалізації протоколів, а з іншого – існуючих практик реалізації протоколів запропоновано розглядати протоколи в одному стеку як окремі незалежні процеси, взаємодія між якими виконується шляхом використання відокремленої сутності, подібної до паттерну прикладного програмування “посередник” (mediator) [6]. Таким чином, реалізація мережевих протоколів стає подібною до реалізації розподілених систем із відповідною специфікою. Кожен з протоколів пропонується реалізовувати відповідно лише до правил власне цього протоколу, а взаємодію між протоколами організовувати завдяки окремому прошарку, винесеному в окрему абстракцію, тобто в певний модуль, який би впливав на поведінку обох протоколів, а більш точно –корегував

би її для кожного протоколу на основі сигналів (повідомлень або викликів), що генеруються кожним із цих протоколів (Рис. 1a). Для більш ніж двох протоколів підхід масштабується або шляхом застосування спільного керуючого модуля для всіх протоколів (Рис. 1b), або шляхом використанням декількох окремих модулів (Рис. 1c).

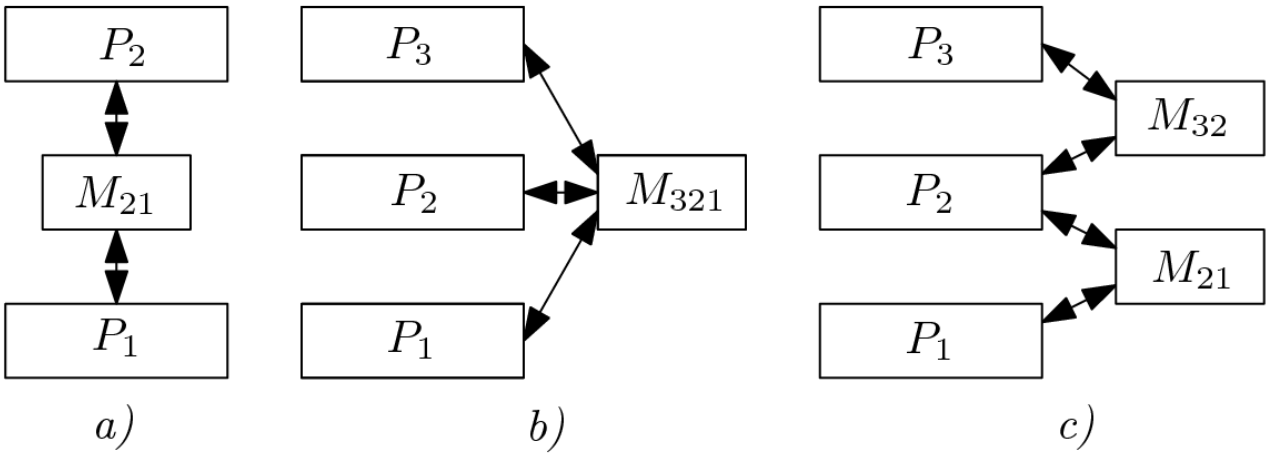

Рис. 1. Використання посередників для зв'язку між протоколами

Для реалізації запропонованого підходу необхідно виконувати ряд вимог:

1. Кожен протокол має повністю задекларувати (описати) свої публічні інтерфейсні стани, сигнали та дані, якими він оперує.
2. Будь-які внутрішні (непублічні) процеси та стани протоколу не можуть суперечити його публічному станові.
3. Кожен протокол має бути реалізований в термінах лише цього протоколу.

Завдяки цим правилам опис поведінки кожного протоколу в системі концентрується лише на власному стані протоколу, без залежності від станів інших протоколів, які реалізовані у стеку. Це дозволяє досягти того рівня абстрагування, за якого опис кожного окремого протоколу стає відокремленою задачею, і може бути виконаний довільним запропонованим формальним методом, так як семантично і синтаксично зникають залежності від протоколів інших рівнів, які часто стають на заваді створенню містких зручних формалізацій (специфікацій).

