УДК 004.434

В. І. Межуєв, д-р техн. наук,
Т. В. Бодненко, канд.пед. наук

СТВОРЕННЯ OWL-DL ОНТОЛОГІЙ РОЗПОДІЛЕНИХ ПАРАЛЕЛЬНИХПРОГРАМНИХ СИСТЕМ

Анотація. *Присвячена аналізу доцільності створення OWL-DLонтологій розподілених паралельних програмних систем етапу інформаційної технології предметно-орієнтованого математичного моделювання. Визначення типів метамоделі як OWL класів і використання OWL-обмежень як граматики метамоделі надає можливість побудови валідних моделей програмних систем.*

Ключові слова*: інформаційна технологія, математичне моделювання, моделі програмних систем, модулювання апаратних систем, предметно-орієнтоване математичне моделювання, логічний аналіз, онтологічна метамодель*

V. Mezhuyev, ScD.,
T. Bodnenko, PhD.

DEVELOPMENT OF OWL-DL ANTOLOGIES OF DISTRIBUTED PARALLEL SOFTWARE SYSTEMS

Abstract. *This paper analyses the feasibility of development of OWL-DL ontologies of distributed parallel software systems as a stage of information technology of domain-specific mathematical modelling. Identification of the types of metamodel as OWL classes and use OWL-restrictions as a metamodel grammar leads to the construction of valid (i.e. the meeting system specifications) models of software systems.*

Keywords: *information technology, ontology,metamodel, mathematical modelling, models of software systems, object-oriented mathematical modelling, logical analysis*

В. И. Межуев, д-р техн. наук,
Т. В. Бодненко, канд. пед. наук

СОЗДАНИЕ OWL-DL ОНТОЛОГИЙ РАСПРЕДЕЛЕННЫХ ПАРАЛЛЕЛЬНЫХПРОГРАММНЫХ СИСТЕМ

Аннотация. *Посвящена анализу целесообразностисоздания OWL-DL онтологий распределенныхпараллельныхпрограммных систем как этапаинформационной технологи предметно-ориентированногоматематическогомоделирования. Определение типов метамодели как OWL классов и использование OWL-ограничений как грамматики метамодели позволяет построения валидных (т.е. соответствующих спецификациям) моделей программных систем.*

Ключевыеслова: *информационная технология, математическое моделирование, моделипрограммных систем, моделирования аппаратных систем, предметно-ориентированное математическое моделирование, логический анализ, онтологическая метамодель*

Вступ

У попередніх роботах нами був запропонований метод розробки метамоделей на основі логічних моделей предметних областей (ПрО)[1]. Також були розглянуті особливості застосування *OWL-DL*онтологій для розробки метамоделей та моделей програмних систем (ПС)[2].

У даній статті створення онтологій розглядається як етап інформаційної технології (ІТ) предметно-орієнтованого математичного моделювання (*DomainSpecificMathematicalModelling*) (*DSMM*) [3]. Розробка метамоделі здійснюється шляхом аналізу предметних та математичних властивостей про доцільність застосування онтологій для розробки

метамоделейдоводиться на прикладі розподілених паралельних ПС. На основі мови онтологій *OWL* (*Web Ontology Language*) [4] створюються метамоделі для моделювання програмних систем, а також топологій мереж розподілених обчислювальних вузлів.

Основна частина

Ринок сучасного програмного забезпечення (*SW*) та апаратних засобів (*HW*) потребує збільшення продуктивності роботи розробників, зниження вартості та більшої кількості функціональних можливостей комп'ютерних систем. Крім того, сучасні *SW* й *HW* системи досягли того рівня складності, для якого вирішальними стають вимоги відмовостійкості та безпеки. Саме тому створення інформаційної технології моделювання програмних та апаратних систем, що дозволить здійснити перевірку їх властивостей ще на етапі проектування, є актуальною науковою та технічною проблемою.

Розглянемо застосовність *IT DSMM* до створення метамоделей для моделювання програмних та апаратних систем. Концептуальне моделювання, зокрема, висунення вимог та специфікацій, є першою фазою розробки будь-якої системи. Запропонована у [5] онтологічна метамодель була використана нами для специфікації ПС на природній мові. Використання даної концептуальної метамоделі дозволило структурувати попередньо розрізнені вимоги та специфікації до майбутньої системи, здійснити їх уніфікацію, виділити різні аспекти у розробці системи та ін.

