УДК
(519.682.6+519.683.2):(004.414.2+00
4.057.2+004.412)

Согласование схем программных систем с диаграммами компонент UML и их сравнительная оценка

В. О. Мищенко
Харківській національний університет імені В. Н. Каразіна, Україна

Статья посвящена вопросам использования энергетических метрик моделей на этапе разработки проекта программной системы. Ранее технологии использования этих метрик развивались для применений на этапе написания исходных кодов в соответствии с проектными решениями. Энергетические метрики требуют составления схемы программной системы (СПС), по которой можно сгенерировать каркас будущей программы. В работе доказано, что класс всех СПС можно изоморфно вложить в класс диаграмм UML, использующих базовые компоненты. Разработан метод составления СПС для СПС и для UML модели, ориентированной на преимущественное использование компонент.

Ключевые слова: *программная система, схема, UML, диаграмма компонент, метрики качества, модель, разработка проекта, этап, кодирование.*

Статтю присвячено теорії використання енергетичних метрик моделей на етапі розробки проекту програмної системи. Раніше технології використання цих метрик розвивалися для застосування на етапі написання вихідних кодів згідно з проектними рішеннями. Енергетичні метрики вимагають складання схеми програмної системи (СПС), за якою можна згенерувати каркас майбутньої програми. У роботі доведено, що клас всіх СПС можна ізоморфно вкласти в клас діаграм UML, що використовують базові компоненти. Розроблено метод складання СПС для СПС і для UML моделі, орієнтованої на переважне використання компонент.

Ключові слова: *програмна система, схема, UML, діаграма компонент, метрики якості, модель, розробка проекту, етап, кодування.*

The article devoted to questions of the use of energetic metrics of models in the drafting of a software system. Previously, the technology of use these metrics developing for applications in the stage of writing the source code in accordance with design solutions was developed. Energetic metrics require building of the software system chart (SPC), with which you can generate a skeleton of the future program. It is proved that the class of all SPC can be isomorphically embedded in the class of diagrams of UML that use the basic components. The method for the preparation of the SPC for SPC and SPC for UML model, which focused on the preferential use of the components, was developed.

Key words: *software system, scheme, UML, component diagram, quality metrics, model, project development, stage, coding.*

1. Введение

Сфера применимости средств, обсуждаемых в данной работе – это проекты, сложность которых хорошо отражается в структуре создаваемой программной системы. Наглядное моделирование этой структуры необходимо при проектировании таких систем и при контроле над их разработкой. Отсюда вытекает актуальность наличия построителей схем программных систем (СПС) [1-3], способов оценки сложности таких систем, связанных с их использованием,

рисков и способов оценки более общих графических систем проектирования, таких как диаграммы компонент UML [4].

Цель статьи состояла в том, чтобы дать интерпретацию СПС в форме диаграмм компонент, открыв путь к построению СПС с помощью индустриальных моделереров UML, и распространить на определённые классы графических моделей программных систем энергетические метрики, позволяющие оценивать с определенных точек зрения надёжность, сложность и совершенство таких графических моделей.

2. Постановка задачи

Схемы программных систем могут выступать и как средство моделирования готового кода для его оценки методами энергетического анализа программ [2,3], и как средство моделирования систем на этапе проектирования, позволяющее, в принципе, автоматически сгенерировать «костяк» программной системы [5]. Однако широкий спрос на разработку инструментов, предназначенных исключительно для поддержки СПС, маловероятен. Предпочтительна специализация уже существующих систем поддержки компьютерной графики. Мы установим, что можно с успехом использовать современные UML моделереры, поскольку всякой СПС можно по детерминированному алгоритму придать вид системы диаграмм базовых компонент UML. Эти диаграммы будут иметь специальный вид. Однако мы разработаем корректные способы оценки примитивов энергетических метрик для несколько более общих графических моделей, которые включают и класс всех СПС, и системы довольно произвольных диаграмм компонент UML. В качестве примера простого приложения проведено сравнение сложности и совершенства исходного и приведенного к UML графических представлений СПС.

3. Вложение класса всех СПС в системы диаграмм UML

Способ графического изображения СПС, сложился в публикациях [2,3,5-8] как способ визуализации их теоретико-множественного определения [1]. Эти изображения назывём графическими СПС (ГСПС). Они используют популярные примитивы рисования и пиктограммы, характерные, например, для диаграмм потоков (но, вообще говоря, с совершенно другим смыслом). В специальных случаях схемы, на которых дано (возможно, без деталей) изображение СПС или её частей в целом, называем эскизами СПС, а детализированные изображения частей СПС - подсхемами.

Структурные модели исходных текстов программ (программных систем) или их части, разработанные на унифицированном языке – UML, будут здесь обсуждаться только в форме диаграмм этого языка. По форме они могут быть просто диаграммами или же кадрами, «вырезанными» из других диаграмм с добавлением подробностей [4].

Наше доказательство вложенности (с точностью до изоморфизма) класса СПС в класс диаграмм компонент UML 2.x поясним наглядно, поскольку не предполагаем детального знакомства читателя с СПС. Для этого воспроизведём существующее определение СПС как математической структуры, иллюстрируя это графическими примерами. Параллельно, опираясь на те же примеры, дадим описание способа построения эквивалентной модели UML. Описания элементов

СПС в первых 5 пунктах дословно соответствуют первоначальному определению СПС в [1], а основное содержание дальнейших уточняющих пунктов сложилось позже на опыте реального использования СПС [6-8].

1^0. Схема программной системы P представляется множеством *модулей*, которое распадается на непересекающиеся подмножества: I - *интерфейсных* и R - *реализующих* модулей.

На модулях системы P определены четыре бинарные отношения, описанные в пунктах 2^0-5^0.

2^0. Отношение «*имеет тело»* является частично определённой функцией на I со значениями в R. Эта функция разным интерфейсным модулям ставит в соответствие разные реализующие модули (тела) и обозначается префиксным символом b («тело»). В этих обозначениях bM - тело интерфейсного модуля M.

Подразумевается, что все модули СПС именованы. Многие языки разработки (например, C++ или Ада) предусматривают возможность переименования модулей, то есть явное снабжение их полезными синонимами. Все такие переименования (вместе с определениями модулей с помощью настройки шаблонов) относят к специальному интерфейсному модулю, который называется *остаточным*. Тела у него нет.

Средства графического изображения интерфейсных модулей и их тел иллюстрируют рис.1, 2.

На графических СПС (рис. 1) интерфейсные модули и их тела изображаются прямоугольниками. Тела надписаны словом body и вслед, если помещается, - именем своего интерфейсного модуля. Их можно не подписывать вовсе, но необходимо соединить со своим интерфейсным модулем линией, по возможности, прямой и короткой.