Розглянемо детальніше роль медіатору. Для простоти прикладу оперуватимемо медіатором для двох протоколів. Роль медіатору полягає в тому, щоб на основі інтерфейсних сигналів, що генеруються розроблюваними протоколами, реалізувати залежність станів та переходів протоколів вищого рівня від станів та переходів протоколів нижчого рівня. Фактично, медіатор реалізує ту частину поведінкової логіки протоколу, яка пов'язана із використанням цим протоколом можливостей іншого протоколу (протоколу нижчого рівня). Для рис. 1a, пара $\{P_2, M_{21}\}$ повністю виражає поведінкову логіку протоколу P_2, причому P_2 описує протокол "як такий", а M_{21} виражає його поведінковий зв'язок із протоколом P_1.

На практиці безпосередню реалізацію протоколів та медіатору можна повністю закапсулювати, визначивши наступні частини:

1. Формальний публічний інтерфейс протоколів ($I(P_n)$):
   - множина публічних сигналів, що генеруються кожним протоколом ($SG(P_n)$);
   - множина публічних сигналів, що приймаються кожним протоколом ($SR(P_n)$);
   - опис (перелік) публічних станів протоколів ($T(P_n)$);
   - опис публічних даних, якими оперують протоколи ($D(P_n)$).
2. Способи визначення логіки медіатору, в термінах інтерфейсів обох протоколів (процедурне, декларативне або процедурно–декларативне представлення) ($R(M_{nm})$).

Таким чином, схема взаємодії протоколів і медіатору набуває вигляду, представленого на рис. 2. Зауважимо, що кожен із протоколів взаємодіє лише з елементами, які визначені в ньому, і жодним чином не торкається логіки, станів, сигналів чи даних, описаних в іншому протоколі. Задачу цієї "інтеграції" виконує медіатор. Аналогічним чином виконується розміщення клієнтського коду, тобто коду, що в решті решт користується можливостями протоколу, що специфікується: адже

клієнтський код – це, за своєю суттю, той самий медіатор, який тільки поєднує між собою не 2 (або більше) різні рівні протоколів, а програмну бізнес–логіку із протоколом. Така уніфікованість дозволяє використовувати пропоновані засоби моделювання протоколів навіть за межами частин програмного забезпечення, що торкаються специфікації протоколів. А також значно полегшує заміну реалізацій протоколів в кінцевому продукті.

