СРЕДСТВА ПОВЫШЕНИЯ УСТОЙЧИВОСТИ МОДЕЛЕЙ ОБЪЕКТОВ НА ОСНОВЕ СИМУЛЯЦИИ СИСТЕМЫ ТИПОВ

В.Э.Вольфенгаген, С.В. Косиков, И.О. Слепцов
(г. Москва, «Институт ЮрИнфоР-МГУ)
e-mail: igor.slieptsov@mail.com

TOOLS FOR IMPROVING THE STABILITY OF OBJECT MODELS BASED ON THE TYPE SYSTEM SIMULATION

V.E. Wolfengagen, S.V. Kosikov, I.O. Slieptsov
(Moscow, «Institute JurInfoR-MGU»)

Abstract. The paper discusses the problem of the stability of semantic systems. Different approaches to improve the stability are considered, an approach based on introduction a type system is described. The proposed solution is based on using the type system simulation in a dynamically typed language. It allows detecting type errors in run-time and makes the debugging more productive. The article presents the results of the experience. The developed JavaScript library was used for tasks of modeling complex dynamic objects.

Keywords: type systems, debugging, dynamically typed languages, semantic system.

Введение. Задача повышения устойчивости семантически ориентированных систем продолжает оставаться актуальной [1]. Подходы, позволяющие исключить классы ошибок, основываются либо на системе типов [2, 3], которая позволяет во время компиляции вывести тип каждого выражения и выявить ошибки, связанные с несоответствием типов, либо на системе контрактов [4-6], которая во время исполнения проверяют выполнение проверочных утверждений. Возможно также сочетание этих двух подходов.

Известно, что в языках с динамической типизацией использование стандартных подходов наталкивается на определённые затруднения. В динамически типизированных языках не происходит статического вывода типов, при этом каждый объект во время исполнения связан с метаинформацией о типе (формате) этого объекта. Отсутствие статической проверки типов делает код уязвимым к ошибкам типизации.

В настоящей работе будет показан подход к повышению устойчивости семантических механизмов за счёт симуляции системы типов средствами динамически типизированного языка для выявления ошибок типизации на наиболее раннем этапе во время исполнения. На основе предлагаемого подхода был разработан инструмент в виде библиотеки *JavaScript*, апробированный на решении задач моделирования динамики объектов сложного вида.

Системы типов как инструмент повышения устойчивости. Среди средств повышения устойчивости семантических систем можно выделить системы типизации, необходимой частью которых является инструмент статического вывода типов. Простая система типов [1, 3-5] предоставляет набор предопределённых (атомарных) типов, основные способы комбинирования типов и правила вывода типов для каждой синтаксической конструкции языка, и традиционно используются для статического анализа и обнаружения ошибок на этапе компиляции или предобработки.

Обнаружение ошибок во время исполнения происходит при помощи системы контрактов. Преимущество контрактов в том, что контракт может быть описан с использованием выражений языка, поэтому возможно составить любое условие, которое выразимо в используемом языке.

Системы типов и контрактов могут использоваться совместно. Однако, в динамически типизированных языках отсутствует механизм статической проверки типов, механизмы связывания переменной с одним типом возможных значений или отсутствуют, или ограничены только примитивными типами и, как правило, имеются ограничение на задание пользовательские типы данных. Такие системы наиболее неустойчивы к ошибкам типизации.

Возможны разные подходы к решению этой проблемы. Типизированные над- [5, 10-16] и подъязыки [6, 13] требуют специального анализатора. Для надъязыков также требуется компилятор или транслятор. Такие решения привносят дополнительную зависимость от перечисленных инструментов. Системы контрактов и проверочных утверждений позволяют средствами языка описывать контракты и проверять их во время выполнения, они не требуют дополнительных инструментов, но, как правило, рассчитаны на работу с утверждениями относительно конкретного объекта и не предназначены для комбинирования предикатов как объектов первого порядка.

