УДК 681.3.51

ТИПІЗАЦІЯ ДАНИХ У СИСТЕМАХ КРИТИЧНОГО ПРИЗНАЧЕННЯ

Ю.С. Манжос
(подав д.т.н., проф. В.С.Харченко)

Розглядаються обмеження класичних систем типів даних та їх вплив на надійність програмного забезпечення комп'ютеризованих комплексів критичного призначення.

Вступ. Реальні потреби сучасної авіаційної та космічної техніки обмежують імовірність допустимого ризику величиною 0,000000001 за робочий інтервал часу, який для аеробусу дорівнює 10 годинам, а для виробів ракетно-космічної техніки (РКТ) вимірюється місяцями і навіть роками. Навіть після системного тестування програмне забезпечення (ПЗ) містить 1,5 ... 4 дефекти на 1000 операторів [1]. За умови існування ПЗ обсягом у десятки мільйонів інструкцій, що керує аерокосмічними виробами, ГЕС, АЕС тощо, стає зрозумілим тривога за наше майбутнє. Наприклад, причиною загибелі у вересні 1999 року міжпланетного апарата НАСА “Mars Climate Orbiter”, який занадто заглибився в атмосферу Марса під час проведення аерокосмічного гальмування, було використання при розробленні програмних модулів різних одиниць виміру довжини [2]. Сьогодні найефективнішою методологією забезпечення потрібного рівня якості ПЗ є багатоверсійність, що має суттєвий недолік - багаторазове підвищення витрат унаслідок паралельного розроблення кількох версій. Це потребує як ретельного аналізу можливостей існуючих технологій, так і пошуку нових принципів забезпечення якості ПЗ, найважливіші складові якого - надійність, супроводжуваність та безпека.

Мета статті - дослідження впливу типізації даних на складність та надійність ПЗ.

Методи забезпечення надійності програмного забезпечення. Згідно з класичною монографією [3], існуючі принципи та методи забезпечення надійності ПЗ спрямовані на *уникнення дефектів, знайдення дефектів, виправлення дефектів та забезпечення стійкості до дефектів*. Принципи та методи першої групи дозволяють мінімізувати або зовсім виключити дефекти, другої - базуються на функціях ПЗ, що допомагають виявляти дефекти, третьої - базуються на функціях ПЗ, призначених для виправлення дефектів та їх наслідків. Нарешті, принципи

та методи четвертої групи призначені для підвищення здатності продовження функціонування дефектного ПЗ.

Найбільш поширеними є методи перших двох груп. Принципи та методи першої групи спрямовані на недопущення помилок шляхом боротьби зі складністю ПЗ, зумовленою необхідністю відображення частини реального світу, об'єкта керування та цільової функції керування. Складність ПЗ набагато перевищує складність апаратури. Для будь-якого апаратного проекту існує значна частина повторно використовуваних компонентів і кінцева кількість рішень. У той же час кількість оптимальних рішень програмних проектів нескінченна, а частина повторно використовуваних елементів незначна. Проблема полягає в тому, що дуже часто ми закладаємо у проект зайву складність. Але надмірна складність програмних проектів призводить не лише до зайвих витрат при розробленні, верифікації, супроводженні, а й надає програмним дефектам більше шансів на розповсюдження. Ось чому найкращими вважаються найшвидші, найменші, найпростіші, найзрозуміліші та створені з найменшими зусиллями проекти [4]. Об'єктно-орієнтована (ОО) парадигма теоретично дозволяє зменшити складність програмного забезпечення та уникнути при проектуванні семантичного розриву з реальним світом, відобразити всю складність не лише об'єкта керування, а й частини реального світу. Основою ОО - парадигми є типізація даних. Як відзначено у [5], введення типів даних у програмуванні пов'язане з розумовою діяльністю людини, що потребує створення понять і роботи з ними.

Для полегшення цього сучасні мови програмування, крім основних (базових) типів даних, призначених для дій над різноманітними числами та рядками, мають також засоби для визначення складних, структурованих типів даних, за допомогою яких у програмах відображаються стани складних об'єктів або їх сукупностей. До них належать конструкції побудови перелічених типів, масивів, записів, списків тощо.

Типізація даних дозволяє писати і верифікувати ПЗ у загальних поняттях (типах та операціях над ними) конкретної предметної області, не ускладнюючи подробицями реалізацій типів даних. Обмеження базовими типами даних недопустиме, оскільки може призвести до ситуації, коли програмні об'єкти належать до одного базового типу (наприклад, реальних чисел), але до різних категорій (наприклад, час і довжина).

Вплив типізації даних. Щоб зрозуміти важливість типізації даних для забезпечення надійності програмного забезпечення, розглянемо класи програмних дефектів.

1. Використання неправильного ідентифікатора об'єкта.
2. Призначення об'єктам некоректних значень. Наприклад, якщо об'єкт зберігає значення яскравості світла зірки, то присвоєння йому

від'ємної величини буде некоректним.