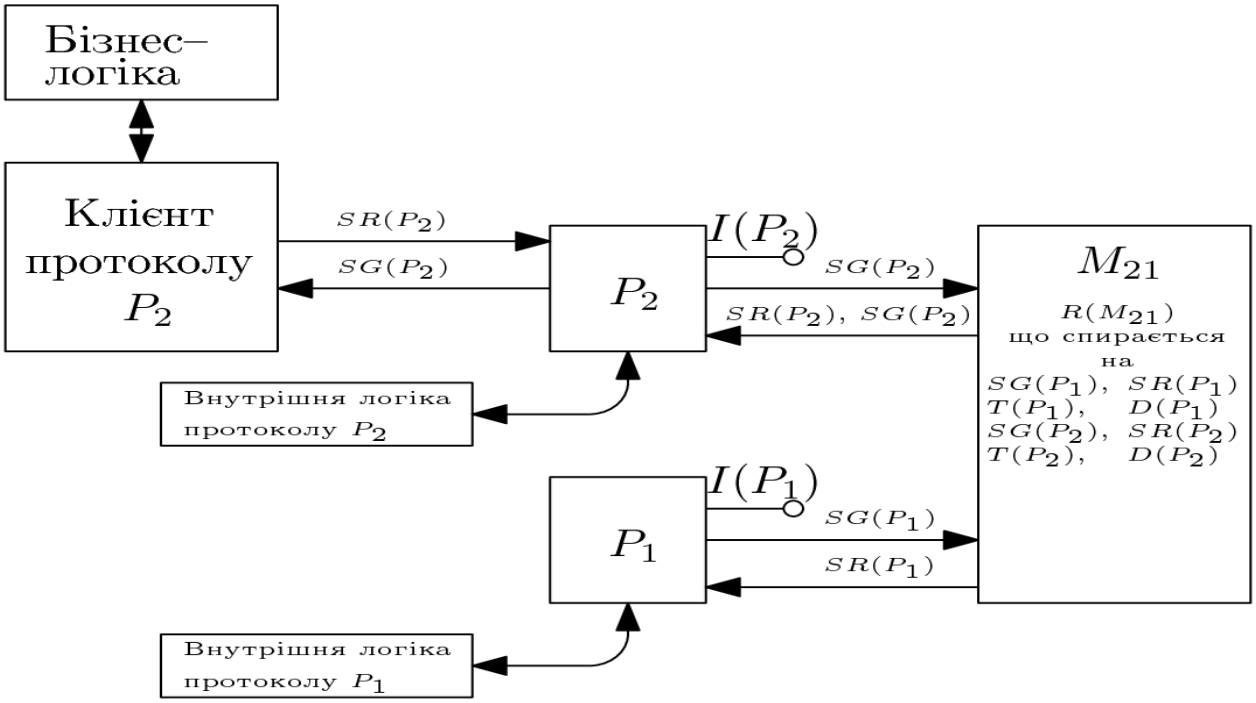

Рис. 2. Взаємодія елементів системи протоколів

В залежності від характеру протоколу може змінюватися об'єм та роль коду, що відповідає за внутрішню логіку протоколу. В залежності від обраного способу представлення протоколів може бути доцільно реалізовувати внутрішню логіку в термінах інтерфейсних станів та сигналів протоколу. За певних обставин внутрішня логіка протоколу може взагалі реалізуватися виключно в термінах публічного інтерфейсу протоколу.

Таким чином, за умови визначення цих частин, питання внутрішньої реалізації протоколу та медіатору теоретично не є принциповим. Втім, метою даного дослідження є знаходження зручних та функціональних формальних способів представлення поведінки протоколів. Тому пропонується звернутися до ряду відомих абстракцій, які б підійшли для представлення поведінковий моделі протоколу та медіатору. Такими абстракціями можуть бути кінцеві автомати. Перевага їх використання для вирішення даної задачі полягає в тому, що по–перше, їх властивості дуже добре досліджені в тому числі в сфері реалізації мережевих протоколів і розподілених систем [7], по–друге, кінцеві автомати різною мірою часто вже використовуються в існуючих текстових специфікаціях протоколів, а по–третє, вони дуже добре дозволяють формалізувати свої моделі в загальному вигляді.

Виходячи із цього, формальне представлення протоколу P_n зводиться до формального представлення кінцевого автомату $A(P_n)$, що описує даний протокол. А отже, формальне представлення повної поведінки протоколу P_2 в стеку над протоколом P_1 є послідовністю формального представлення автоматів $A(P_2)$ та $A(P_1)$, а також –опису взаємодії між ними $R(M_{21})$.

Питання формального представлення $R(M_{21})$ носить інший характер. Фактично, взаємодію автоматів P_2 та P_1 можна описати процесом перетворення вхідних кортежів $<t(P_2), t(P_1), [sg(P_2),] [sg(P_1),] [d(P_2),] [d(P_1)]>$ у кортежі $<[sg(P_2),] [sr(P_2),] [sr(P_1)]>$. Дані перетворення є ключовими у визначенні поведінкової логіки протоколів. Завдяки їм автомати, що відповідають за різні рівні протоколів, отримують можливість "співпрацювати": тобто виконувати власні внутрішні переходи на основі сигналів іншого автомату, генерувати сигнали у відповідь на сигнали іншого автомату, передавати дані від і до

іншого автомату у відповідності до описаних форматів даних.

В практичній площині застосування пропонованого підходу вимагає або наявності в системі вже готових базисних реалізацій протоколів, на які б могла спиратися розробка нових протоколів вищого рівня, або вимагатиме реалізацію базисного шару самим розробником. В тому чи іншому випадку в системі існуватиме один або більше протоколів, які з одного боку, будуть реалізовані в умовно вільній формі, а з іншого – матимуть інтерфейс у тому вигляді, у якому він може використовуватись запропонованим декоративним шляхом. До цього шару доцільно віднести протоколи, вже доступні користувачеві системи у традиційному вільному вигляді, реалізувавши "обгортки–адаптери" цих протоколів для використання у системах, створених за запропонованою методикою (Рис. 3).