Між тим, доцільність розробки онтологічної метамоделі не обмежується лише структуруванням тверджень про властивості майбутньої системи у рамках певної концептуальної схеми. *OWL* є, власне, сімейством логік, що мають суттєво різні властивості. Зокрема, *OWL-DL* (*DL* – *DescriptionLogic*) дозволяє здійснити перевірку властивостей ПС за допомогою систем логічного аналізу (англ. *reasoners*) (наприклад, FACT++ [7]). Застосування *OWL-DL* передбачає поєднання двох найважливіших стадій розробки систем - концептуального моделювання (зокрема, специфікації систем) й формальної перевірки (верифікації) властивостей систем.

Розглянемо технологію створення метамоделі на основі *OWL-DL* онтології. Найпоширенішим способом побудови онтологій є організація об'єктів ПрО в таксономію підклас-суперклас [4]. На рис. 1 представлено спрощену онтологію паралельного програмного додатку (англ. *Application*), який відображає*hasSubclass* (мати підклас, наприклад, *TaskhasSubclassUserTask*) між класами й *hasInstance* (мати екземпляр), між класом та його екземплярами (наприклад, *UserTaskhasInstance Task*1). Зазначимо, що у даній таксономії як корінь використовується тип *Application* (в *OWL* традиційно всі класи є підкласами класу 'Річ' – англ. *Thing* [4]).

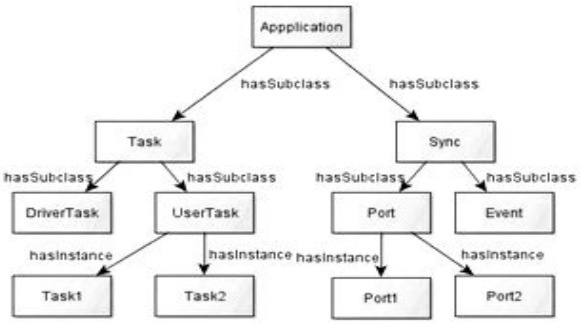

Рис. 1. Онтологія програмного додатку

Співвідношення *hasSubclass* в *OWL* визначають, що коли *Port* (порт) є підкласом типу *Sync* (від англ. *Synchronization* – синхронізація), всі елементи множини *Port* також належать до множини *Sync*. Аналогічне правило застосовне і до інших *OWL* класів. Це також означає наслідування властивостей: наприклад, якщо всі *Sync* мають предикат синхронізації, а *Port* є *Sync*, то з цього випливає, що *Port* також повинен мати властивість 'предикат синхронізації'.

Наведемо визначення класів *Task* та *Sync* за допомогою дескрипційної логіки:

$$Task \equiv DriverTask \cup UserTask \quad (1)$$

$$Sync = Port \cup Event\,. \quad (2)$$

Класи *Task* і *Sync* визначені як такі, що не пересікаються. Це означає, що будь-який концепт онтології не може бути екземпляром більш, ніж одного з цих класів. Наступна формула *DL* виражає це відношення:

$$Task \cap Sync = \varnothing. \quad (3)$$

Не всі класи в онтології ПС можуть мати прямі екземпляри. Наприклад, *Sync* є *OWL*суперкласом для *Port, Event* та інших типів об'єктів синхронізації, лише які можуть мати екземпляри. Ця властивість формалізується за допомогою наступних *DL* формул:

$$\forall has\text{Instance}.Sync = \varnothing$$

$$\forall has\text{Instance}.\forall hasSubclass.Sync \neq \varnothing. \quad (4)$$

Формули (1) – (4) є предикатами *OWL* класів, що у контексті *IT DSMM* застосовуються як правила граматики метамоделі для визначення способів інстанціації та поєднання екземплярів типів *Task* та *Sync*. У той же час (1) – (4) можуть слугувати входом для застосування логічних аналізаторів (зокрема,

FACT++) для верифікації моделей ПС, побудованих як *OWL-DL* онтології. У даному випадку може бути перевірена коректність співвідношень підклас-суперклас та клас-екземпляр онтології ПС. Крім того, логічні аналізатори дозволяють перевірити відповідність понять, обчислити виведені ієрархії та виділити еквівалентні класи онтологій.

