УДК 681.3.06, 519.8

С.В. СЕМЕНЕЦ

МЕТОДЫ ОЦЕНКИ ПРОИЗВОДИТЕЛЬНОСТИ ПРОГРАММНЫХ СИСТЕМ ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ

Abstract: *The performance estimation techniques of the mobile devices software at the early lifecycle stages is examined in the article. There comparative analysis and examples of practice are shown. Special attention is paid to problem of using the formal mathematical methods for estimation of software characteristics within the bounds of UML specification language.*
Key words: *software performance, mobile system, specification language, lifecycle.*

Анотація: *У статті розглянуті існуючі методи оцінки продуктивності програмних систем для мобільних пристроїв на ранніх стадіях життєвого циклу, проведено їх порівняльний аналіз, показані приклади використання на практиці. Особлива увага приділена дослідженню питання використання існуючих формальних математичних методів для оцінки характеристик програмної системи в рамках мови специфікації UML.*
Ключові слова: *продуктивність, мобільна система, мова специфікації, життєвий цикл.*

Аннотация: *В статье рассмотрены существующие методы оценки производительности программных систем для мобильных устройств на ранних стадиях жизненного цикла, проведен их сравнительный анализ, показаны примеры применения на практике. Особое внимание уделено исследованию вопроса использования существующих формальных математических методов для оценки характеристик программной системы в рамках языка спецификации UML.*
Ключевые слова: *производительность, мобильная система, язык спецификации, жизненный цикл.*

1. Введение

В процессе создания программной системы возникает необходимость оценки ее производительности. Особенно актуальна эта задача на ранних стадиях жизненного цикла. Такая оценка дает возможность на этапе проектирования делать выводы о правильности тех или иных архитектурных решений, перестраивать модель будущей программной системы в зависимости от требований к ее производительности, более того, оценка позволяет достигать лучшего соответствия требованиям, что, в свою очередь, позволяет снизить риск потери контроля над проектом или риск неприемлемой производительности.

На сегодняшний день существуют несколько математических формализмов, возможности которых можно использовать для построения модели оценки производительности программной системы, применяя как стохастические методы, так и методы имитационного моделирования.

В современных программных системах архитектура является решающим фактором при разработке концепций, конструировании, управлении и развитии создаваемой системы. Как правило, архитектура описывается с помощью нескольких взаимосвязанных представлений, каждое из которых является одной из возможных проекций организации и структуры системы. Исходя из того, что подавляющее большинство современных программных систем проектируется на базе UML-моделей, необходимо провести сравнительный анализ методов оценки производительности системы на основе как статических, так и динамических диаграмм этого языка моделирования.

Особенностью данного исследования является то, что его объектом есть программные системы для мобильных устройств. Характеристики проектируемой мобильной системы можно разбить на два класса: количественные характеристики и качественные (характеристики достаточности или достижимости). К первому классу относятся, прежде всего, такие характеристики системы, как производительность, объем доступной памяти, разрешение экрана. К

качественным относятся возможные тупиковые ситуации, достаточность имеющихся ресурсов. Отсюда возникают задачи расчета количественных характеристик, поиска и анализа качественных.

Кроме того, необходимо исследовать вопрос использования существующих формальных математических методов для оценки характеристик мобильной программной системы в рамках выбранной методологии и языка спецификации, что обусловлено последующим использованием модели для генерации исходного кода. Существуют несколько методов решения этой задачи, на которых мы остановимся подробнее. Большинство этих методов основано на предварительном построении модели производительности, однако некоторые предусматривают создание модели выполнения программного обеспечения (ПО), модели работы системы, а также их комбинации.

Необходимо отметить, что комитет OMG также работает над решением этой проблемы, а именно: ставит цель разработки шаблона, обладающего следующими возможностями:

– получение требований по производительности в контексте проектирования;

– привязка характеристик производительности к отдельным элементам UML-модели;

– спецификация параметров выполнения, которые могут быть использованы средствами моделирования для расчета характеристик производительности;

– представление результатов расчета производительности.

2. Модели оценки производительности

К моделям оценки производительности программной системы, как правило, относят системы массового обслуживания (СМО), стохастические алгебры процессов, стохастические сети Петри, а также имитационные модели.

Рассмотрим их более подробно.