Ниже представлен новый подход, в котором рассматриваются контракты определённого аппликативного [9, 17, 18] вида *id(expr)*, состоящие из одного проверяемого объекта *expr* и идентифицирующей функции *id* – выражения языка, которое представляет тип объекта. Предлагается набор комбинаторов, которые позволяют комбинировать идентифицирующие функции в соответствии с тем, как в системах типов комбинируются типы. Подход сочетает в себе следующие естественные преимущества: (1) не требуется дополнительных инструментов (специальных компиляторов, интерпретаторов и т.п.); (2) язык описания формата значений переменной близок к описанию типа в простой системе типов, допускаются расширения; (3) обнаружение ошибки типизации происходит во время исполнения.

Симуляция системы типов. В соответствии с теорией моделей, с каждым типом T может быть связано подмножество соответствующих типу объектов D универсума всех возможных объектов – экстенсионал [17, 19]. На D задано отношение эквивалентности, которое рассматривает различные представления одного экземпляра типа как эквивалентные, замена объекта на эквивалентный ему не меняет семантику программы. В общем случае отношение эквивалентности невычислимо.

В рамках данного подхода тип *T* определяется как пара, состоящая из отношения эквивалентности и идентифицирующей функции

$$id(x) = \begin{cases} x', & x:T, \\ W, & \neg(x:T), \end{cases} \qquad (1)$$

где x' – некоторый объект, эквивалентный x, символом W_{err} обозначается исключительная ситуация, параметризованная некоторым объектом *err* [20].

id(expr) является контрактом того, что *expr* имеет тип *T*. При этом *id* возвращает объект, эквивалентный исходному. Таким образом, в отличие от других систем контрактов, утверждение типизации не обязано быть отдельной командой, а может использоваться для любых выражений, которые являются подвыражениями. Если аргумент не принадлежит типу *T*, порождается исключительная ситуация с информацией, необходимой для диагностики и отладки. Идентифицирующая функция, в отличие от отношения эквивалентности, обязана за конечное время возвращать значение или порождать исключительную ситуацию.

Основным средством задания атомарных типов является сопоставление типу с характеристической функцией *p* идентифицирующую функцию

$$t_p(x) = \begin{cases} x, & p(x), \\ \mathrm{W}_{(p,x)}, & \neg p(x). \end{cases}$$

Рассмотрим, технику определения идентифицирующих функций для сложных типов. Идентифицирующая функция типа списка, элементы которого удовлетворяют типу *T* с идентифицирующей функцией *id*, имеет вид

```
function (x) {
  if (isList(x)) throw …;
  try {
    return x.map(x);
  } catch (e) {
    throw …;
```

```
    }
  }
```

Структура этой идентифицирующей функции:

– Общие ограничения (*UntypedArrayId*).

– Проверка каждого элемента идентифицирующей функцией типа элемента и формирование эквивалентного объекта (*obj.map(id)*).

– Формирование объекта ошибки.

Эта структура легко распространяется на все функторы: таблицы, списки, ассоциативные массивы, множества, – все однопараметрические типы, для которых реализован аналог функции *map* [9]. Далее абстракцией по *id* возникает комбинатор типизированного списка как способ порождения сложных (составных) типов. Ещё одним способом комбинирования является объединение типов. Экстенсионал объединение типов равен объединению их экстенсионалов. Её идентифицирующая функция имеет вид

```
function(x) {
  try {
    return id1(x);
  } catch (e) {
    try {
      return id2(x);
    } catch (e) {
      throw …;
    }
  }
}
```

Диагностика ошибок типизации. Поскольку определены комбинаторы идентифицирующих функций, которые порождают исключительные ситуации с объектами ошибок одного для каждого семейства идентифицирующих функций, естественно выделить классы ошибок (в объектно-ориентированных языках) или конструкторы объектов ошибок (в остальных). Пример диаграммы классов приведён на рис. 1.