Але зазначимо, що *OWL-DL* онтології охоплюють лише статичні структурні властивості ПрО (у нашому прикладі, розподілених паралельних ПС). Щоб відтворити динаміку поведінки системи необхідно визначити логічні формули для функціональних, часових та інших властивостей ПрО. Перевірка таких властивостей залишається за межами можливостей *OWL-DL* онтологій і пов'язаних з ними логічних аналізаторів.

Розглянемо особливості визначення правил граматики метамоделей як *OWL*-властивостей. У вищесказаному предикати *OWL* класів були використані як граматика, яка визначає правила створення екземплярів типів метамоделі. Проаналізуємо, як засобами *OWL-DL* можна визначити способи поєднання екземплярів типів метамоделі у складні структури.

Співвідношення об'єктів онтології ПС можна виразити за допомогою властивостей об'єктів *OWL*. Прикладами взаємодії об'єктів паралельної програмної системи є *PutData* (відіслати дані) і *GetData* (забрати дані), що пов'язують екземпляри типів задачі (*Task*) й об'єкту синхронізації (*Sync*).

OWL властивості об'єктів поєднують екземпляри з певної області визначення з іншими екземплярами, що мають область значень. Областю визначення *PutData* є множина екземплярів класу задачі (тип *Task*). Область значень - множина екземплярів класу об'єктів синхронізації (тип *Sync*). Ці множини використовуються для перевірки валідності програмних систем. Наприклад, якщо має місце визначення, що певний об'єкт$_1$ пов'язаний властивістю *PutData* з деяким об'єкт$_2$, то з цього випливає, що об'єкт$_1$ є екземпляром типу *Task*, а об'єкт$_2$ є екземпляром типу *Sync*.

Властивості *OWL* також можуть мати підвластивості, які відображають моделювання ієрархій властивостей ПрО. Підвластивостіспеціалізують власні супервластивості (наприклад, властивість *hasName*- мати ім'я, спеціалізує більше загальну *OWL* властивість *hasAttribute*- мати атрибут). Зокрема, поділяємо *OWL* властивості об'єктів на сервіси керування задачами (наприклад, *StartTask* й *StopTask*) і сервіси синхронізації (наприклад, *PutData* й *GetData*).

Кожна властивість класу *OWL* онтології може мати відповідну інверсну властивість. Наприклад, властивість *isPartOf* (є частиною) має обернену властивість *consistsOf* (складається з), тобто якщо "*Topology-consistsOfNodes*" (топологія складається з вузлів) з цього випливає, що "*Nodesis-PartOfTopology*" (вузли є частиною топології).

Інверсність дозволяє формально визначити семантично протилежні пари *OWL* властивостей, наприклад, *StartTask* й *StopTask*, *PutData* та *GetData* та ін., що зумовлено симетрією взаємодії задач ПС. Зазначимо, що принцип симетрії взаємодії задач використовуємо для верифікації програмних додатків. У той же час зазначимо, що OWL властивості,виражені за допомогою інверсних пар не є семантично еквівалентними: наприклад з того положення, що $Task_1 StartTask Task_2$ не випливає, що $Task_2 StopTask Task_1$. Формулювання ж подібної властивості є констатацією причинно-наслідкового зв'язку між $Task_1$ та $Task_2$, та потребує застосування логічної імплікації.

Таким чином, *OWL*-обмеження використовуємо для побудови правил граматики метамоделей, що слугують для побудови відповідних специфікаціям моделей програмних додатків (*Application*). Такі моделі ПС будуються як множини, що включають екземпляри типів об'єктів синхронізації (*Sync*) та задач (*Task*):

$$Application \equiv Task \cup Sync . \qquad (5)$$

Визначимо порт (*Port*) як один із об'єктів синхронізації взаємодії програмних задач. Валідний додаток припускає, що якщо задача $Task_1$ здійснює виведення даних у порт (викликає *PutData*), то повинна існувати інша задача $Task_2$, що забирає ці дані (викликає *GetData*) з цього ж порту.