1. **Аппарат СМО** широко применяется в качестве моделей производительности с целью представления и анализа ресурсов распределенных систем. Как отмечено в [1], СМО представляет собой совокупность пунктов, в которую поступают объекты (входящий поток) через некоторые промежутки времени, подвергаются там соответствующим операциям (обслуживанию) и затем покидают систему (выходящий поток). Математический аппарат СМО представляет собой ряд методов оценки и анализа характеристик моделируемой системы. К этому же типу можно отнести и расширенные СМО, которые предназначены для представления систем реального времени, синхронных и параллельных систем, систем с конечной емкостью очереди или памяти и совместно используемыми ресурсами.

Другим расширением СМО являются так называемые многоуровневые СМО (Layered Queuing Network), позволяющие применять шаблоны клиент-серверного взаимодействия в параллельных и распределенных программных системах [2, 3]. Основное отличие между обычными и многоуровневыми СМО состоит в том, что в «классических» СМО сервер может становиться клиентом (заказчиком) по отношению к другим серверам во время обслуживания запросов собственных клиентов. Многоуровневые СМО представляются как нециклический граф, узлы которого являются программными сущностями и аппаратными устройствами, а дуги – запросы сервисов.

2. **Стохастические временные сети Петри.** В соответствии с [4] стохастической сетью Петри называется недетерминированная модель сети Петри, в которой у каждого перехода случайное время срабатывания. Сеть Петри есть пятерка:

$$C = \{P, T, \Pr e, Post, m_0\}, \quad (1)$$

где P – множество мест; T – множество переходов; $P \cap T = \varnothing$; $\Pr e : P \times T \to IN$ – входная функция; $\Pr e : T \times P \to OUT$ – выходная функция; $m_0 : P \to IN$ – начальная разметка.

Стохастические сети Петри используются, как правило, при моделировании поведения параллельных систем. Как отмечено в [28], стохастической временной сетью Петри называется сеть Петри, где экспоненциально распределены случайные переменные, представляющие время действия ассоциированных с ними переходов.

Сети Петри обладают следующими свойствами и соответствующими им методами анализа:

1. Безопасность (число фишек 0 или 1 позволяет реализовать позицию триггером).

$$\forall \mu' \in R(C, \mu) \to \mu'(p_i) \le 1, \quad (2)$$

где μ' – текущая маркировка сети Петри; $R(C, \mu)$ – множество достижимости R сети Петри C с маркировкой μ; $p_i \in P$ – позиция сети Петри.

2. Ограниченность (позиция k-ограничена или k-безопасна, если число фишек не может превышать k, реализуется счетчиком).

$$\forall \mu' \in R(C, \mu) \to \mu'(p_i) \le k \ , \quad (3)$$

где k – целое число ограничения.

3. Сохранение (если общее число фишек в сети остается постоянным, сеть строго сохраняющая).

$$\sum_{p_i \in P} \mu'(p_i) = \sum_{p_i \in P} \mu(p_i) \ . \quad (4)$$

4. Активность.

Как отмечено в [5], переход называется активным, если он не заблокирован (нетупиковый). Переход t_j сети Петри C называется потенциально запустимым в маркировке μ, если существует маркировка $\mu' \in R(C, \mu)$, в которой t_j разрешен.

5. Достижимость и покрываемость.

Задача достижимости состоит в том, чтобы для данной сети Петри с маркировкой μ и маркировки μ' определить $\mu' \in R(C, \mu)$.

Задача покрываемости состоит в том, чтобы определить, существует ли такая достижимая маркировка $\mu'' \in R(C, \mu)$, что $\mu'' \ge \mu'$ для данной сети Петри с маркировкой μ и маркировки μ'.

6. Последовательность запусков.

Одна из задач анализа сетей Петри, решение которой сводится к определению очередности запуска переходов сети. Этот вопрос анализа лежит в области языков сетей Петри.

7. Эквивалентность и подмножества.

Преимущественно используются в целях оптимизации. Суть в удалении пассивных переходов и позиций, а также в переопределении некоторых переходов. Основная сложность состоит в том, что в зависимости от типа эквивалентности (равенство множеств достижимости либо равенство множеств последовательностей) нельзя изменять количество позиций или переходов.

К методам анализа сетей Петри относятся:

1. Дерево достижимости. Представляет множество достижимости сети Петри, полученное путем перебора всех возможных последовательностей запусков переходов. Существуют несколько способов приведения дерева к конечному представлению. Как правило, все они относятся к средствам ограничения новых маркировок: терминальные и дублирующие вершины. Важным свойством алгоритма построения дерева достижимости является то, что он заканчивает свою работу [6, стр. 95].