На эквивалентной UML все модули СПС должны моделироваться компонентами (базовые компоненты из пакета BasicComponents). Тела должны быть членами пакета с именем Body и иметь имена, совпадающие с именами тех интерфейсных модулей, телами которых они являются. Тела модулей находятся в отношении реализации со своими интерфейсными модулями так, что на диаграммах (рис. 2) соединяются с ними пунктирной стрелкой с полым треугольным наконечником, который упирается в интерфейсный модуль. Указание типа классификатора может быть любым (ключевое слово, значок).

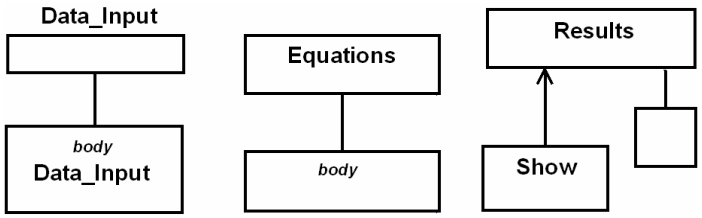

Рис. 1. Интерфейсные модули, их тела; родительский (Result) и дочерний (Show) модули

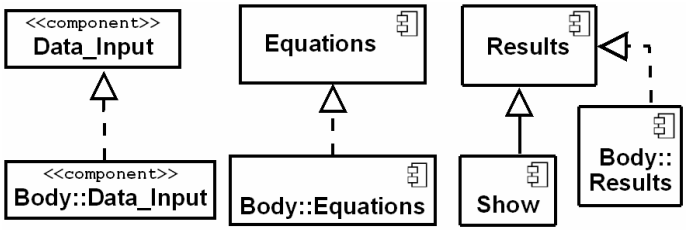

Рис. 2. UML модель, эквиалентная СПС, изображенной на рис. 1

Если СПС представлена несколькими эскизами и подсхемами, тела не обязательно указывать на них всех (на рис. 1, 2 не показано тело Show).

3^0. Отношение *«имеет родителя»* является частично определённой функцией p на I со значениями в I, причём соответствующее отображение не порождает зацикленных траекторий:

$$p^n M = M \Rightarrow n = 0 \quad , \tag{1}$$

где p читается как «родитель». Элементы прообраза $p^{-1}M'$ - «наследники» M'.

На графических СПС связь с родителем (если есть) изображается обычной стрелкой. Можно указывать простое собственное имя дочернего модуля (полное имя состоит из одного или нескольких префиксов - имен модулей – предков).

На UML дочерний модуль моделируется специализацией родительского модуля, что на диаграммах (рис. 2) изображается сплошной линией-стрелкой с треугольным полым наконечником, который упирается в родительский модуль.

4^0. Отношение *«имеет надмодуль»* является частично определённой функцией s на R со значениями в R, причём соответствующее отображение (модуль $\mapsto$ надмодуль) не порождает зацикленных траекторий:

$$s^n B = B \Rightarrow n = 0 . \tag{2}$$

Тело не должно иметь надмодулей:

$$\neg \exists B, M : sbM = B . \tag{3}$$

Всякий элемент B из прообраза $s^{-1}B'$ называется субмодулем модуля B'.

Пример графической СПС, имеющей субмодуль, - на рис. 3. В полное имя субмодуля добавляются префиксы - имена надмодулей данного субмодуля.

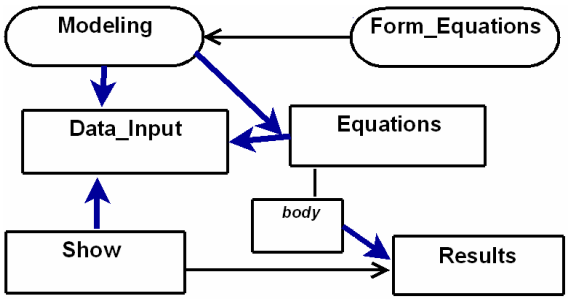

Рис. 3. Субмодуль (Form_Equations), самостоятельный реализующий модуль (Modeling) и зависимости (например, Show зависит от Data_Input)

В эквивалентной UML модели модуль, обладающий субмодулями, представлен в том же пространстве имён также пакетом с таким же, как у него, именем. Членами этого пакета являются его непосредственные субмодули и пакеты их субмодулей. Отношение субмодуля с его надмодулем моделируется отношением агрегативной принадлежности, которое на диаграммах изображается сплошной линией-стрелкой с полым ромбовидным наконечником, который упирается в надмодуль (рис. 4). Если нет конфликтов имён, то указывается прямое имя субмодуля, иначе – квалифицированное.

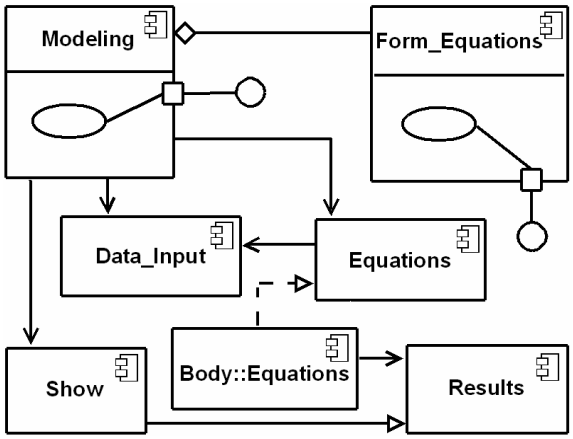

Рис.4. Эквивалентная диаграмма фрагмента СПС, изображенного на рис. 3

Реализующий модуль, который не является ничьим телом, на ГСПС изображается прямоугольником со скруглёнными краями или овалом (рис. 3).

В UML моделях такой реализующий модуль представляется компонентом, имеющим безымянный порт поведения, типизованный одним поддерживаемым интерфейсом, имя которого, как правило, совпадает с именем модуля и может умалчиваться. Рекомендуется использовать символ интерфейса – шарнир (как на рис. 4), но допустим и альтернативный способ изображения в UML с помощью классификатора и пунктирной стрелки с треугольным полым наконечником.

5^0. Отношение *«зависит от»* может рассматриваться как частично определённая многозначная функция m на P со значениями в I. Обычно удобнее смотреть на неё как на всюду определённую функцию со значениями в множестве подмножеств I. Три отношения: транзитивное замыкание отношения «имеет родителя», произведение транзитивного замыкания отношения «*имеет надмодуль*» на обратное к отношению «имеет тело» и отношение *«зависит от»*, - не должны пересекаться.

На ГСПС отношение *«зависит от»* изображает «толстая» стрелка (рис. 3).