При цьому кожне *OWL-DL* обмеження визначає анонімний клас (множину), що містить

екземпляри, які задовольняють дану логічну формулу. Наприклад, *OWL* обмеження для класу задачі *Task*: *PutData.Port* визначає анонімний клас, екземпляри якого є об'єктами типу *Task* і беруть участь у взаємодії *PutData*. Інше *OWL* обмеження для класу задачі: *GetData.Port* визначає анонімний клас, екземпляри якого є членами класу задачі (типу *Task*) й беруть участь у взаємодії *GetData*. Перетин цих анонімних класів визначає суперклас валідної задачі ValidTask, елементи якого беруть участь як у PutDatatакіGetData взаємодіях.

$$ValidTask \equiv \forall PutData.Port \cap \forall GetData.Port \quad .(6)$$

Таким чином, правило валідного відношення з кожним портом програмного додатку будується як перетин множин, що включають всі екземпляри задач, задіяних у *GetData* та *PutData*. Таке *OWL* обмеження є специфікацією валідного додатку, а його результат визначає множину валідних екземплярів додатків.

Зазначимо, що визначення (6) відображає властивість симетрії взаємодії задач.

Даний підхід є прикладом застосовності *OWL* онтологій для створення метамоделей для моделювання ПС і перевірки їх властивостей. Можна навести також й інші приклади, що ілюструють можливість запропонованої *IT*. Наприклад, наведемо специфікацію головної задачі програмного додатку *MainTask*, як таку, що включає визначення головної функції *main*:

$$MainTask \equiv Task \cap \exists hasFunction.main \quad . \quad (7)$$

MainTask належить до класу *Task* і включає (принаймні одну) головну функцію *main*. Зазначимо, що для визначення властивості, задача повинна мати *тільки одну* головну функцію. Це передбачає застосування *OWL* обмежень на кількість елементів класу (що визначає точну кількість відношень, в яких може брати участь екземпляр класу).

Іншим шляхом є конкретизація властивості *hasFunction*, а саме, визначення взаємодії *hasMainFunction* як функціональної властивості. Це означає, що для даного екземпляру класу додатків може існувати лише один екземпляр класу функцій (головна функція), який пов'язаний з екземпляром додатку через властивість *isMainFunctionOf*. Враховуючи, що валідний програмний додаток завжди має головну функцію, таким чином, властивість (7) є необхідною частиною специфікації валідних ПС.

Розглянемо приклад створення онтології топології обчислювальних вузлів розподілених ПС. Розробку онтології топології визначаємо як задачу, що передує побудові метамоделі для моделювання обчислювальної мережі.

З метою використання сервісів гетерогенних і розподілених обчислювальних вузлів, будемо використовувати єдине (уніфіковане) подання властивостей кожного вузла (типу *Node*) у певній конфігурації, що включають *HW*-атрибути (*Parameters, Paths, Compiler-Options* і т.ін.).

Побудуємо онтологію обчислювальної мережі, яка складається з *HW* вузлів (типу *Node*) і зв'язків (типу *Link*), що з'єднують ці вузли. Зв'язки можуть бути односпрямованими (*Unidirectional*) або двоспрямованими (*Bidirectional*). Щоб бути досяжним, кожен вузол у даній топології повинен мати, принаймні лише один вхід й один вихід. Таким чином, між будь-якими двома вузлами обчислювальної мережі повинен існувати, принаймні, один шлях.

Оскільки односпрямовані тадвоспрямовані зв'язки є відношеннями між двома вузлами, їх можна розглядати як бінарні властивості *OWL*.Наприклад *hasUniLink* (має односпрямований зв'язок), і *hasBiLink* (має двоспрямований зв'язок).

Двоспрямований зв'язок є симетричним, тобто, якщо $Node_1$ має зв'язок з $Node_2$, із цього випливає, що $Node_2$ має зв'язок з $Node_1$ (рис. 2). З математичної точки зору, властивість *hasBiLink* є власною інверсною властивістю.

