МНОГОАГЕНТНАЯ МОДЕЛЬ СРЕДЫ ПОДДЕРЖКИ ПРОГРАММНОГО ПРОДУКТА ДЛЯ СИСТЕМ СО СЛОИСТОЙ АРХИТЕКТУРОЙ

Гурьянов В.И.
(Региональный институт психологии и гуманитарных наук, Чебоксары)
vg2007sns@rambler.ru

Рассмотрена проблема адаптивности программных продуктов. Изучено решение, использующее композиционную адаптацию и метод расширения ядра. Рассмотрена среда поддержки продукта как многоагентная система с одноуровневой архитектурой.

Ключевые слова: адаптивная программная система, композиционная адаптация, слоистая архитектура, многоагентная система, социотехническая система.

Введение

Большинство специалистов сходятся в том, что программные системы следующего поколения будут адаптивными. За последние 3–4 года рядом ведущих исследовательских центров предложены разнообразные методы создания адаптивных программных систем (adaptive software) [8]. Несмотря на это, целостная методология проектирования подобных систем пока находится в стадии зарождения.

Концептуальной основой новых методологий могут стать кибернетика и теория развития сложных систем. В частности, начиная с 2004 года, сформировалось направление, известное как программная кибернетика, которое ставит цель решить эти проблемы [11]. На текущем этапе развития этого направления большой интерес представляют модели эволюции программного продукта, допускающие строгое математическое описание.

Перспективным в этом отношении может оказаться изучение программных систем, архитектура которых определяется паттерном "выделение слоев" [3]. В статье автора [2] предложена методология проектирования модельных адаптивных программных систем, основанная на методе расслоения класса системы по интерфейсам (метод расширения ядра) в рамках парадигмы порождающего программирования [5]. За пределами этого исследования остался вопрос о способах формирования базы конфигураций программной системы.

В данной работе показано, что адаптивные программные системы не могут рассматриваться в отрыве от среды своего существования. База конфигураций есть результат развития среды поддержки продукта. Предложена методика проектирования среды поддержки адаптивного программного продукта как многоагентной системы.

1. Интерфейсная модель адаптации

Приведем некоторые понятия, необходимые для дальнейшего изложения [2].

С тем, чтобы формализовать взаимодействие информационной системы с бизнес-процессом, будем исходить из интерфейсной модели обмена сообщениями между подсистемами. В UML для моделирования стыковочных узлов в системе используются интерфейсы (см. рис.1).

Рассмотрим процесс адаптации *s*-элемента в паре *S(m,s)*. Соотнесем *m* с бизнес-процессом, а *s* – с информационной системой. Комплекс будем рассматривать как систему, взаимодействующую с окружающей средой. В рамках данного исследования *m*-элемент будем рассматривать как заданный и неизменный. Напротив, *s*-элемент может изменять свою структуру и состав.

Определение 1. Рассмотрим интерфейс подсистемы с точки зрения задач, решаемых программной системой. Под отдельным интерфейсом *f* будем понимать набор функций подсистемы, предназначенных для решения одной задачи (что, собственно говоря, совпадает с понятием интерфейса в UML). Под *интерфейсом подсистемы*, в данной статье, будем понимать кортеж *I*

= (f_0 , f_1 , … ,f_n), f_i – отдельные интерфейсы. Каждому интерфейсу f сопоставим идентификатор, и будем обозначать его той же буквой. Всюду, где не указано обратное, будем понимать f как идентификатор интерфейса.

Приведенное выше определение интерфейса отражает прагматический аспект взаимодействия. Вместо бизнес-процесса можно рассматривать варианты использования (Use case). В этом случае интерфейс соединяется с тем Use case, который его поддерживает.