На эквивалентной диаграмме зависимость модуля СПС *A* от модуля *B* трактуется как однонаправленная ассоциация, причём *B* достижимо для *A*. Это изображается (рис. 4) сплошной линией-стрелкой с «раскрытым» наконечником.

6^0. Любая реализация всякого модуля (не обязательно окончательная или сколько-то полная) определяет список (не обязательно линейный) программных символов (ПСим). Он называется *собственным блоком* этого модуля:

$$< X > \text{ - собственный блок модуля } X. \qquad (4)$$

С каждым модулем связывается непустое множество попарно не пересекающихся подсписков ПСим собственного блока, которые называются *внутренними блоками*. *Блок* модуля - это его собственный или внутренний блок.

Необходимо правило, позволяющее выделять внутренние блоки в собственном блоке модуля. Эти блоки называются *явными.* Дополнение объединения всех явных внутренних блоков модуля, если оно не пусто, считается тоже внутренним блоком, который называется тогда *остаточным* блоком данного модуля. Его имя Rest; если необходимо - Rest (*имя_модуля*).

Блоки, подобно модулям, делятся на *интерфейсные* и *реализующие*. В числе последних могут быть тела интерфейсных блоков, а, если это не так, то блок называется *самостоятельным*.

7^0. На множестве внутренних интерфейсных блоков каждого модуля X задана частично определённая функция b_X со значениями в множестве внутренних блоков всех модулей схемы P. Эта функция ставит в соответствие блоку X_i его *тело* – некоторый реализующий блок $b_X(X_i)$. Разным блокам в качестве тел соответствуют разные блоки. Блоки, связанные с интерфейсным модулем, могут быть только интерфейсными. Телом интерфейсного блока M_i модуля $M \in I$ может служить только блок, который соответствует телу этого модуля ($b_M(M_i) \subseteq < bM >$) или блок субмодуля тела (собственный или внутренний), причём не только субмодуля в буквальном смысле, но и субмодуля субмдуля и т.п. Если внутренний блок связан с $B \in R$, то тело этого блока может быть связано либо с B, либо с его субмодулем $B' \in s^{-1}B \subset R$, либо с субмодулем субмодуля и т.п.

Остаточного блока у остаточного модуля нет.

На ГСПС блоки изображаются овалами (полными или срезанными на одном конце) или прямоугольниками (полными или срезанными) со скруглёнными краями или параллелограммами. Они могут быть вытянуты как горизонтально, так и вертикально, надпись имени – внутри. Блоки реализующих модулей и приватные блоки интерфейсных модулей изображаются строго внутри своих модулей. Изображения публичных блоков должны иметь общую часть границы с границей модуля (отсюда их срезка). Остаточный блок представляет та часть изображения модуля, которая не относится к изображению явных внутренних блоков, она, как было сказано, именуется Rest. На эскизах и некоторых подсхемах СПС внутренние блоки можно опускать, как на рис. 1-3.

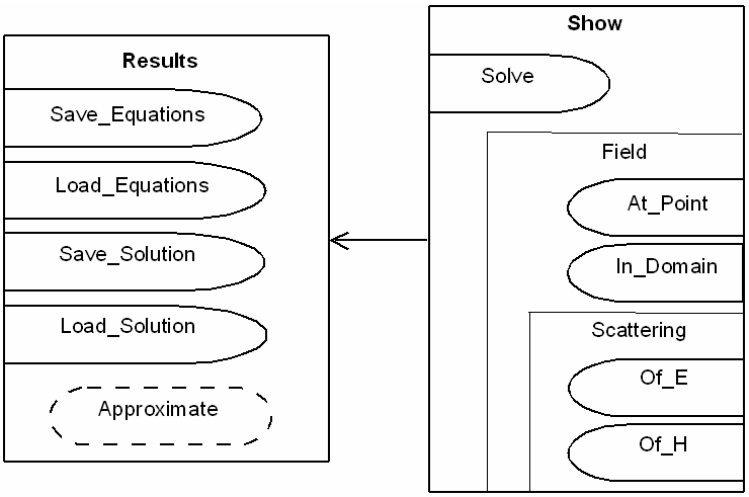

Рис. 5. Сопоставление блоков модулям Results и Show

На эквивалентных UML диаграммах явные внутренние блоки нужно представлять с помощью операций самого компонента или его классов (рис.6).

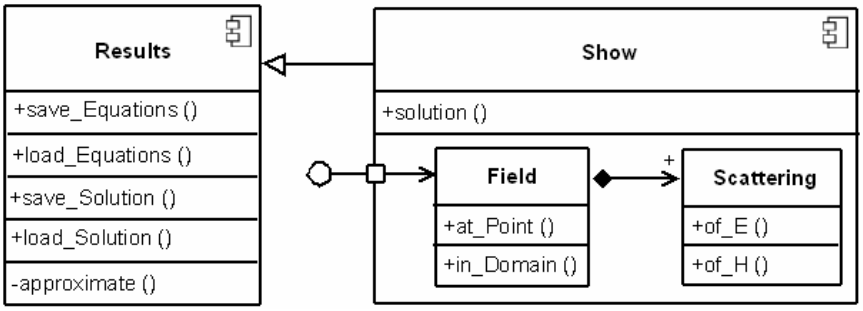

Рис.6. Модель фрагмента ПС, изображенной на рис. 5

8^0. Собственные блоки разных модулей, как списки экземпляров ПСим, не пересекаются (хотя не исключено, что могут быть, например, равны между собой с точностью до ПСим, представляющих имя).

9^0. Дальнейшие правила нужны для подсчёта энергетических мер, но рассматриваемый в них вопрос о статусе блоков, важен также и для определения полного состава блоков, связанных с модулем, а также природы субмодулей.

Определить *статус блоков* СПС – это разбить множество внутренних блоков модуля СПС на непересекающиеся *группы*, каждая из которых помечена, как *автономная*, *свободная, структурная* или *усложнённая* (смысл в том, что для каждого из статусов применяется своя формула расчёта спецификационной энергии). При этом автономная группа, если она есть, то только одна. Ей принадлежит остаточный блок модуля, если он есть. *Интерфейсная группа* – это группа, содержащая только интерфейсные блоки и интерфейсные группы. Иначе группа содержит тела всех блоков некоторой интерфейсной группы (и дополнительно может содержать интерфейсные блоки, группы, их тела и самостоятельные реализующие блоки). Такая группа считается *телом*

соответствующей интерфейсной группы. Интерфейсная группа не может иметь более одного тела, и разные группы - одинаковые тела.