С помощью этого метода существует возможность определения безопасности сети Петри, ограниченности, сохранения, а также ее покрываемости. К недостаткам данного метода относится то, что его применение невозможно для всех задач достижимости и активности, а также определения последовательности запусков. Решение этих задач ограничено возможностью бесконечного числа маркировок ω.

2. Матричные уравнения. Подход основан на матричном представлении сети Петри. Суть его состоит в том, что сеть Петри представляется в виде двух матриц D^+ и D^-, соответствующих входной и выходной функциям сети. Каждая матрица имеет m строк, где m – количество переходов и n столбцов, причем n – количество позиций. Таким образом, получаем

$$D^-\left[j,i \right] = \#(p_i, I(t_j)), \tag{5}$$

где $D^-\left[j,i \right]$ – элемент j-й строки i-го столбца матрицы D^-;

$I(t_j)$ – входная функция перехода t_j;

$$D^-\left[j,i \right] = \#(p_i, O(t_j)), \tag{6}$$

где $O(t_j)$ – входная функция перехода t_j.

Матричная теория сетей Петри позволяет решать задачи сохранения и достижимости, однако имеет ряд серьезных недостатков, связанных, прежде всего, с тем, что составная матрица изменений $D = D^+ + D^-$ не полностью отражает структуру сети. Кроме того, отсутствует информация о последовательности в векторе запуска сети Петри.

3.**Стохастические алгебры процессов** являются широко известной техникой моделирования, применяемой для функционального анализа параллельных систем. Такие системы описываются как множества сущностей либо агентов, выполняющих атомарные действия, которые используются для отображения поведения системы, синхронизации и коммуникации.

Алгебра процессов предоставляет формальную модель параллельной системы, обладающую как свойством абстрактности (внутренним поведением компонентов системы можно

пренебречь), так и свойством композитности (система может быть промоделирована в терминах взаимодействия подсистем).

Стохастические алгебры процессов – это расширение алгебр процессов, которое добавляет к модели квантификацию, делая ее применимой для анализа производительности. Суть расширения состоит в ассоциировании со случайной переменной, представляющей время, каждого действия.

3. Расчет производительности мобильной системы

Приведенные математические формализмы являются достаточно развитыми формальными инструментами моделирования систем. Они позволяют сделать как качественную, так и количественную оценку производительности программной системы. Однако, как уже было отмечено выше, существует необходимость оценки проектируемой системы в контексте языка UML, не выходя за рамки выбранной спецификации системы. Существует несколько возможностей для использования математических формализмов в рамках UML-модели.

Одним из методов является расширение нотаций UML с целью добавления информации о производительности. Рассмотрим применяемые подходы, основанные на этом методе.

Подход, базирующийся на диаграммах последовательности, а также прототип инструментального средства, представлен в работе [7]. Моделирование осуществляется с помощью так называемой анимированной диаграммы последовательности, позволяющей отслеживать события. Аналогичный подход представлен в работе Арифа и Шпайрса [8], где каркас модели используется для генерации моделирующих программ на базе диаграммы классов и диаграммы последовательности, а также некоторой случайной и статистической информации. Авторами используется структура, названная языком моделирования SimML, которая формирует имитационную программу по соответствующей UML-спецификации. Предлагаемый пакет построения UML-диаграмм позволяет пользователю нарисовать диаграммы классов и последовательностей, описать необходимую информацию и автоматически сформировать ориентированную на программный процесс имитационную модель. Имитационная программа формируется на языке программирования Java. Подход предлагает XML-преобразование исходной UML-модели для того, чтобы обеспечить одновременное хранение структурной информации и информации, необходимой для имитационного моделирования.