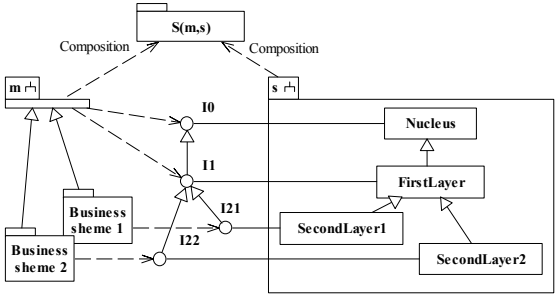

Рис. 1. Пример интерфейса между подсистемами m и s

Определим способ выделения интерфейсов. Предлагается метод, который можно назвать *методом делегирования информационных функций* от бизнес-процесса к информационной системе. Концепция метода состоит в том, что пара *S(m,s)* рассматривается как эволюционирующая система, направление развития которой определяется возрастанием степени специализации элементов *m* и *s*. Делегирование осуществляется дискретным образом, как передача информационных функций. Допустим, что *s*-элемент пары специализируется на обработке информации, а *m*-элемент – на функциях производства. Начальное состояние – все информационные функции считаются интегрированными в бизнес-процесс и входят в состав его бизнес-

функций. Таким образом, важна последовательность появления интерфейсов в I = (f_0 , f_1 , … ,f_n), что отражено в определении. Интерфейсы образуют иерархическую структуру, для описания которой, будем использовать онтологический подход (т.е. f_i будем определять так, чтобы идентификаторы могли быть обработаны программной системой; например на базе DSL).

Пример 1. На рис.1. представлены два варианта одного бизнес-процесса, которые обозначены как Sheme1 и Sheme2. Интерфейсы I21 и I22 находятся в отношении обобщения с I1. Бизнес-процессы не детализированы, напротив, программная система s представлена в виде диаграммы классов.

Определение 2. Интерфейсы, которые реализуют основную функцию программной системы, обозначим идентификатором f_0. Будем говорить, что два бизнес-процесса являются *схожими* (с точки зрения программной системы), если в интерфейсе бизнес-процесса $I(m)$ присутствует один и тот же идентификатор f_0. Схожие бизнес-процессы, отличающиеся составом интерфейсов, будем называть *вариантами* рассматриваемого бизнес-процесса. В примере 1 это Sheme1 и Sheme2.

Пример 2. Для бизнес-процесса «Хранение» идентификатор f_0 будет обозначать интерфейс, который обрабатывает сообщения ПОСТУПЛЕНИЕ, ВЫДАЧА, ХРАНИМОЕ_КОЛИЧЕСТВО и запрос СООБЩИТЬ ХРАНИМОЕ_КОЛИЧЕСТВО. Решаемая бизнес-процессом задача – задача управления, т.е. выдача команд на выполнение функций ПРИНЯТЬ, ВЫДАТЬ. Вариант использования (Use case) можно определить как «Учет». Прочие интерфейсы появляются на второй и последующих итерациях делегирования информационных функций. Интерфейсом может быть совокупность методов обработки сообщений для задач снабжения, сбыта, планирования или отчетности (Use cases: «Закупка», «Анализ спроса», «Эффективность использования склада» и др.), т.к. необходимость решения этих задач определяется особенностями бизнес-процесса.

Информационная система предназначена для обслуживания бизнес-процесса и должна соответствовать некоторому набору требований, который обозначим как Z и будем в дальнейшем называть *спецификацией* программной системы. Определим спецификацию, выданную *m*-элементом пары, как Z = (f_0 , f_1 , ...

f_n), причем $Z = I(m)$. Рассмотрим три аспекта связи m и s: прагматический, семантический и синтаксический. Согласно принципу подчиненности процесс адаптации начинается с прагматического аспекта. Сделаем следующее предположение: адаптация системы обусловлена прагматическим аспектом интерфейса подсистемы, т.е. составом и последовательностью появления элементов в $I(m) = (f_0, f_1, \ldots f_n)$. Далее допустим, что адаптация в семантическом и синтаксическом аспектах происходит мгновенно (по сравнению со временем реорганизации s-элемента).

Под *структурной адаптацией* будем понимать процесс реструктуризации программного обеспечения w таким образом, чтобы его интерфейс соответствовал заданному, т.е. $I(w) = Z$. Будем также полагать, что после адаптации сохраняется соотношение $I(m) = Z$. В частности, развитие информационной системы можно рассматривать как структурную адаптацию. Следует различать программное обеспечение w и s-элемент пары.