3. Використання некоректної операції над об'єктом.
4. Використання некоректної адреси об'єкта.

Вищезазначені дефекти можуть виникнути внаслідок помилок на різних етапах проектування програмного забезпечення або дефектів компілятора, редактора зв'язків чи операційної системи. Сучасні мови високого рівня передбачають можливість визначення частини подібних дефектів при компіляції чи верифікації програмного забезпечення. На сьогодні єдиним засобом, що забезпечує перевірку під час компіляції та унеможливлює використання деяких помилкових арифметичних та логічних виразів є обмеження множин значень та операцій до мінімально необхідного. Отже для впровадження формальної верифікації, з одного боку, необхідно забезпечити потрібну кількість типів даних і операцій для вирішення завдань конкретної предметної області, а з іншого боку - аксіоматично ввести властивості значень та елементарних операцій над ними [6].

Оцінимо потенційні можливості ОО - парадигми у відображенні предметної області, пов'язаної з РКТ чи АЕС. Для цього звернемося до системи СІ [7], яка визначає 27 геометричних та механічних понять, наприклад, *довжина, плоский кут, об'ємний кут, маса, час, площа, об'єм* тощо. Крім того, існує 12 понять, пов'язаних з теплом, 24 – з електромагнітними явищами та 6 – із освітленням. Отже, загальна кількість понять налічує близько 70. Побудова взаємно-однозначного відображення множини понять предметної області на множину типів даних потребує створення множини типів еквівалентної потужності. Для здійснення контролю коректності обчислювальних виразів слід визначити елементарні арифметичні операції над декартовим добутком двох множин типів даних, а саме: *додавання, віднімання, множення, ділення*. У мові С++ [8] ситуація погіршується, оскільки також потрібне визначення елементарних арифметичних операцій, поєднаних із присвоєнням: +=, -=, *=, /=. Далі, крім двох класів елементарних арифметичних операцій, виникає потреба у визначенні ще 6 бінарних операцій порівняння: $<$, $>$, $\leq$, $\geq$, $\neq$, $=$ також над декартовим добутком двох множин типів даних.

Таким чином, використання у повному обсязі можливостей типізованих даних потребує визначення у С++ близько 70000 операцій, тому реалізація бібліотеки підтримки "реальних" обчислень класичними методами вимагає ресурсів на проектування та верифікацію, порівняних із великим проектом. Навіть повторне використання верифікованої та атестованої бібліотеки призведе до значного збільшення обсягу коду, що загальмує розробку ПЗ.

Дещо полегшити ситуацію може використання парадигми узагальнюючого програмування [8], що дозволяє визначати не реалізації функ-

цій (методів класів), а їх шаблони, за допомогою яких компілятор, користуючись типами як параметрами, сам генерує реалізації.

Насправді ситуація ускладнюється тим, що при обчисленнях можуть утворюватися проміжні результати похідних типів, що не належать вищезазначеним множинам, обмежуючи верифікаційну спроможність компіляційного контролю над реальним програмним кодом.

Висновки. Класичні системи типів даних для забезпечення часткової формальної верифікації ПЗ на рівні компіляції потребують додаткового ускладнення проекту, яке пропорційне квадрату кількості понять предметної області. Це, в свою чергу, через обмеженість ресурсів обчислювальних комплексів не дозволяє використовувати потенційні можливості типізації даних в повній мірі та потребує застосування інших методів досягнення необхідного рівня надійності, наприклад, збільшення кількості тестів та впровадження диверсійних технологій розроблення програмного забезпечення.

ЛІТЕРАТУРА

1. *Gibson R. Managing Computer Projects. - Prentice-Hall, 1992.*
2. *СОИ и ПРО компьютеры. – Компьютерра, 2002.– № 8 (431).*
3. *Майерс Г. Надежность программного обеспечения: Пер. с англ. – М.: "Мир", 1980.*
4. *Brooks, Fredrick P.,Jr., No silver Bullet: Essence and Accidents of Software Engineering // Computer, April 1987.*
5. *Агафонов В.Н. Типы и абстракция данных в языках программирования // Данные в языках программирования. – М.: Мир, 1982.*
6. *Хоор К. О структурной организации данных // Дал У., Дейкстра Э., Хоор К. Структурное программирование: Пер. с англ. – М.: Мир, 1975.*
7. *Яворский Б. М., Детлаф А. А. Справочник по физике. – М.: Наука, 1979.*
8. *ISO/IEC 14882, Standard for the C++ Programming Language.*

Надійшла 3.04.2002

***Манжос Юрій Семенович**, провідний інженер-програміст НВО "Хартрон Аркос". У 1984 році закінчив Харківський авіаційний інститут. З 1999 року – старший викладач Національного аерокосмічного університету "ХАІ". Область наукових інтересів – розроблення систем реального часу.*