В [9, 10] предложено использование диаграмм кооперации с диаграммами состояний для всех возможных объектов в пределах модели. Таким образом охватываются все возможные варианты поведения системы. Основная идея заключается в том, чтобы преобразовать каждую диаграмму состояний, которая представляет объекты диаграммы взаимодействия в сеть Петри. Полученное множество сетей Петри может быть объединено в модель стохастической временной сети Петри, представляющей всю систему в целом, а именно обобщенную стохастическую сеть Петри. Кроме того, в [9] описываются некоторые предварительные идеи относительно получения моделей сетей массового обслуживания (СМО) из UML-спецификации. Диаграммы прецедентов используются для назначения параметров рабочей нагрузки и моделирования различных классов запросов системы. Диаграммы реализации используются для определения использования и

доступности системных ресурсов. Основная идея заключается в том, чтобы определить соответствие объединенных диаграмм размещения и компонентов с моделями СМО, отображающими компоненты и их связи с центрами обслуживания.

В работе [11] представлен метод получения модели в терминах стохастической алгебры процессов из спецификации UML. Спецификация ПО описывается посредством объединения диаграмм взаимодействия и диаграмм состояния всех взаимодействующих и вложенных в них объектов. На рис. 1 показан пример объединенной диаграммы «производитель–буфер–потребитель».

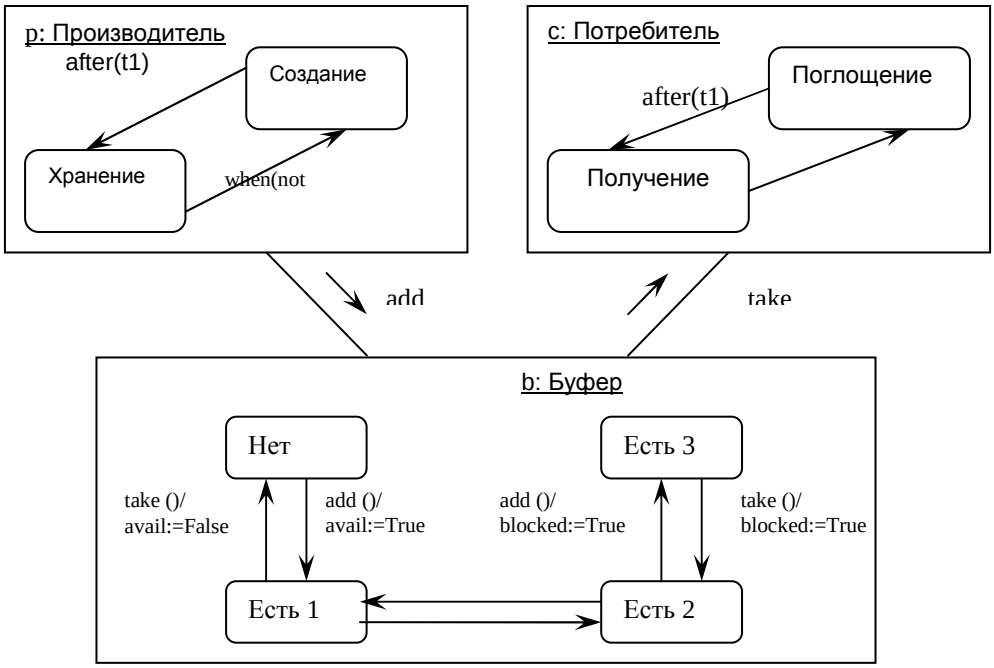

Рис. 1. Пример объединенной диаграммы

После создания такой модели получают описание в терминах стохастической алгебры процессов. Каждому объекту диаграммы взаимодействия ставится в соответствие так называемая PEPA-модель (Performance Evaluation Process Algebra), описанная в [12]. Например, у приведенной выше модели объект «Производитель» имеет два состояния и мы рассматриваем (моделируем) каждое из них как агент с соответствующими именами: «Producer» и «Blocked». Переход от «Producer» к «Blocked» требует выполнения действия создания, которое занимает время t1. В терминах PEPA это оценивается как $i/t1$. Переход от «Blocked» к «Producer» требует выполнения пассивного действия add. Действие определено как пассивное, так как скорость, с которой оно фактически выполняется, полностью определяется объектом «Буфер». Описание в терминах алгебры процессов выглядит следующим образом:

$$\mathrm{Pr}\,oducer \overset{def}{=} (produce, 1/t1).Blocked \ ; \tag{7}$$

$$Blocked \overset{def}{=} (add, \mathrm{T}).\mathrm{Pr}\,oducer \ . \tag{8}$$

Аналогичные выражения описывают объект «Потребитель»:

$$Consumer \overset{def}{=} (consume, 1/t2).Halted \ ; \tag{9}$$

$$Halted \overset{def}{=} (take, \mathrm{T}).Consumer \ . \tag{10}$$

“Буфер” является простым вычисляющим процессом, не позволяющим произвести действие “take “, если у него 0 элементов, и не позволяет произвести действие “add“, если их 3.