Для управления сборкой необходимо определить связь интерфейса объекта $I(s)$ с его структурой. Во многих случаях описание этой связи вызывает значительные трудности и является основным препятствием для построения модели системы. Данная проблема может быть решена для слоистых структур.

Особенность слоистых структур состоит в направленности связей со слоя i на слой i+1, но не наоборот [3]. Для определения слоев предложен метод расслоения класса по идентификаторам спецификации $Z = (f_0, f_1, \ldots f_n)$. Метод основан на последовательном применении наследования к классу продукта T, причем с каждым слоем будем связывать один интерфейс с идентификатором $f_i \in Z$. Пример такой архитектуры (классы Nulcleus, FirstLayer и т.д.) приведен на рис.1.

Фактически, данный метод представляет собой объектно-ориентированное обобщение *метода расширения ядра*. В качестве аналога метода можно рассматривать итерационную процедуру построения структуры организации посредством делегирования обязанностей. Метод расширения ядра достаточно эффективен в процессе проектирования небольших офисных приложений. Предлагаемое решение позволяет создавать адаптивные программные системы, удобные для теоретического

анализа. В [2] приведен пример такого приложения для среды динамического программирования Smalltalk.

Обозначим слои буквами алфавита $V_T = \{a_0, a_1, \ldots, a_n\}$ некоторого формального языка Lu, множество идентификаторов интерфейсов обозначим как P. Сопоставим каждой букве алфавита $a_i \in V_T$ идентификатор $f_j \in P$ и обозначим множество пар (a_i, f_j) как F, а фактор-множество как $V_T/F = \{f_0|\{a_1, a_2, \ldots a_k\}, f_1|\{b_1, b_2, \ldots b_l\}, \ldots, f_N|\{z_1, z_2, \ldots z_m\}\}$, где $a_j, b_j, \ldots, z_j \in V_T$. Далее, сопоставим каждой букве $a_i \in V_T$ упорядоченную пару $L_i = (\{l_1,\ldots,l_{Ni}\}, \{r_1,\ldots,r_{Mi}\})$, где l– левые (входные), r– правые (выходные) связи. Обозначим алфавит связей как V_L,, а множество пар L_i как L. Пятерка $J(V_T, V_L,, P, F, L)$ определяет механизм сборки системы s.

Рассмотрим характерный частный случай, когда процесс формирования архитектуры программной системы может быть представлен орграфами. Для описания ориентированного дерева сборки удобна следующая реляционная запись.

Определение 3. Проблема адаптации может быть решена, если класс T продукта s имеет множественный интерфейс $I(T) = \{I_\lambda\}_{\lambda \in J}$, J – множество индексов. Адаптация сводится к выбору одного из интерфейсов. Далеко не все языки программирования поддерживают множественные интерфейсы. Более того, создать такой класс для программной системы можно только в некоторых частных случаях. Для систем динамического программирования может быть выполнена эмуляция множественного интерфейса.

Введем для $I_\lambda \in I(T)$, $I_\lambda = (f_0, f_1, \ldots, f_q)$ кортеж вида $d_\lambda = ((f_0, a_\lambda), (f_{\lambda 1}, b_\lambda), \ldots, (f_{\lambda q}, z_\lambda))$, $f_i \in P$, $a_\lambda, b_\lambda, \ldots, z_\lambda \in V_T$ (далее *сегмент*) и составим множество $D = \{d_1, d_2, \ldots, d_M\}$ из всех d_λ, $\lambda \in J$. Будем называть его *базой конфигураций* объекта s. Можно говорить об отображении X: $D \to I(T)$. Пусть $R = X^1(Z)$. Таким образом, задача адаптации продукта $s(T)$ будет решена, если задано отображение X и правило выбора $d \in R$.. О сегменте d будем говорить как об *активном сегменте D*. С точки зрения объектно-ориентированного программирования имеет место динамическое порождение класса T' с единственным интерфейсом $I(T') = Z$.