Рис. 2. Симетрія двоспрямованого зв'язку

Двоспрямований зв'язок є транзитивним. Якщо $Node_1$ має зв'язок з $Node_2$, а$Node_2$ з $Node_3$, то $Node_1$ є досяжним з $Node_3$, тобто між ними існує шлях (рис. 3). Зазначимо, що ми не розрізняємо у даних міркуваннях поняття логічного зв'язку й фізичного шляху.

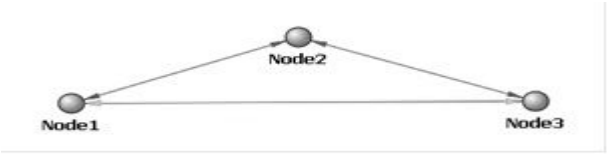

Рис. 3. Транзитивність двоспрямованого зв'язку

Визначені специфікації дозволяють побудувати правила граматики онтологічних метамоделей для моделювання валіднихтопологій обчислювальних вузлів шляхом накладання обмежень на *OWL* класи, наприклад:

$$\exists hasBiLink.Node. \quad (8)$$

Вираз (8) визначає анонімний клас вузлів, що мають, принаймні, один двоспрямований зв'язок з іншим вузлом. Такий клас *OWL* можна розглянути як специфікацію правила побудови простішої моделі деякої реальної топології.

Щоб змоделювати властивість досяжності (а саме, щоб бути досяжним, кожен вузол топології повинен мати як вхідний, так і вихідний зв'язок), визначаються дві під властивості *hasUniLink*: *hasInUniLink* (має вхідний односпрямований зв'язок) і *hasOutUniLink* (має вихідний односпрямований зв'язок).

Перетин таких *OWL* класів, які мають вхідні й вихідні односпрямовані зв'язки, дає нам анонімний клас, що відповідає специфікації:

$$\exists hasInUniLink.Node \cap \exists hasOutUniLinkNode. \quad (9)$$

Враховуючи, що вузли можуть також мати двоспрямовані зв'язки, можна розширити визначення валіднихтопологій(тобто відповідних вимозі наявності хоча б одного шляху між будь-якими двома вузлами) у такий спосіб:

$$(\exists hasInUniLink.Node \cap \exists hasOutUniLinkNode) \cup \exists hasBiLinkNode. \quad (10)$$

На основі даних специфікацій за допомогою систем логічного аналізу перевіряється валідність топології обчислювальних вузлів ще на стадії концептуального моделювання ПрО. У наступних роботах буде розширена множина властивостей ПрО, що можуть бути змодельовані й перевірені на етапі концептуального моделювання. Одним із пріоритетних напрямків являються побудовиметамоделі на основі часової логіки дій (*TemporalLogicofActions*) [8] для дослідження та перевірки властивостей поведінки систем.

Висновки

Створення метамоделей на основі мови онтологій *OWL* лежить у рамках методу розробки метамоделей як логіко-алгебраїчних систем. Визначення типів метамоделі як *OWL* класів і використання *OWL*-обмежень як граматики метамоделі надає можливість побудови валідних (тобто відповідних системним специфікаціям) множин екземплярів типів. Доведена доцільність онтологічного моделювання як етапу розробки метамоделей програмних систем. Побудова онтологічних метамоделей є способом перевірки властивостей системи на ранніх етапах її розробки (починаючи зі специфікації вимог). Розширення виразності онтологій часовими властивостями, модальною логікою та ін. дозволяє поєднати етап концептуального моделювання й стадію перевірки динамічних властивостей систем.

Список використаної літератури

1. Межуєв В. І. Методрозробки метамоделей наосновілогічнихмоделейпредметнихобластей / В. І. Межуєв, О. М. Литвин, О. О. Литвин // Вісник НТУ «ХПІ» «Математичнемоделювання в техніцітатехнологіях. – Харьків .: НТУ «ХПІ». – 2012. – № 2. – С. 153 – 161.

2. Межуев В. И. Моделированиесвойствоперационнойсистемыреальноговремени*OpenComRTOS*припомощи *OWL-DL* онтологий / В. И. Межуев // Збірникнауковихпраць ДонНТУ серії «Інформатика, кібернетикатаобчислювальнатехніка». – 2009. – Вип. 10(153). – С. 39 – 46.