$$Buffer_0 \overset{def}{=} (add, 1/t3).Buffer_1 \ ; \tag{11}$$

$$Buffer_1 \overset{def}{=} (add, 1/t3).Buffer_2 + (take, 1/t4).Buffer_0 ; \tag{12}$$

$$Buffer_2 \overset{def}{=} (add, 1/t3).Buffer_3 + (take, 1/t4).Buffer_1 \ ; \tag{13}$$

$$Buffer_3 \overset{def}{=} (add, 1/t3).Buffer_2 \ . \tag{14}$$

Кроме того, в статье представлена попытка формирования Марковской цепи с непрерывным временем непосредственно из диаграмм UML. Ключевой момент здесь заключается в том, что в любой момент времени для каждого объекта на диаграмме взаимодействия существует только одно внутреннее состояние. Совокупность объектов в текущем состоянии называется маркировкой. Получение всех возможных маркировок возможно посредством проведения взаимодействий. Это позволяет построить соответствующую диаграмму переходов и затем Марковские цепи.

В работах [13, 14] представлены расширения UML как аннотации производительности (pa-UML) программных систем. Суть в том, что с помощью набора правил преобразования из pa-UML-диаграмм получают обобщенные стохастические сети Петри. Индексы производительности при этом определяются методами классического анализа.

Шаблон, позволяющий использовать UML-диаграммы при построении моделей производительности, представлен в работе [15]. Моделирование производительности происходит на основе определенной текстовой нотации, названной Performance Modelling Language и служащей для представления характеристик UML-модели, имеющих значение при оценке производительности. Полученная таким образом модель преобразовывается в СМО с моделированием распределения ресурсов.

Аннотации для представления производительности на UML-диаграммах представлены также в [16]. Работа описывает исследования шаблонов компонентного взаимодействия систем типа клиент-сервер на диаграммах классов и диаграммах кооперации. UML-диаграммы аннотируются с помощью XML-нотаций с параметрами, указывающими нагрузку и потребность в обслуживании. После этого получают модель массового обслуживания и проводят ее анализ с целью получения необходимых индексов производительности.

Модели, описывающие производительность системы в контексте UML, представлены также в работе [17], где предложена методология для получения сетевых моделей на базе моделей производительности из набора UML-диаграмм. Методология использует диаграммы прецедентов, диаграммы классов, последовательности и развертывания. Данный набор диаграмм интегрируется с множеством параметров, определяемых проектировщиком (архитектором) системы, в результате синтезируется исключительно точная модель производительности.

К вышеперечисленным методам получения модели производительности следует добавить методы, основанные на Software Performance Engineering методологии (SPE) [18].

Таким образом, на данном этапе развития программной инженерии существует множество методов получения производительности на ранних стадиях жизненного цикла ПО. Рассмотрим их с точки зрения полноты охвата UML-диаграмм и применяемых аналитических методов (табл.1) .

Таблица 1. Методы получения производительности

Название/авторы метода	**UML-диаграммы**	**Методы анализа**
SPE Вильямс и Смит	Sequence	Преобразование спецификации программного обеспечения (СПО) в модель выполнения
CLISSPE Менанске, Гома	UseCase	СМО
CHAM Бальзамо	Sequence	СМО
Мирандола, Кортелеса	UseCase, Class, Deploy, Interaction	СМО
Шаблон Client/Server Гома, Менанси	Class, Interaction	XML, СМО
Шаблон LQN	Interaction, Class, Sequence, UseCase, Deploy	СМО
SimML Ариф, Шпайрс	Class, Sequence	Java\XML ИМ
OPNET Мигель	Class	ИМ
Кинг, Пули	Interaction, Class, UseCase	SPN
Пули	Interaction	САП, цепи Маркова
Смит, Вильямс	Sequence	СМО

4. Выводы

Приведенные методы охватывают как полный жизненный цикл ПО, так и отдельные этапы проектирования. Как отмечено в [5], подходы, ориентированные на проектирование, предусматривают формальное моделирование поведения программы, основываясь на стохастических сетях Петри либо стохастических алгебрах процессов. Такие модели объединяют функциональные и нефункциональные характеристики и обеспечивают однозначное соответствие СПО и модели производительности.