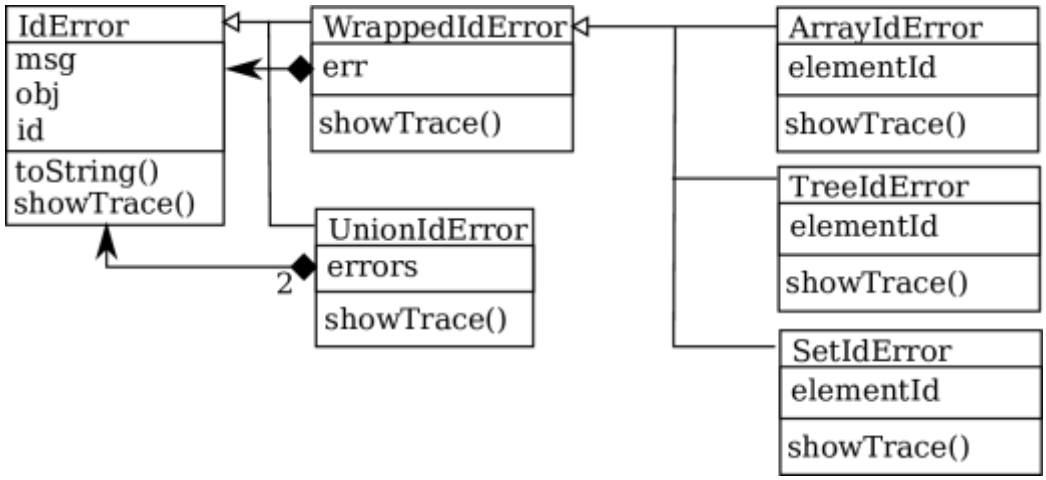

Рис. 1. Вариант UML-диаграммы классов ошибок типизации для использования в классовых объектно-ориентированных системах

Корневой класс ошибок *IdError* содержит строковое сообщение, не прошедший проверку типов объект и идентифицирующую функцию. *WrappedIdError* расширяет *IdError* полем err, которое содержит объект ошибки подобъекта. Например, класс ошибок типизиро-

ванного массива *ArrayIdError* расширяет *WrappedIdError* и помимо информации о массиве содержит объект ошибки, созданный идентифицирующей функцией элемента.

Соответствие структуры объекта ошибок и структуры проверяемого выражения достигается методом оборачивания объектов ошибок в *WrappedIdError*. Более общим случаем является задача переинтерпретации ошибок, то есть изменение некоторой функцией объекта ошибки идентифицирующей функции:

```
function MapErrorId(id, f) {
  return function(obj) {
    try {
      return id(obj);
    } catch (e) {
      throw f(e);
    }
  }
}
```

Методы *toString()* и *showTrace()* являются частью интерфейса каждого класса ошибок (рис. 1). Первый из них возвращает краткое строковое описание ошибки. Метод *showTrace()* линеаризует объект ошибок, представляя его в виде массива объектов ошибок. Например, *WrappedIdError.showTrace()* добавляет себя в конец массива *err.showTrace()* и возвращает результат. Основная идея использования *showTrace()* заключается в возможности просмотреть объект ошибок не как дерево, которое нужно разворачивать и следить за контекстом, а как последовательность ошибок от самой общей (корня) до самой частной (листа), причём каждая ошибка содержит краткое строковое описание и всю дополнительную информацию, обусловленную классом ошибки.

Заключение и выводы. Предложен новый подход повышения устойчивости семантических систем с динамической типизацией, который заключается в симуляции системы типов средствами языка и осуществления проверки условий согласованности типов в динамически типизированном языке. Разработанный механизм включает в себя возможности описания базовых пользовательских типов и средств комбинирования (декартово произведение, прямая сумма и т. п.). Предлагаемый подход обеспечивает разработку инструментов контроля типов объектов во время исполнения в языках без статической системы типов, что делает его применимым ко всем динамически типизированным языкам.

Работа частично поддержана грантами РФФИ 16-07-00914, 16-07-00912, 17-07-00893.

ЛИТЕРАТУРА

1. Ismailova L. Yu. Basic Constructions of the Computational Model of Support for Access Operations to the Semantic Network // Procedia Computer Science.

2. Пирс Б. Типы в языках программирования / Перевод с англ. М.: Издательство «Лямбда пресс»: «Добросвет», 2011.

3. Cardelli L. Type systems //ACM Computing Surveys. – 1996. – Т. 28. – №. 1. – С. 263-264.

4. Findler R. B., Felleisen M. Contracts for higher-order functions //ACM SIGPLAN Notices. – ACM, 2002. – Т. 37. – №. 9. – С. 48-59.