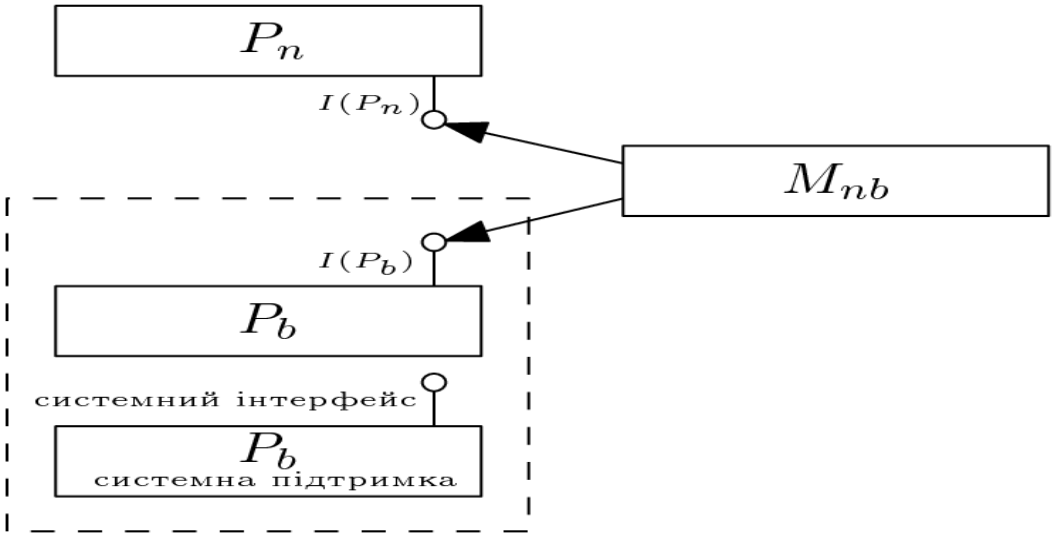

Рис. 3. Приклад взаємодії протоколу P_n із протоколом P_b, що належить базисному шарові

Отримані результати

Запропонована методика дозволила на практиці створити формальні описи і перетворити їх у подальші реалізації ряду простих мережевих протоколів, що знаходяться у стеку безпосередньо над протоколом TCP: клієнтську та серверну частину протоколу daytime, спрощений варіант протоколу HTTP, а також клієнтську та серверну частину двох нових протоколів, що знаходяться в стеку один над одним і в свою чергу базуються на протоколі UDP. В якості початкового текстуального опису протоколів для daytime та HTTP використовувались відповідні документи RFC: RFC–867 [4], RFC–2068 [5], для нових протоколів було створено стислі описи правил протоколів у вигляді списку. В якості представлення формального подання описів протоколів використовувались 2 підходи: вільна форма опису кінцевих автоматів із застосуванням таблиць опису поведінкової взаємодії протоколів із подальшим перетворення у програний код, а також безпосереднє представлення кінцевих автоматів та правил їх взаємодії із використанням можливостей мета–програмування мови C++. Окрім того, кожен із цих протоколів було розроблено класичним шляхом на мові C++ із використання BSD Sockets в бібліотеці Boost [3]. У порівняні із класичним підходом значно більший час було витрачено на створення "інфраструктури" TCP, що для способу формального подання протоколів означало опис і реалізацію кінцевого автомату TCP. Однак як тільки вона була створена один раз, більше не було потреби в її модифікації. Для класичного підходу для вирішення цієї задачі безпосередньо використовувались можливості бібліотеки TCP в бібліотеці Boost. Одночасний опис і створення автоматів протоколів із застосуванням можливостей декларативного програмування C++ і програмування логіки цих протоколів засобами класичного програмування зайняли приблизно однаковий час. Використання двоетапного підходу: спочатку опису, а потім його перетворення у реалізацію, зайняло

приблизно в півтори рази більше часу,. Найбільш цікаві результати були отримані при необхідності зміни логіки роботи протоколів (при навмисному внесенні змін в текстуальні описи протоколів): в цьому випадку зміна декларативних описів протоколів на C++ зайняла найменший час, зміна описів автоматів та їх реалізацій була знову приблизно в півтори рази довшою. Зміна класичного програмного коду зайняла найбільше часу – більше ніж у 2 рази довше у порівняння із декларативним C++ підходом.