Когда s-элемент пары получает спецификацию Z от m-элемента, инициируется субпроцесс «проектирование», который сводится к выполнению отображения $R = X^{1}(Z)$ и, если необходимо, выбору $d \in R$. Затем проект системы $d = ((f_0, a_\lambda), (f_{\lambda 1}, b_\lambda), \ldots, (f_{\lambda q}, z_\lambda))$, $\lambda \in J$ передается в субпроцесс «рекомпозиция», который синтезирует класс T' программной системы с интерфейсом $I(w) = (f_0, f_{\lambda 1}, \ldots, f_{\lambda q})$, $\lambda \in J$. По завершению процесса сборки классов производится перезагрузка программной системы с сохранением актуального состояния («регенерация»). В итоге, изменение спецификации Z приводит к изменению структуры w так, что равенство $I(w) = Z$ будет выполнено, что и позволяет говорить об адаптивности s.

Реляционная запись процесса сборки классов позволяет свести процесс адаптации к выполнению автоматических процедур.

2. Среда существования продукта как многоагентная система

Поставим теперь вопрос о формировании базы конфигураций программной системы.

Отдельный бизнес-процесс, как правило, может поддерживать разработку только одного сегмента D. С другой стороны описанная выше модель адаптивности эффективна только при условии, что D имеет более одного сегмента. Тем самым, рассмотренная выше модель адаптивной системы будет не полной, если не рассмотреть способ формирования базы конфигураций $D = \{d_1, d_2, \ldots, d_M\}$. Каждый сегмент d_λ должен соответствовать одной ветви сборки и ветвь должна быть верифицирована.

Определение 4. Предположим, что в общем случае производство базы конфигураций должно быть результатом кооперации всех пользователей системы s. Тогда централизованная и децентрализованная разработка программного обеспечения будет частным случаем такой кооперации. Множество всех пользователей одной и той же адаптивной программной системы будем называть *средой существования программного продукта*. Если среда существования продукта рассматривается в

контексте обеспечения инжиниринга, то будем говорить о *среде поддержки продукта*.

Несмотря на множество работ в области программной кибернетики, модель среды существования адаптивной программной системы, как самостоятельная проблема, еще не рассматривалась [11].

Методология проектирования такой системы будет определяться парадигмой агентного программирования [10]. Проектирование многоагентной системы (далее МАС) для среды существования продукта имеет ряд особенностей. Рассмотрим ключевые отличия.

Разработку МАС начнем с того, что в роли агентов будем рассматривать объекты адаптивной программной системы.

Согласно определению, онтология должна обеспечивать формирование словаря терминов, допускающих интерпретацию программной системой. Будем выделять *онтологию требований* (спецификация и смысл идентификаторов интерфейсов f_i) и *онтологию конфигураций*. Онтология конфигураций зафиксирована в базе конфигураций D.

Онтологический инжиниринг (например, методология системы Protégé) предполагает формирование законченной базы знаний, которая затем только используется агентами. Напротив, в рассматриваемом случае, формирование онтологий – это основная функция многоагентной системы. Поэтому часть средств разработки онтологий должны быть включены в функциональность агентов. Отметим, что эта особенность сближает рассматриваемую многоагентную систему с системами распределенного искусственного интеллекта.

Таким образом, ключевым вопросом проектирования многоагентной системы следует считать уточнение онтологического инжиниринга. Выделяют следующие значимые свойства онтологии: существенность (охват предметной области), непротиворечивость, независимость от реализации, декларативность, расширяемость, ясность.

Поддержание корректности и целостности, непротиворечивости и расширяемости онтологий можно обеспечить путем жесткого (неизменяемого) определения обязательств агентов.