Язык программной реализации данной СПС может предусматривать конструкции, оформляющие группировку блоков (примерами служат классы в объектно-ориентированном программировании и мониторы в параллельном программировании). Эта конструкция за вычетом группируемых блоков и внутренних групп образует блок, который включается в автономную группу модуля и называется *остатком* группы (а саму группу называем *обязательной)*. Это правило применяется рекурсивно, раз группа может содержать подгруппы. Остаток интерфейсной группы - блок интерфейсный, а остаток реализующей – реализующим блоком и телом остатка соответствующей интерфейсной группы.

Часть блоков модуля может не охватываться автономной или обязательными группами. Это - *свободные* блоки, составляющие свободную группу. Имея в виду проблему подсчёта спецификационной энергии модулей [9], группы, как правило, полагают структурными. Однако, если какая-то насчитывают более 8 явных блоков, то нужно проанализировать альтернативу – дать ей статус свободной. Свободные блоки должны быть свободной группой или присоединяться к автономной. Что именно – зависит от специфики проекта.

Остаточная группа обозначается как - Auton(*имя_модуля*) или в понятном контексте – Auton. На схемах и диаграммах это имя не пишется. Аналогично с именами остатков обязательных групп - Rest(Имя_Группы) или Rest.

В составе субмодулей модуля допускаются (если допускает язык реализации) такие субмодули, собственные блоки которых суть тела интерфейсных групп. Эти группы должны принадлежать надмодулю (или надмодулю надмодуля и т.п.) или (если модуль – тело модуля) интерфейсному модулю тела. При этом, если речь идёт о вложенных друг в друга интерфейсных группах, то субмодуль с телом внутренней группы может иметь надмодулем только модуль с телом непосредственно объемлющей группы. Субмодуль, яувляющийся телом группы, может (если допускает язык реализации) иметь собственные субмодули, собственные блоки которых – реализующие блоки данной группы.

Статус усложненных придаётся группам эксклюзивно (пример - группы блоков, связанных между собой сложной взаимной рекурсией).

На ГСПС группировка блоков фиксируется их обводкой тонкой линией по форме четырехугольника. Группировки на основе разных конструкций дифференцируются использованием четырёхугольников разных типов, начиная с прямоугольника (например, для языка Ада используются прямоугольники, параллелограммы и трапеции [5]). Изображение тела группы сохраняет форму интерфейсной группы, исключая субмодуль: он для всех тел прямоуголен.

Тела блоков, изображенных в интерфейсном модуле, в теле этого модуля можно без нужды не изображать. Если такие блоки составляют группу, то её тело можно в теле модуля не изображать.

Интерфейсный блок из реализующего модуля, который имеет тело, можно без нужды не изображать. Аналогично для групп.

Отметим, что приватные в целом группы изображаются с помощью четырехугольника, помещенного целиком внутрь изображения модуля. Публичные в целом группы обязаны иметь с модулем общую часть границы.

На эквивалентных диаграммах UML обязательные группы представляются классами. Из альтернативных способов их изображения на диаграммах предпочтителен (ради схожести с графическими СПС) стиль «прозрачного ящика». В этом случае группа, если только она не является единственной, представляется внутри классификатора компонента классификатором класса. Отношения между группами, вытекающие из вложенности соответствующих им конструкций языка разработки, моделируются с помощью композитных ассоциаций, они изображаются стрелками с ромбовидным закрашенным наконечником. Блоки внутри групп описываются в слотах классификаторов этих групп как операции (с подходящей назначению диаграммы степенью подробности). При необходимости на общей диаграмме можно ограничиться лишь демонстрацией всех групп с помощью классификаторов классов, а их внутреннюю структуру показать на отдельных диаграммах-кадрах. Изображения субмодулей, собственные блоки которых суть тела блоков (а не тела групп!) необходимо снабжать портами поведения. Однако при наличии таких групп, а, тем более, нескольких видов групп (соответствующих разным конструкциям языка реализации) разницу этих видов стоит акцентировать. Пригодились бы стереотипы, однако стандартные не подходят. Проще всего предусмотреть в модели, что соответствующие классы являются специализациями разных абстрактных классов с характерными именами (*Task*, *Monitor* и т.п.).

Способы изображения групп блоков см. на рис. 5, 6. В модуле Result все блоки свободны. В модуле Show есть обязательные группы: Field и вложенная в неё Scattering. Обе структурные. Блок Solve входит в группу Auton(Show) вместе с Rest(Field) и Rest(Skattering). Для того, чтобы подчеркнуть публичность части Field компонента Show, на диаграмме показано, что она реализует интерфейс, а публичный статус Scattering вытекает из достижимости для Field и публичности соответствующего конца ассоциации.

10^0. Языки реализации программных систем, как правило, располагают средствами высокого уровня для использования заготовок программного кода (называемых также шаблонами или настраиваемыми модулями). Самостоятельные модули СПС, то есть реализующие модули, которые не являются ни телами, ни субмодулями, плюс все интерфейсные модули, могут иметь признак заготовки, но при расчёте энергетических мер обрабатываются аналогично обычным модулям. То же самое – для групп и блоков. Изображения тел заготовок от изображений обычных тел не отличаются.

Если модуль-заготовка используется (по другому говорят – настраивается или привязывается) на том же программном уровне, к которому относятся самостоятельные модули СПС, то его следует относить к остаточному модулю данной СПС в качестве интерфейсного блока. У такого блока нет тела. Если заготовка используется на верхнем уровне внутри модуля, то относится к остаточному блоку этого модуля, а, если на верхнем уровне обязательной группы, то – к её остатку.

Если модуль M зависит от результата настройки заготовки T, то, M в СПС считается зависящим от T (а не от Rest, как можно было бы подумать):

$$T \in m(M) \quad . \tag{5}$$

На графических СПС внешняя граница изображения настраиваемого модуля (группы, блока) даётся пунктиром. Для параметров настройки (в точном их описании или, хотя бы, ради информации о количестве) дорисовывается небольшой прямоугольник («монтажная кабина»). Модуль, который является результатом настройки, изображается или учитывается количественно в остаточном модуле. Результаты настроек внутри модулей относятся к остаточному блоку или остатку соответствующей группы и учитываются при подсчёте энергетических мер. Поскольку остаточный модуль, судя по известным примерам реальных систем, обычно не оказывает заметного влияния на расчёт энергетических мер, то он может на графической схеме не изображаться. Но принадлежащий ему именованный результат настройки рекомендуется указывать в виде маленького значка (по форме напоминающего изображение модуля) в качестве транзитного пункта на стрелке зависимости модулей. Например, в обозначениях (5) результат настройки, как транзитный пункт, размещается на стрелке ассоциации, ведущей от *M* к *T*. Можно все стрелки, идущие от транзитного пункта к заготовке, при их изображении совмещать.

