

Процик Петро Павлович,
Аспірант кафедри теорії та технології програмування
Київського національного університету ім. Тараса Шевченка
Адреса – Кафедра теорії та технології програмування,
Київський національний університет ім. Тараса Шевченка,
01033, Київ, вул. Володимирська, 69,
E-mail: piter.protsyk@gmail.com

Система Easy Requirements Environment автоматизованої обробки формалізованих вимог до програмних систем.

Процик П П.

Програмні системи усе більше проникають у різні сфери життя. Важливим критерієм якості програмної системи є надійність її роботи. Створення надійних систем є складною та багатоаспектною проблемою. Вона пов'язана з процесом організації розробки програмного продукту, методів які будуть застосовані, засобами перевірки якості системи.

Процес створення програмної системи, як правило, починається зі збору вимог замовника до системи і створення специфікації системи на їх основі. Цей етап у розробці програмних систем одержав назву “requirements engineering”[1]. Його мета виявити, проаналізувати, верифікувати, задокументувати вимоги до майбутньої системи, побудувати модель системи та дослідити її властивості. Автоматизація цих задач значно прискорює та покращує виконання цього етапу в розробці програмних систем [7]. Системи, які підтримують автоматизовану обробку вимог називають засобами керування вимогами (Requirement Management Tools). Порівняльний аналіз існуючих на ринку засобів наведено в роботі [7].

В доповіді розглядається система Easy Requirements Environment (ERE), розроблена автором, яка дозволяє виконувати задачі керування вимогами, а також виконувати їх автоматизований аналіз. Особливістю системи є використання формальної мови для запису вимог та виконання її автоматизованого аналізу, оскільки, аналогічні системи, зазвичай, орієнтовані на обробку вимог записаних природною мовою.

У системі вимоги розглядаються як складні сутності, згруповані у класи вимог за певними властивостями. Таке подання дозволяє зберігати різні способи подання вимог (за допомогою формальних мов, природною мовою з різним рівнем строгості) та виконувати їх аналіз у автоматичному режимі.

Класи вимог утворюють ієрархічну деревовидну структуру з підтримкою наслідування властивостей. Клас та його властивості описується за допомогою множини атрибутів. Опис кожного атрибуту

складається з імені та типу значення, яке приймає атрибут: (Name, Type). Тому формально, клас визначається таким чином:

$$\text{Class} \stackrel{\text{def}}{=} \{ (\text{Name}_i, \text{Type}_i) \mid i \in I \text{ and } \text{Name}_i \neq \text{Name}_j, \text{ якщо } i \neq j \};$$

Класи використовуються для опису властивостей та групування вимог. Вимоги в системі представлені екземплярами відповідних класів. Екземпляр класу містить конкретні значення атрибутів. Формально, це визначається таким чином:

$$\text{Instance}_{\text{Class}} \stackrel{\text{def}}{=} \{ (\text{Name}_i, \text{Value}_i) \mid i \in I \wedge (\text{Name}_i, \text{TypeOf}(\text{Value}_i)) \in \text{Class} \wedge \text{Name}_i \neq \text{Name}_j, \text{ якщо } i \neq j \}, \text{ де } \text{TypeOf}(\text{Value}) - \text{тип значення Value};$$

Наведемо приклад опису класу та його екземпляру:

```
ClassSpecifications = {
    ("Author", String) ,
    ("Creation date", Date) ,
    ("Requirement", String)
};

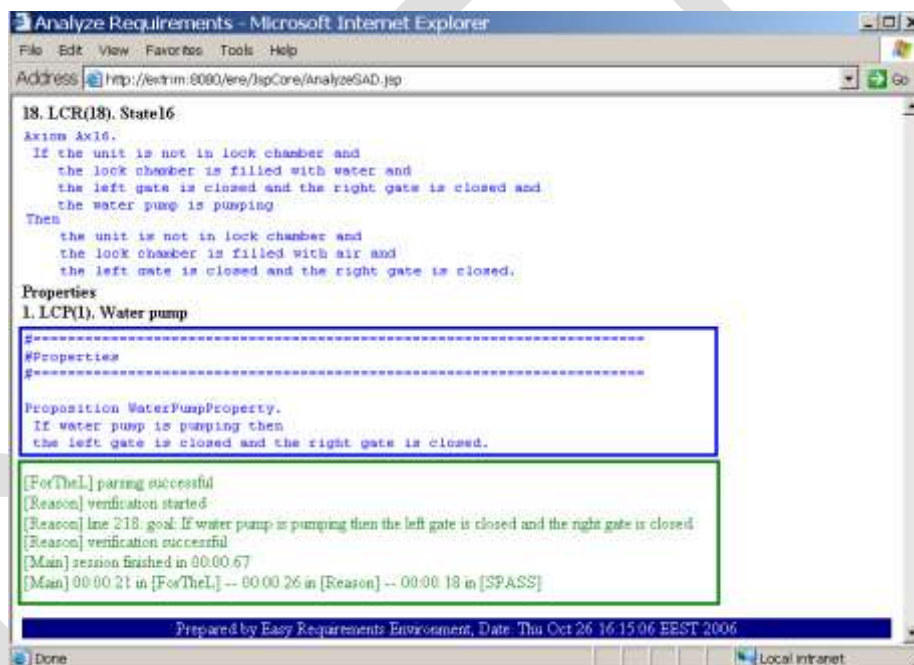
RequirementSpecifications = {
    (Author, "Petro Protsyk") ,
    (Requirement, "The system shall provide
        appropriate data and
        access security.")
};
```