Висновки

В результаті проведеного дослідження було, по–перше, доведено доцільність використання методик формального подання поведінкової логіки мережевих протоколів, а по–друге було запропоновано одну з таких методик, можливість використання якої було перевірено практично. Такою методикою було розглянуто спосіб подання протоколів у вигляді незалежних процесів, основаних на кінцевих автоматах із описом взаємодії між ними. Практичні результати показали, що найбільш ефективним з точки зору часових затрат на первинне створення і подальшу підтримку реалізацій протоколів є їх реалізація в декларативному вигляді на кінцевий мові програмування (використовувалась мова C++). Втім, як враховуючи, що далеко не всі популярні мови мають елементи декларативного або функціонального програмування, ефективним також можна вважати підхід, при якому описи елементів протоколу виконуються у певній формальній формі, після чого вони максимально автоматично перетворюються у кінцеві реалізації. Класичне програмування за дотримання високих стандартів створення коду дозволяє найбільш швидко розробляти первинні реалізації відносно нескладних протоколів. Проте при роботі із більш складними протоколами та при необхідності змінювати існуючі реалізації класичний підхід програє підходу формального визначення поведінкової логіки протоколів. Окрім цього, формальний підхід автоматично дозволяє виконувати моніторинг роботи протоколу на рівні не абстрактних об'єктів мови програмування, а реальних складових частин цього протоколу, а отже дозволяє пришвидшити пошук та локалізацію помилок в реалізації протоколу, оцінювати його продуктивність тощо.

Втім, важливим є розробка ефективного та зручного оточення для створення формальних описів (будь–яким способом: безпосереднім чим двоетапним). Таким чином, з великою обачністю має обиратися конкретний спосіб подання кінцевих автоматів: їх використання в будь–якому випадку збільшує архітектурну складність кінцевого коду, а отже впливає на продуктивність системи, що є однією з проблем запропонованої методики. Окрім того, отримані результати залишають відкритими ряд питань для подальшого дослідження і вдосконалення, такі як: вибір ефективної методики реалізації запропонованої моделі в термінах конкретного середовища розробки (окрім дослідженої мови C++); створення оточення та бібліотек для підтримки цих методів; питання оптимальності отриманих реалізацій: наскільки вони можуть бути наближені до реалізацій із використанням класичних підходів; питання трансформації правил протоколів в правила поведінки процесів, а також їх формального подання; питання представлення блоків даних.

Сукупність отриманих результатів і перспективи розв’язання існуючих проблем роблять дану тематику актуальною в сфері сьогоденного мережевого програмування.

Список посилань

1. Babich F. and Deotto L. Formal methods for specification and analysis of communication protocols // IEEE Communications Surveys & Tutorial. –2002. –Vol. 4, Q3. –P. 2–20.
2. Basu A., Morrisett G. and Von Eicken T. Promela++: a language for constructing correct and efficient protocols // INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. –1998. –Vol. 2. –P. 455–462.
3. C++ Бібліотека Boost [електронний ресурс]. – Режим доступу: http://www.boost.org. – Boost C++ Library.
4. Daytime protocol (Request For Comments 867) [електронний ресурс] / Postel J. – Режим доступу: http://www. rfc-editor.org/rfc/rfc867.txt — Request for Comments: 867.
5. Hypertext Transfer Protocol – HTTP/1.1 (Request For Comments 2068) [електронний ресурс] /

Fielding R. – Режим доступу: http://www. rfc-editor.org/rfc/rfc2068.txt – Request for Comments: 2068.

6. Gamma. E., Helm R., Johnson R., Vlassides J.M. Design Patterns: Elements of Reusable Object-Oriented Software / Eddison–Wesley Professional. –1994. –416 p.
7. Killian C.E., Anderson J.W., Braud R., Jhala R. and Vahdat A.M. Mace: language support for building distributed systems // Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation. –San Diego, California, USA, ACM, 2007. –P. 179–188.
8. Kohler E., Kaashoek M.F. and Montgomery D.R. A readable TCP in the Prolac protocol language // Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication. –Cambridge, Massachusetts, USA; ACM, 1999. –P. 3–13.
9. Madhavapeddy A., Ho A., Deegan T., Scott D. and Sohan R. Melange: creating a "functional" internet // Proceedings of the 2nd ACM SIGOPS / EuroSys European Conference on Computer Systems. – 2007.
10. Vuong S., Lau, C. and Chan R.I. Semiautomatic Implementation of Protocols Using an Estelle-C Compiler // Software Engineering, IEEE Transactions on. –1988. –Vol. 14. –P. 384–393.
11. Warth A. and Piumarta I. OMeta: an object-oriented language for pattern matching // Proceedings of the 2007 symposium on Dynamic languages. –Montreal, Quebec, Canada; ACM, 2007. –P. 11–19.

Поступила в редакцию 14.12.2009