Существуют также методы оценки производительности, направленные на решение данной задачи в рамках конкретных архитектур, таких как клиент-серверные системы. Однако они не были рассмотрены, так как выходят за рамки предмета исследования.

Дальнейшие исследования в области оценки производительности программных систем для мобильных устройств должны быть направлены на модели адаптации к мобильной среде. В этом

направлении наиболее перспективной считается модель сотрудничества с осведомленностью приложений друг о друге. Данная модель предусматривает наличие общего уровня адаптации, функции которого используются совместно всеми приложениями.

СПИСОК ЛИТЕРАТУРЫ

1. Сигорский В.П. Математический аппарат инженера. – Киев: Техніка, 1975. – С. 703.
2. Woodside C., Neilson J., Petriu S., Mjunidai S. Tlie Stochastic rendezvous network model for performance of synchronous client-server-like distributed software // IEEE Transaction on Computer. – 1995. – Vol. II. – P. 20–34.
3. Rolia J.A., Sevcik K.C. The Method of Layers // IEEE Transaction on Software Engineering. – 1995. – Vol. 21/8. – P. 682.
4. Merseguer Jose, Campos Javier Software Performance Modeling using UML and Petri nets. Dpto. de Informatica e Ingeniera de Sistemas Universidad de Zaragoza CMara de Luna, 1, 50018. – Zaragoza, Spain, 2003. – P. 25.
5. Balsamo and Marta Simponi On transforming UML models into performance models // Dip. Informatica. – Italy: Universita Ca Foscari di Venezia, 2000. – P. 6.
6. Питерсон Дж. Теория сетей Петри и моделирование систем: Пер. англ. – М.: Мир, 1984. – 264 с.
7. Pooley R., Kabajunga C. Simulation of UML Sequence Diagrams // Proc. of 14th UK Performance Engineering Workshop / Edinburgh R. Pooley and N. Thomas Eds., UK PEW ’98. – 1998. – July. – P. 14.
8. Arief L.B., Speirs N.A. A UML Tool for an Automatic Generation of Simulation Programs // Proc. of Second International Workshop on Software and Performance, WOSP2000. – Ottawa, Canada. – 2000. – September. – P. 12.
9. King P., Pooley R. Estimating the Performance of UML Models using Petri Nets // Private communication. – 1999. – P. 20.
10. King P., Pooley R. Using UML to Derive Stochastic Petri Net Models // Proc. of the Fifteenth UK Performance Engineering Workshop, Department of Computer Science, The University of Bristol / Davies N., Bradley J., editors, UKPEW ’99. – 1999. – July. – P. 13.
11. Pooley R. Using UML to Derive Stochastic Process Algebra Models // Proc. of XV UK Performance Engineering Workshop. – 1999. – P. 11.
12. Hillston Compositional Approach to Performance Modelling. – Cambridge University Press, 1996. – P. 170.
13. Merseguer J., Campos J., Mena E. Performance Evaluation for the design of Agent-based Systems: A Petri Net Approach // Proc. of Software Engineering and Petri Nets (SEPN 2000). – Aarhus, Denmark. – 2000. – June. – P. 20.
14. Merseguer J., Campos J., Mena E. A Pattern-Based Approach to Model Software Performance // Proc. of Second International Workshop on Software and Performance, WOSP2000. – Ottawa, Canada. – 2000. – September. – P. 7.
15. Kahkipuro P. UML based Performance Modeling Framework for Object-Oriented Distributed Systems // Proc. of Second International Conf. on the Unified Modeling Language, October 28–30. – USA, LNCS, Springer Verlag. – 1999. – Vol. 1723. – P. 17.
16. Gomaa H., Menasce D.A. Design and Performance Modeling of Component Interconnection Patterns for Distributed Software Architecture // Proc. of Second International Workshop on Software and Performance, WOSP2000. – Ottawa, Canada. – 2000. – September. – P. 126.
17. Cortellessa V., Mirandola R. Deriving a Queuing Network based Performance Model from UML Diagrams // Proc. of Second International Workshop on Software and Performance, WOSP2000. – Ottawa, Canada. – 2000. – September. – P. 12.
18. Smith C.U. Performance engineering of software systems. – The Sei Series in Software Engineering, Addison-Wesley, 1990. – P. 496.