У системі деревовидна структура класів вимог з відповідними екземплярами представлена таким чином (мал. 1):

- **System Definitions** [Definitions of the System]
 - Class Instances**
 - [Purpose of the system](#)
 - [Conventions](#)
 - [States description](#)
- **Specifications** [Formal Specification of the system]
 - Class Instances**
 - [LCR\(1\). Base notions](#)
 - [LCR\(2\). Initial State](#)
 - [LCR\(3\). State1](#)
 - [LCR\(4\). State2](#)
 - [LCR\(5\). State3](#)
 - **Properties** [Properties of the give specifications]
 - Class Instances**
 - [LCP\(1\). Water pump](#)

Мал.1: Дерево класів вимог

Важливим класом вимог представленим у системі є вимоги, які допускають формалізацію. Формалізація вимог виконується для того, щоб виключити з них неоднозначність, неузгодженість, некоректність, виконувати автоматичний аналіз та будувати моделі систем. В якості засобу формалізації в системі використовується мова ForTheL [4, 5]. Особливість цієї мови в тому, що вона є формальною, та її синтаксис близький до звичайної англійської мови. Це допускає її використання широким колом спеціалістів, на відміну від спеціалізованих мов специфікації. Крім того, за допомогою системи SAD [4, 5], інтегрованої з ERE, можна виконувати перевірку властивостей вимог, формалізованих мовою ForTheL (мал. 2). Формалізація вимог та логічних мов відбувається на підставі композиційно-номінативного підходу [6].



```
18. LCR(18), State16
Axiom Ax16.
  If the unit is not in lock chamber and
  the lock chamber is filled with water and
  the left gate is closed and the right gate is closed and
  the water pump is pumping
  Then
    the unit is not in lock chamber and
    the lock chamber is filled with air and
    the left gate is closed and the right gate is closed.

Properties
1. LCP(1), Water pump
-----
#Properties
-----
Proposition WaterPumpProperty.
  If water pump is pumping then
  the left gate is closed and the right gate is closed.

[ForTheL] parsing successful
[Reason] verification started
[Reason] line 218, goal: If water pump is pumping then the left gate is closed and the right gate is closed.
[Reason] verification successful
[Main] session finished in 00:00:67
[Main] 00:00:21 in [ForTheL] -- 00:00:26 in [Reason] -- 00:00:18 in [SPASS]
```

Мал.2: Перевірка властивостей ForTheL специфікацій

Слід також зазначити, що архітектура системи дозволяє підключати додаткові програмні засоби обробки формальних специфікацій та представлень вимог іншими мовами.

Система побудована з використанням технології J2EE і має WEB інтерфейс. Передбачена одночасна робота багатьох користувачів з системою в on-line режимі. Існує підсистема керування доступом.

Серед додаткових функцій слід відзначити лексичний аналіз текстуальних вимог, автоматичну підготовку документу специфікації системи, роботу з різними типами даних – текстовими, графічними, документами різних форматів.

Подальший розвиток системи буде пов'язаний із збагаченням методів обробки вимог, побудов моделей системи, а також розвитком мов формалізації.

ЛІТЕРАТУРА:

1. Соммервилл, Иан. Инженерия программного обеспечения, 6-е издание. : Пер. с англ. – М. : Издательский дом «Вильямс», 2002. – 624 с.: ил. ISBN 5-8459-0330-0 (rus), ISBN 0-201-39815-X (eng), Pearson Education Limited, 2001.
2. IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications, IEEE 1998.
3. IEEE Computer Society, Guide to the Software Engineering Body of Knowledge (SWEBOOK), version 2004.
4. Alexander Lyaletski, Konstantin Verchinine, Anatoli Degtyarev, and Andrey Paskevich. System of Automated Deduction (SAD): Linguistic and Deductive Peculiarities. In M.A. Klopotek, S.T. Wierzchon, and M. Michaliwicz, editors, *Advances in Soft Computing: Intelligent Information Systems 2002*, pages Physica-Verlag, Springer, pages 413-422, 2002.
5. System for Automated Deduction, SAD ([http:// ea.unicyb.kiev.ua/sad.en.html](http://ea.unicyb.kiev.ua/sad.en.html)).
6. Nikitchenko, N.S.: Composition nominative approach to the explication of the notion of program, *Problems of Programming*, No 1, 1999, pp. 16-31.
7. Karl E. Wiegers: Automating Requirements Management, *Software Development magazine*, July 1999.