При моделировании с помощью UML используется хорошо разработанный там аппарат образцов (templates) и их привязки (binding). Образцы модулей и групп изображаются как модули и группы, но их верхняя часть частично (по возможности справа) перекрывается пунктирным прямоугольником, где располагаются параметры. Для образцов блоков и их привязок (то есть, результатов настройки) подобные изображения не применяются. Соответствующая информация располагается в строке описания операции. Удобно, если остаточный модуль не показывается, привязки образцов на уровне модулей изображать с помощью новых компонент со стрелками

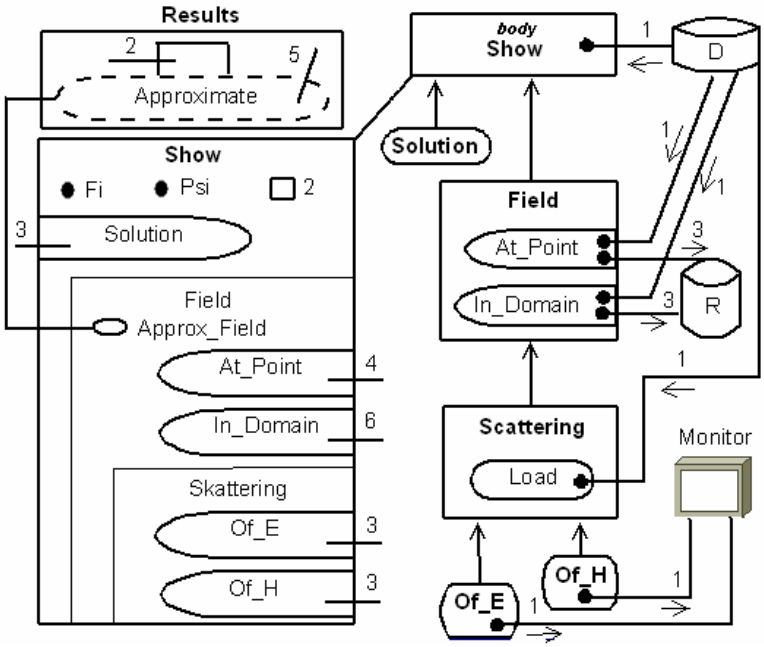

Рис. 7. Пример схемы с заготовкой и данными для определения числа формальных параметров блоков

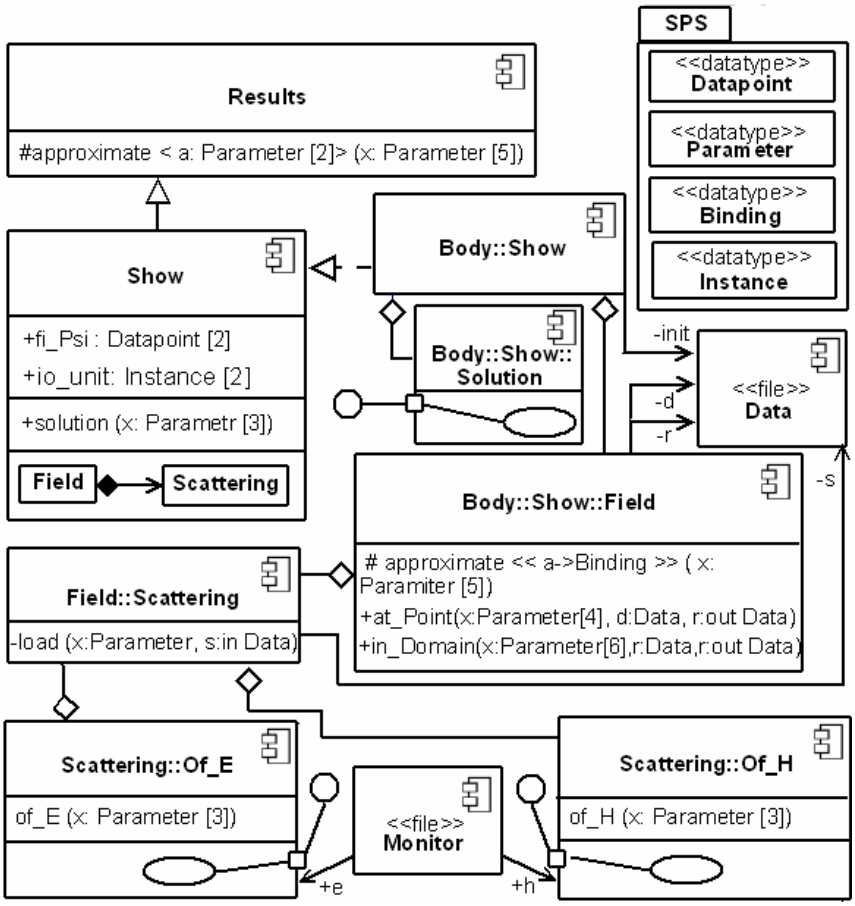

Рис.8. UML модель подсистемы, СПС которой изображена на рис.7

зависимости (пунктир, наконечник треугольный полый), которым приданы строки, определяющие связывание. Иначе новые компоненты изображаются внутри компонента - остаточного модуля, а строки, определяющие связывание, - внутри новых компонент. Примеры на ГСПС и диаграмме находим на рис. 7, 8.

11^0. На определённом этапе разработки на ГСПС необходимо показать *формальнее параметры* блоков. Предполагается, что вся информация о них содержится в полной СПС, т.е. исходных кодах. ГСПС обычно содержат частичную информацию, как результат текущих проектных решений.

Формальные параметры блока это каналы для информационных обменов блока с его окружением однотипными или, по крайней мере, хорошо между собой согласованными данными. Хотя для целей разработки и контроля программ на ГСПС желательно было бы указывать виды и типы всех параметров, для оценки энергетических метрик достаточно знать их количества.

У интерфейсного блока и его тела число параметров одинаково.

По определению, блоки остаточного модуля имеют по одному формальному (входному) параметру (тип которого объединяет значения - имена модулей).

Множество формальных параметров блока, который не входит в остаточный модуль, слагается, в общем случае, из трёх непересекающихся множеств:

множество *параметров настройки* (если это настраиваемый блок или остаточный блок настраиваемого модуля или остаток настраиваемой группы),

множество *параметров обращения* к блоку,

множество *квазиобъектов* ввода-вывода.

Число формальных параметров блока – сумма числа элементов этих множеств (причём число квазиобъектов ввода-вывода обычно удаётся только оценить).