3. Межуєв В. І. Інформаційнатехнологіярозробкикомплексних інструментальнихзасобів предметно-орієнтованого математичного моделювання / Віталій Іванович Межуєв : Автореф. дис. д-ра техніч. наук. – Одеса : ОНПУ, 2012. – 36 с.

4. OWLWebOntologyLanguageGuide. -[Електроннийресурс]. Режим доступу -http://www.w3.org/TR/owl-guide, (accessed 05.12.2014).

5. Mezhuyev V., Sputh B., and Verhulst E., (2010), Interacting Entities Modelling Methodology for Robust Systems Design, Advances in System Testing and Validation Lifecycle, *CPS Publ.*, pp. 75 – 80

6. Baader F., Calvanese D., McGuinness D. L., Nardi D., and Patel-Schneider P.F., (2003), The Description Logic Hand book: Theory, Implementation, Applications, *Cambridge University Press Publ.*, Cambridge, UK, 624 p.

7. FACT++ [Електронний ресурс]. – Режим доступу : http://owl.man.ac.uk/factplusplus (accessed 05.12.2014).

8. Lamport L., (2002), Specifying Systems: the TLA+ language and Tools for Hardware and Software Engineers, *Addison-Wesley Publ*, Boston, 364 p.

Отримано 29.10.2014

References

1. Mezhuyev V.I. Metodrozrobki metamodelei na osnovi logichnikh modelei predmetnikh oblastei [A Method for Developing Metamodelson the Base of Logical Models of Domains], (2012), *Vestnik NTU "HPI" "Mathematical Modelling in Engineering and Technology. NTU "KPI" Publ.*, Kharkov, Ukraine, Vol. 2, pp. 153 – 161 (In Ukrainian).

2. Mezhuev V.I. Modelirovanie svoistv operatsionnoi sistemy real'nogo vremeni OpenComRTOS pri pomoshchi OWL-DL ontologii [Modelling Properties of Real-time Operating System OpenComRTOSby OWL-DL Ontologies], (2009), *Scientific Papers of Donetsk National Technical University Series "Informatics, Cyberneticsand Computer Science" Publ*., Donets'k, Ukraine, Vol. 10 (153), pp. 39 – 46 (In Russian).

3. Mezhuyev V.I. Informatsiina tekhnologiya rozrobki kompleksnikh instrumental'nikh zasobiv predmetno-orієntovanogo matematichnogo modelyuvannya [Information Technology for Development of Integrated Tools for Domain Specific Mathematical Modelling], (2012), *Thesis of Doctorof Technical Sciences ONPU Publ.*, Odessa, Ukraine, 36 p. (In Ukrainian).

4. OWL WebOntologyLanguageGuide (In English), Available at: http://www.w3.org/TR/owl-guide (accessed 05.12.2014).

5. Mezhuyev V., Sputh B., and Verhulst E., (2010), Interacting Entities Modelling Methodology for Robust Systems Design, Advances in System Testing and Validation Lifecycle, *CPS Publ.*, pp. 75 – 80 (In English).

6. Baader F., Calvanese D., McGuinness D. L., Nardi D., and Patel-Schneider P.F., (2003), The Description Logic Hand book: Theory, Implementation, Applications, *Cambridge University Press Publ.*, Cambridge, UK, 624 p. (In English).

7. FACT++[FACT++], Available at: http://owl.man.ac.uk/factplusplus (accessed 05.12.2014) (In English).

8. Lamport L., (2002), Specifying Systems: the TLA + language andTools for Hardware and Software Engineers,*Addison-Wesley Publ.*,Boston, 364 p. (In English).

Межуєв
ВіталійІванович
д-р техн. наук, проф. каф.
інформатики та програмноїінженерії факультету комп'ютерних таенергозберегаючихтехнологійБердянського державного педагогічного ун-ту.
E-mail: mejuev@ukr.net

Бодненко
Тетяна Васильєвна,
кад.пед.наук, доц., каф.
автоматизації та комп'ютер-но-інтегрованих технологійУчбово-наукового ін-ту фізики, математики та комп'ютер-но-інформаційних систем Черкаського нац. ун-ту ім. Богдана Хмельницького.
E-mail: tanja25@list.ru