Обозначим множество пар $S(m_\mu,s_\mu)$, $\mu \in H$ (H – множество индексов), имеющих общий интерфейс f_0 , как Ξ. Пусть на множестве Ξ существует некоторое подмножество активных пользователей, т.е. таких, которые могут вносить изменения либо в базу конфигураций D системы, либо в ее компоненты $a \in V_T$. Допустим, что между элементами Ξ существует возможность передачи базы конфигураций и может быть определена некоторая операция объединения баз. Обозначим обновляемую базу конфигураций как D_μ^+, а некоторую другую – как D_v^-. В случае слоистых структур можно ввести специальную операцию соединения $\aleph$: $(D_\mu^+, D_v^-) \to (D_\mu^{+\prime}, D_v^{-\prime})$ [1]. Данная операция эмулирует множественное наследование. Регенерация системы по $D_\mu^{+\prime} = \aleph_1(D_\mu^+, D_v^-)$ приведет к обновлению модулей системы, причем интерфейс системы, определяющий адаптацию, сохранится. По $D_v^{-\prime} = \aleph_2(D_\mu^+, D_v^-)$ регенерация системы невозможна, этот вариант базы конфигураций удобен для транспортировки (и, тем самым, операция $\aleph$ обеспечивает самодостаточность системы).

Итак, можно определить следующий шаблон проектирования многоагентной системы. Зададим четыре основных типа агентов: два реактивных агента D^+ и D^- и два интеллектуальных агента $S(m, s)$ и $S'(m, s)$ в роли заказчика и в роли исполнителя соответственно. Взаимодействие агентов D^+ и D^- определяется операцией соединения. Агенты $S(m, s)$ и $S'(m, s)$ должны иметь некоторые ментальные свойства. Будем рассматривать многоагентные системы с одноуровневой (полностью децентрализованной) архитектурой.

Протокол исполнителя $S'(m, s)$ предполагает: (а) сбор сведений посредством стандартного сервиса сети о множестве Ξ, (б) рассылка предложений и (в) протокол рассылки D^-.

Протокол заказчика $S(m, s)$ – протокол ведения переговоров для модели контрактных сетей [9]. В основе модели лежит простейшая идея рыночных торгов. В этом случае $S(m, s)$ выступает в роли агента-менеджера, а $S'(m, s)$ – как агент-исполнитель. Агент-менеджер отбирает самые выгодные для него предложения (по ожидаемой степени соответствия $I(w) = Z$) и заключает соглашение с выбранными агентами-исполнителями на поставку D^-.

Язык коммуникации для агентов $S(m, s)$ и $S'(m, s)$ может быть KQML (Knowledge Query and Manipulation Language).

Отдельным вопросом является проблема существенности предметной онтологии. Именно достижение существенности и является целью МАС. Можно ожидать, что эволюция МАС приведет к выработке некоторой «универсальной» базы конфигураций, которая обеспечит существенность предметной онтологии. С тем, чтобы исследовать возможные варианты эволюции, составим математическую модель МАС. Будем придерживаться подхода, использованного в работе [7].

Рассмотрим все возможные варианты баз конфигураций, которые могут появиться на множестве пользователей Ξ. Данное множество обозначим как Ω. Пусть в каждой паре $S(m_\mu,s_\mu)$ реализуется один из интерфейсов $\{I_k\}_{k\in K}$, K – множество индексов. Обозначим соответствующее фактор-множество как $\{\Xi_k: k \in K\}$, N_k – количество пар в Ξ_k.

Зададим отображение множества индексов $\Lambda = \{0,1,\ldots,N\}$ на Ω. В Ω выделим некоторые подмножества. Ситуацию, когда база конфигураций пустая, будем отмечать индексом 0 и обозначим символом $D_0 \in \Omega$. Если база конфигураций $D = (d_1)$ определяет слово из одной буквы $d_1=\{(f_0, a)\}$, соотнесенной с интерфейсом f_0, то положим, что $D \in \Omega_{\text{dist}}$ и индексируются элементами подмножества Λ_{dist} Первые два случая соответствуют начальным состояниям среды. Если Ω_{dist} содержит более одного элемента, то программные продукты имеют одинаковые функции, но отличаются ядром системы.