Параметр обращения, если речь идёт об остаточном блоке реализующего модуля, который не является телом, или о внутреннем группируемом блоке, имеет смысл обычного формального параметра, какие есть у процедур и функций (результат функции трактуется как выходной формальный параметр). Для остаточного блока интерфейсного модуля или остатка группы смысл параметра обращения иной. Такой параметр, по определению, входной, и он является либо публичным изменяемым объектом данного блока (то есть, видимой для обращения извне постоянной, переменной) либо публичной настройкой заготовки. Типами таких параметров служат типы объектов (если это объекты) или имена используемых заготовок.

Квазиобъекты ввода-вывода - это абстракции однотипных (часто в неформальном смысле) или очень тесно взаимосвязанных между собой данных, направляемых в файлы или из них получаемых. При общем проектировании и разработке спецификаций модулей состав квазиобъектов ввода-вывода блоков может определяться по формальным признакам в соответствии с методом разработки. Если исходные тексты даны без документации, то сами квазиобъекты чётко не определяются, но их количество легко оценить [2].

На ГСПС наличие параметров настройки и параметров обращения символизируют небольшие стрелки, пересекающие границу изображения модуля, блока или пристройки к заготовке (если речь идёт о параметрах её настройки). Естественно, что направленность такой стрелки подсказывает направление передачи (привязки) параметра, а, если это модифицируемый параметр, то стрелка должна быть двусторонней. Отдельные параметры обращения (настройки) могут символизировать залитые (пустые) узелки, изображаемые на конце, противоположном наконечнику, или на пересечении стрелки с границей элемента схемы, к которому относится данный параметр. Возле стрелки можно указывать название и тип параметра. Но можно и объединить все параметры одного вида и возле стрелки указать их число. Можно даже, опуская у стрелки наконечник (наконечники), объединить вообще все параметры всех видов, указав возле оставшейся от стрелки чёрточки общее количество этих параметров. Напомним, что параметры блоков остаточного модуля СПС можно не указывать явно, поскольку относительно его блоков и его изображения на СПС действуют описанные выше специальные соглашения. При наличии у реализующего блока квазиобъектов ввода-вывода, от этого блока должна идти линия к изображению файла, помещенного на том же поле схемы, где располагаются изображения модулей. Рядом с (почти) прямолинейными участками этой линии изображаются (почти) коллинеарные ей стрелки, которые символизируют отдельные квазиобъекты или их совокупности аналогично параметрам обращения. При желании изобразить отдельные квазиобъекты с помощью залитых узелков, желательно, чтобы их форма явно отличалась от формы узелков параметров обращения (например, если те были круглыми, то

эти могут быть ромбиками). В качестве изображений файлов удобно использовать пиктограммы внешних устройств или носителей данных.

На UML диаграммах параметры обращения (не считая блоков упомянутого только что остаточного модуля) изображаются согласно правилам для указания параметров операций, а также изображения атрибутов (это может понадобиться для остаточного блока модуля или остатков групп). Файлы нужно изображать компонентами с использованием допустимых стереотипов и пиктограмм.

Способы изображения различных параметров представлены на на рис. 7, 8.

Отметим, что для целей энергетического анализа нет необходимости указывать, какой из самостоятельных реализующих модулей (или блоков интерфейсных модулей) используется для запуска программы или отдельных частей системы на выполнение. Поэтому на схемах и диаграммах, используемых в проектировании, это при необходимости делается с помощью дополнительных элементов изображения. На ГСПС принято обводить границы запускаемых модулей двойной или утолщенной линией. На эквивалентных UML диаграммах нужно именовать интерфейсы запускаемых модулей предопределёнными в проекте именами Main, Run, Start, Test (при необходимости с нумерацией, напр., Test_2). На рис. 3, 4 такие примеры имеем.

4. Схемы программных систем для СПС и диаграмм UML

Пусть графическая модель проектируемой системы представляет собой физически множество чертежей, каждый из которых является нагруженным многокомпонентным в общем случае, графом. Определим, что в такой системе отдельные чертежи являются интерфейсными модулями, а связные компоненты графов - обязательными группами. Вершины объявим блоками, а концы рёбер - формальными параметрами блоков, в которые они упираются. Пусть объект моделирования является общим для двух чертежей A, B. Тогда B определяется как дочерний (сыновний или наследниик) чертежа A в одном из двух случаев. Или это обстоятельство объявлено средствами данного графического языка моделирования. Или же граф чертежа B является подмножеством графа чертежа A, а система нагрузки соответствующей части A является подсистемой нагрузки B. При этом A может быть, а может не быть чьим-то наследником. Дочерний модуль B может иметь собственных наследников по любому из правил. Также, если на исходном чертеже один или несколько узлов представлены уточняющими графами на отдельных чертёжах, то такие чертежи являются дочерними по отношению к исходному чертежу.

Эта схема непосредственно применима к СПС, причём эскизы рассматриваются в качестве корневых модулей, а подсхемы – в качестве наследников, предполагая, что СПС в целом не нарушает требований раздела 3.

Для диаграмм компонет в UML дочерними определим модули, которые оформлены как диаграммы – кадры из других диаграмм, то есть имеют рамку и заглавие, поясняющее, в частности, пространство имён, представленное кадром. Такое пространство должно быть именованным элементом, представленным подходящим изображением на одной или нескольких других диаграммах. Предположим (и это обеспечено, если диаграмма эквивалентна СПС), что из этих диаграмм можно выделить одну (главную для данного именованного

элемента) диаграмму. Эта диаграмма считается родительским модулем рассматриваемой диаграммы. Кадр *B* кадра *A* – это дочерняя диаграмма *A*.

Перейдём к зависимостям в смысле СПС. Проблемы здесь общие для всех рассматриваемых средств графического моделирования, включая ГСПС, но будет понятнее, если решения излагать на примере, скажем, UML. Требуется, чтобы на каждой из системы диаграмм UML было выделено множество именованных элементов, которые являются ключевыми с точки зрения смысла именно этой диаграммы (в частности, потребуем, чтобы такие множества от разных диаграмм не пересекались). Тогда, если на какой-либо диаграмме имеется определение ключевого элемента другой диаграммы, то первую из них считаем кандидатом на зависимость от второй. Но не исключено, что отношение «кандидат на зависимость» может не удовлетворять требованиям, ограничениям СПС. Возможны циклы (например: диаграмма *A* зависит от *B*, а та, в свою очередь, - от *A*). Другой дефект – многозначность выбора «от кого зависеть» при фиксированном ключе. Рассмотрим все варианты решения последней проблемы, при которых можно ликвидировать циклы минимальным числом разрезов. Увеличим их число, рассмотрев все способы реализации минимальных разрезов. Тогда отношение «зависеть от» определяется как вариант, максимизирующий работу программирования. Это определение имеет теоретический смысл, и предполагает назначать процедуру выбора на случай неоднозначного решения.