5. Ahmed A. et al. Blame for all //ACM SIGPLAN Notices. – ACM, 2011. – Т. 46. – №. 1. – С. 201-214.

6. Аннотации типов в Python: [сайт]. URL: http://itscreen.tk/blog/22-type-hints-in-python/. дата обращения: 2017-12-20.

7. Geuvers H. Introduction to type theory //Language Engineering and Rigorous Software Development. – Springer Berlin Heidelberg, 2009. – С. 1-56.

8. Barendregt H., Dekkers W., Statman R. Lambda calculus with types. – Cambridge University Press, 2013.

9. Вольфенгаген В. Э. Методы и средства вычислений с объектами. Аппликативные вычислительные системы. — М: JurInfoR Ltd., АО «Центр ЮрИнфоР», 2004. – XVI+789с.

10. Siek J., Thiemann P., Wadler P. Blame and coercion: together again for the first time //ACM SIGPLAN Notices. – ACM, 2015. – Т. 50. – №. 6. – С. 425-435.

11. Abadi M. et al. Dynamic typing in a statically typed language //ACM transactions on programming languages and systems (TOPLAS). – 1991. – Т. 13. – №. 2. – С. 237-268.

12. Henglein F. Dynamic typing: Syntax and proof theory //Science of Computer Programming. – 1994. – Т. 22. – №. 3. – С. 197-230.

13. Siek J. G., Taha W. Gradual typing for functional languages //Scheme and Functional Programming Workshop. – 2006. – Т. 6. – С. 81-92.

14. Wadler P., Findler R. B. Well-Typed Programs Can't Be Blamed //ESOP. – 2009. – Т. 9. – С. 1-16.

15. Siek J. G., Wadler P. Threesomes, with and without blame //ACM Sigplan Notices. – ACM, 2010. – Т. 45. – №. 1. – С. 365-376.

16. Siek J., Wadler P. The key to blame: Gradual typing meets cryptography. – 2016.

17. Kosikov S. V., Wolfengagen V. E., Ismailova L. Yu. The Presentation of Evolutionary Concepts //First International Early Research Career Enhancement School on Biologically Inspired Cognitive Architectures. – Springer, Cham, 2017. – С. 113-125.

18. Ismailova L. Yu., Kosikov S. V., Wolfengagen V. Applicative Methods of Interpretation of Graphically Oriented Conceptual Information //Procedia Computer Science. – 2016. – Т. 88. – С. 341-346.

19. Исмаилова Л. Ю. Модели представления изменяемых объектов // Сборник научных трудов SWorld. 2014. Т. 6. № 2. С. 8-13.

20. Вольфенгаген В. Э. Конструкции языков программирования. – ООО «ЮрИнфоР-Пресс», 2001.

ВЫЯВЛЕНИЕ КРИТИЧЕСКИ ВАЖНЫХ ОБЪЕКТОВ С ПОЗИЦИИ НАРУШЕНИЯ КИБЕРБЕЗОПАНСОТИ НА ПРИМЕРЕ ЭНРЕГЕТИКИ

Д.А. Гаськова
(Иркутск, Институт систем энергетики им. Л.А. Мелентьева СО РАН)
e-mail: gaskovada@gmailcom
Identify

THE IDENTIFICATION OF CRITICAL FACILITIES FROM THE POSITION OF CYBERSECURITY VIOLATION BY THE EXAMPLE OF ENERGY

D. Gaskova
(Irkutsk, Melentiev Energy Systems Institute of SB RAS)

Abstract. The article describes methods for identification of critical facilities, being a significant trend in researching critical infrastructures, particularly in the energy sector. The proposed methods are focused on the investigation of the energy object state in relation to the violation of cybersecurity of its information infrastructure. The cyber threats are believed to be important contemporary threats to energy security in Russia. The proposed methods formed the basis of development information-analytical system used for monitoring of cybersecurity violations in energy sector.

Key words: cybersecurity, cyber threats, semantic models, critical infrastructures, energy facilities

Введение. В период активной информатизации всех сфер жизни общества, была выявлена, в первую очередь на западе [1], необходимость обеспечения защищенности информа-