Пусть $u_i^{(k)}$ количество s – элементов имеющих базу конфигураций вида $i \in \Lambda$ и входящих в Ξ_k. Допустим, что N_k = const, $\forall k \in K$. Для описания эволюции среды запишем динамическую систему в виде ($v_i^{(k)} = u_i^{(k)}/N_k$, $u_i^{(k)} = v_i^{(k)} N_k$, $v_i^{(k)} \to u_i^{(k)}$)

$$\frac{1}{N_l}\frac{du_i^{(l)}}{dt} = u_i^{(l)}[\sum_k \frac{N_k}{N_l}\sum_j(-b_{ij}^{(k)} + b_{ji}^{(k)})u_j^{(k)}] - (\sum_j a_{ij}^{(l)}) f_i(u_i^{(l)}) + \sum_j a_{ij}^{(l)} f_i(u_j^{(l)})$$

$$\sum_j u_j^{(k)} = 1, \forall k \in K ,$$

где верхний индекс при переменных указывает на индекс фактор-множества $\{\Xi_k: k \in K\}$, $f(u_i) \sim u_i^2$. Система уравнений разбивается на блоки по индексу группы Ξ_l , $l \in K$. В каждом блоке l используются переменные $u_i^{(l)}$ для $\forall i \in \Lambda$.

Смысл коэффициентов следующий. Пусть некоторая пара $S(m_\alpha,s_\alpha)$ с базой конфигураций i получает базу конфигураций j от некоторой пары $S(m_\beta,s_\beta)$ и возможна замена базы конфигураций i на j. Интенсивность этого процесса $b_{ij}^{(k)}$, $l = k$ (*матрица предпочтений*). Возможна диффузия баз конфигураций за границы Ξ_k. Интенсивность диффузии из Ξ_k в Ξ_l определяется значениями $b_{ij}^{(k)}$, $l \neq k$. С другой стороны, если текущая структура s-элемента не соответствует требованиям бизнес-процесса, возможна модификация базы конфигураций $i \to j$. Этот процесс опишем *матрицей модификаций* a_{ij}.

Универсальная база конфигураций $D_u = \{d_1, d_2,\ldots,d_M\}$ соответствует устойчивому стационарному состоянию динамической системы. Таким образом, можно говорить, что требуемое состояние в многоагентной системе, по крайней мере, существует.

Итак, описанная выше многоагентная модель среды существования продукта обеспечивает механизм формирования онтологий и может служить шаблоном проектирования МАС.

3. Методология проектирования

В настоящее время предполагается, что адаптивные программы могут быть реализованы локально, в рамках отдельных бизнес-процессов. Анализ модели позволяет сформулировать альтернативное утверждение. Адаптивные программные системы должны проектироваться и эксплуатироваться как распределенные системы поддержки инжиниринга.

Таким образом, методология проектирования адаптивных систем, наряду с традиционными для порождающего программирования разделами инженерии, должна включать также методологию проектирования многоагентной системы.

Уточним задачу проектирования. Вернемся к динамической системе, описывающую эволюцию онтологий в МАС. Конечное

состояние среды определяется набором устойчивых стационарных состояний. Однако их достижение может потребовать значительного времени. Время достижения состояния – это один из показателей среды поддержки продукта; его значение должно быть минимальным.

Введем пространство визуализации для множества Ξ как декартово произведение $K \times \Lambda$. Потребуем, чтобы конечное состояние $u_i^{(k)} \neq 0$, $i \in \Lambda_k$. для каждой группы Ξ_k, $k \in K$, было единственным. В подпространстве решений возможно два предельных случая: (а) «мозаика» – общая база конфигураций в границах Ξ_k и разная для разных $k \in K$ и (б) «полоса» – общая для всех Ξ_k универсальная база конфигураций D_u. И в том и в другом случае базы конфигураций равноценны с точки зрения интерфейса. Однако последний вариант базы конфигураций предпочтительнее по следующим двум причинам: (а) для поддержки программного обеспечения используются ресурсы всех Ξ пользователей; (б) при колебаниях интерфейса бизнес-процесса ($f_i \to f_j$) адаптация системы не потребует повторного расхода ресурса. Можно также сказать, что универсальная база конфигураций задает всю линейку продукта.