Поэтому обсудим другие варианты. Оказалось, что в практике СПС среди проблем доминируют простейшие циклы (см. пример выше). Простой выход – сливать взаимно зацикленные диаграммы. Рассмотрение примеров показало, что тогда обычно происходит слияние вообще всех диаграмм в один модуль. Был испытан модифицированный метод. А именно, требуется выбрать вариант сокращения множеств ключевых элементов так, чтобы число слившихся после этого модулей было минимально, а число пар в графике отношения зависимости - максимально. Разумеется, задачу следовало бы уточнить, указав условия согласования оптимумов (например, в смысле Парето). Но на практике оказалось, что несколько вариантов сокращения множеств ключевых элементов напрашивается по смыслу, как предпочтительные. После слияния диаграмм для немногих оставшихся из них возможен перебор вариантов, реализуемый обычно даже вручную (и мгновенно – на компьютере). Этот перебор позволяет найти единственный оптимум. Ручной перебор оказывался возможен благодаря такой трактовке задачи: вначале выделить случаи минимального слияния диаграмм и отобрать те, где можно добиться максимального числа зависимостей. Таких вариантов всегда оказывался один или обозримое множество (в последнем случае решение окончательно отбиралось, как наиболее приемлемое по смыслу). Для еще большего «противодействия» эффекту слияний можно использовать градацию «силы» (важности) вхождения ключевых элементов в «чужие» диаграммы. Например, если этот элемент – узел на «своей» диаграмме, а на «чужой» изображен подключённым с помощью ребра, то есть, некоторого отношения, то чем это отношение сильнее, тем сильнее и вхождение элемента в чужую диаграмму. Например, если ключевой элемент *B* на диаграмме *A* доступен другим узлам только вследствие ассоциаций или является их специализацией, а ключевой элемент *A* является обобщением ключевого элемента на диаграмме *B*, то полагаем, что *B* зависит именно от *A*.

5. Анализ примеров

Графические СПС использовалось в [6-8] при подсчете числовых атрибутов метрик качества ПО и в связи с проектированием ПО. Примеры, представленные на рис. 1-8 недалеки от реальных СПС небольших проектов. Они, в первую очередь, служат целям демонстрации, и для сравнений мы привлекли ещё модели приложения из области компьютерного моделирования дифракционных процессов. По нашим правилам однозначно составляются диаграммы, эквивалентные опубликованным СПС указанных проектов. Это позволило сравнить работу составления графических СПС и эквивалентных диаграмм UML: см. табл. 1,2, где ЭД означает «эквивалентная диаграмма».

Размер продукции, *длина,* по традиции, воспринятой энергетическим анализом от науки Холстеда [9], - это общее число использованных в ней ПСим, а *алфавит* – их количество без учёта повторений. Объём по Холстеду равен [9]:

$$Объём = длина \times \log_2 (алфавит) \quad . \qquad (6)$$

Табл. 1. Сравнение параметров двух методов графического представления простых СПС

СПС	из [5]	из [8], рис.2	with Ga-hov A.V.	из [2], рис.7.1	с Боро-винским А.В.	из [6], рис.1
Алфавит ГСПС	15	18	16	22	26	25
Размер ГСПС	29	35	67	73	75	142
Объём ГСПС	118.5	151.3	279.4	334.7	360.6	675.2
Алфавит ЭД UML	16	21	20	26	30	34
Размер ЭД UML	51	57	107	109	107	262
Объём ЭД UML	212.7	257.8	477.2	524.0	525.0	1332.9

В табл.1 приведены объёмы ГСПС и эквивалентных диаграмм UML, и, как легко убедиться, объём эквивалентной диаграммы в среднем в 1.7 больше объёма ГСПС. Приняв знаменитую гипотезу Холстеда, можно полагать, что примерно в такое же число раз возрастает риск технической ошибки при составлении вместо ГСПС эквивалентной UML диаграммы.

В случае простых одномодульных СПС работа по Холстеду пропорциональна отношению квадрата объёма к потенциальному объёму V^*, который должен отражать свойства спецификаций (а не реализации). Последний, как бы он ни определялся, одинаков для графической СПС и эквивалентной диаграммы СПС. Следовательно, усреднив данные отношения, получим 2.9 – оценку того во сколько раз при прочих равных условиях работа составления эквивалентной диаграммы превышает работу составления ГСПС.

Для многомодульных программ оценка соотношения работ не так наглядна, поскольку вместо холстедовских объёмов модулей необходимо рассматривать [1,2] уточняющие их *объёмы разработки* модулей, определять по ним работу кодирования (в данном случае – рисования) всех модулей и только потом переходить к суммированию и сравнению. В этом случае потенциальные объёмы (при делении сумм) не сокращаются.

Отметим, что для рассматриваемых графических языков на основании общих определений [1,3]

$$Объём\ разработ. = Объём + Объём_1 + \dots Объём_k \ , \qquad (7)$$

где *Объём* – холстедовский объём данного модуля (6);

$Объём_1$, … $Объём_k$ – объёмы всех тех модулей СПС графической модели, от которых зависит данный модуль.

Потенциальные объёмы модулей находятся суммированием данной величины по блокам модуля, а для блока

$$V^* = \left(2+\eta_2^*\right) log_2\left(2+\eta_2^*\right) \ , \qquad (8)$$

где η_2^* - стандартное обозначение числа формальных параметров блока, определение которого воспроизводилось выше (в пункте 11^0 из раздела 3).

Табл. 2. Сравнение двух методов графического представления многомодульных СПС

СПС	из [7]		из данной статьи			
модуль	SLAE(рис .5.1)	DSM(рис.5 .2)	рис.1, 2	рис.3, 4	рис.5, 6	рис.7, 8
Алфавит ГСПС	34	34	8	13	19	37
Размер ГСПС	75	68	18	23	29	95
Объём ГСПС	387.7	739.3	59.8	606.1	127.4	556.5
Алфав.граф.UML	39	43	12	17	25	53
ПСим граф.UML	103	135	28	33	44	174
Объём разработ.	544.4	1276.9	106.6	1169.4	209.2	1100.0
Потенц. объём	124.4	148.5	33.25	67.6	6.75	91.3
A_{UML}/A_{SPS}	2.25		3.56			

В табл. 2 среднее отношение холстедовских объёмов модулей составляет 1.75, подкрепляя наблюдения, основанные на предыдущей таблице. В табл.2 учитывается, что эскиз (диаграмма) DSM зависит от эскиза (диаграммы) SLAE. Поэтому, подсчитав объёмы (6), можно рассчитать, следуя (7), объём разработки

$W_{DSM} = Объём_{SLAE} + Объём_{DSM}$ = 387.7+351.6 = 739.3 (для эскиза ГСПС) .

Объёмы разработки для диаграмм рассчитаны так же и приведены в табл. 2.

Взяв потенциальные объёмы из той же таблицы, получим

$$A_{ГСПС} = A_{DSM} + A_{SLAE} = 739.3^2/148.5 + 387.7^2/124.4 = 3680.6+1208.3 = 4888.9$$

Аналогично подсчитывается по данным таблицы работа кодирования диаграмм UML. Результаты сравнения, приведенные табл. 2, подтверждают для малых проектов гипотезу о том, что сложность разработки СПС в форме диаграмм UML выше, чем в форме ГСПС (ориентировачно раза в 3).

Наш подход к энергетическим оценкам графической продукции проектных работ в форме уточнённых правил для диаграмм классов и компонент UML (рассмотренных более общо, независимо от применения для представления

СПС) испытывался на адекватность определений в [10]. Способ определения длины и алфавита сомнений в согласованности с холстедовской традицией по данным работы [10] не вызывают. Однако важный для энергетического анализа вопрос о «сбалансированности в среднем» работы кодирования и спецификационной энергии получил только частичный ответ. Возможно, в особенности для диаграмм компонент, что определения, связанные с архитектурой рассмотренных диаграмм и формальными параметрами блоков, пока не окончательны, нуждаются в усовершенствованиях. Однако возможно другое, что баланс есть, но для диаграмм, в особенности для диаграмм компонент, существенное превышение энергии над работой является их нормальной спецификой. Иначе говоря, оценка уровня таких языков должна быть не прядка единицы (как предполагалось в [10] по опыту традиционных языков программирования) а выше.

Последнее предположение подкрепляется и на выборке диаграмм, рассмотренных в настоящей работе. Например, по данным табл. 2, используя те же формулы подсчёта спецификационной энергии, что и в [10], мы легко обнаруживаем для UML модели [4] превышение энергии над работой примерно в 400 раз. Для модели, определяемой всеми примерами UML диаграмм нашей работы, имеем превышение почти в 12 раз.

5. Заключение

СПС – модель архитектуры программной системы. Она отражает разные её уровни, позволяя на ранних этапах проектирования дать оценку сложности проекта в форме спецификационной энергии и начать разработку исходных текстов, а на последующих – оценивать сложность полученных реализаций в форме работы кодирования и оценивать соответствие проектной сложности [3].

В статье разработан подход к выбору альтернативных графических средств представления СПС с точки зрения уменьшения затрат и рисков технических ошибок при их создании. Конструктивным путём доказана возможность преобразования непосредственного описания СПС (ГСПС) с помощью специальных графических примитивов в диаграмму компонент UML.

Выдвинута концепция моделирования графической продукции проектирования систем с помощью СПС. Она реализована в форме определений, содержащих алгоритм построения СПС как модели ГСПС и в качестве модели для моделирующих систем диаграмм компонент UML.

Из результатов работы вытекает практическая возможность внедрения программных технологий с использованием СПС на базе стандартных средств поддержки UML моделирования. Однако в работе на основе оценок для конкретных СПС получила обоснование гипотеза о том, что такое внедрение в несколько раз рискованнее и затратнее, чем, если бы использовались специальные средства поддержки ГСПС. Поэтому целесообразно было бы разрабатывать методы компенсации этих недостатков.

Как показано в работе, имеются откры задачи для дальнейших исследований СПС графических моделей проектирования.

ЛИТЕРАТУРА

1. 1. Мищенко В. О. Прикладной язык семантической структуризации научных текстов как некоторая проекция языка Ада / В. О. Мищенко // Вісник Харківського національного університету: Зб. наук. праць. – Х., 2003. – № 605. – С. 90–105. – (Серія: Математичне моделювання. Інформаційні технології. Автоматизовані системи управління; вип. 2).
2. 2. Мищенко В.О. Энергетический анализ программного обеспечения с примерами реализации для Ада-программ. – Х.: ХНУ имени В.Н. Каразина, 2007. – 119 с.
3. 3. Мищенко В.О. CASE-оценка критических программних систем. Том 1. Оценка качества / Мищенко В.О., Поморова О. В., Говорущенко Т. А. / под ред. Харченко В. С. – : Х: Нац. аэрокосмический ун-т «Харьк. авиац. ин-т», 2012. – 201 с.
4. 4. OMG MOF 2 XMI Mapping Specification [Электронный ресурс]. – Режим доступа: http://www.omg.org/spec/xmi/2.4.1/pdf. – 09.08.2011 .
5. 5. CASE-оценка критических программних систем. Том 1. Оценка качества / Мищенко В.О, Поморова О. В., Говорущенко Т. А. / под ред. Харченко В. С. – : Х: Нац. аэрокосмический ун-т «Харьк. авиац. ин-т», 2012. – 201 с.
6. 6. Мищенко В.О. Гибкая модель приближенных вычислений ядер двумерных гиперсингулярных операторов и архитектура программной реализации / В. О. Мищенко // Вісник Харківського національного університету: Зб. наук. праць. – Х., 2008. – № 809. – С. 132–147. – (Серія: Математичне моделювання. Інформаційні технології. Автоматизовані системи управління; вип. 9).
7. 7. Мищенко В. О. Построение программных систем моделирования дифракции на идеально проводящих экранах, лежащих в диэлектрическом полупространстве // Вісник Харк. нац. ун-ту., – 2008. – № 833. Сер. «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління», вип. 10. – С. 170-184.
8. 8. Духопельников С. В. Программная система для поиска псевдособственых частот цилиндрического волновода со щелью, основанная на МДО / С. В. Духопельников, В. О. Мищенко // Вісник Харківського національного університету: Зб. наук. праць. – Х., 2009. – № 847. – С. 167–173. – (Серія: Математичне моделювання. Інформаційні технології. Автоматизовані системи управління; вип. 11).
9. 9.Холстед М. Х. Начала науки о программах / М. Х. Холстед; пер. с англ. В. М. Юфа. – М.: Финансы и статистика, 1981. – 128 с.
10. 10. Мишенко В. О. Особенности энергетических метрик UML диаграмм / В. М Годунко., В. О.Мишенко, А. В. Пасека // Вестник Харк. нац. ун-та., – 2013. – № 1058. Сер. "Математическое моделирование. Информационные технологии. Автоматизированные системы управления", вып. 21. – С. 13-19.

Надійшла у першій редакції 22.04.2013, в останній - 01.05.2013