Проектируемая среда поддержки продукта будет находиться между этими двумя крайними случаями. Шаблон обеспечивает кооперацию агентов МАС для решения одной задачи – формирование онтологий множества Ξ. С другой стороны внешние факторы – организационные структуры, финансовые отношения, технические ограничения и др. – формируют свои сообщества. Параметрами проектирования выступают внешние факторы МАС, которые накладывают разного рода ограничения на характер обмена базами конфигураций. Для проектирования вторичного сообщества можно использовать методологию нисходящего проектирования [4]. В качестве социальных критериев достаточно выбрать базовые типы взаимодействия агентов [6].

Рассмотрим пример. Пусть $V_T = \{a,b,c\}$, $P = \{f_0, f_1, f_2\}$, $V_T/F = \{f_0|\{a\}, f_1|\{b\}, f_2|\{c\}\}$. Множество интерфейсов на множестве пользователей Ξ будет $\{I_1, I_2\}$, $I_1=(f_0, f_1)$, $I_2=(f_0, f_2)$. Распреде-

лим пары по группам согласно их интерфейсам, фактор-множество будет $\{ I_1|\, \Xi_1, I_2|\, \Xi_2\}$.

Множество баз конфигураций $\Omega = \{ D_0, D_1, D_2, D_3, \}$, где D_0 – символ пустой базы конфигураций, $D_1 = \{((f_0, a))\}$- база конфигураций неадаптированной системы, $D_2 = \{((f_0, a), (f_1, b))\}$ и $D_3 = \{((f_0, a), (f_2, c))\}$ – база конфигураций для слов ab и ac (причем $I(ab) = I_1$, $I(ac) = I_2$), $D_4 = \{ ((f_0, a), (f_1, b)), ((f_0, a), (f_1, c))\}$ – универсальная база конфигураций. Динамическая система будет иметь восемь уравнений по четыре для каждой из групп Ξ_1, Ξ_2.

Пусть в начальном состоянии все пары имеют базу конфигураций D_1. Проследим за изменением базы конфигураций одной пары $S(m_\mu,s_\mu)$, входящей, например, в группу Ξ_1. Если в Ξ_1 уже существуют адаптированные системы и их база конфигураций доступна (переменная $u_2^{(1)} > 0$ или $u_4^{(1)} > 0$), то возможна замена $D_1 \to D_2$ и $D_1 \to D_4$ т.е. $b_{12}^{(1)}$, $b_{14}^{(1)} > 0$, $b_{21}^{(1)} = b_{41}^{(1)} = 0$.

Развитие программного обеспечения может быть выполнено непосредственно в $S(m_\mu,s_\mu)$, т.е. $D_1 \to D_2$ значение $a_{12} > 0$, $a_{21} = 0$. С другой стороны, $S(m_\mu,s_\mu)$ может получить базу конфигураций D_3 (переменная $u_3^{(1)} > 0$), из группы Ξ_2 (переменная $u_3^{(2)} > 0$) и выполнить его модернизацию до D_4 ($a_{14} > 0$, $a_{14} \approx a_{12}$). Последний случай наиболее интересен, т.к. он демонстрирует способ возникновения универсальной базы конфигураций D_4.

Требуемое конечное состояние системы есть $u_0 = u_1 = u_2 = u_3 = 0$, $u_4 = 1$, причем время достижения этого состояния должно быть меньше характерного времени изменения бизнес-среды.

Пусть количество пользователей достаточно велико (техническое ограничение). Другие какие-либо внешние ограничения на множестве Ξ отсутствуют. В качестве типа сообщества можно выбрать эгалитарную структуру в форме координируемого сотрудничества. Обмен словарями спецификаций и базами конфигураций будет организован через доски объявлений.

На том же множестве Ξ могут формироваться и другие сообщества. Например, корпорация, имеющая множество филиалов и заинтересованная в создании универсальной базы конфигураций, может создать свое сообщество в форме простого сотрудничества.

В заключение приведем эталонную модель проектирования. Адаптивная программная система представляет собой систему с распределенной средой поддержки продукта. Экземпляры системы расположены в узлах сети. Программная система функционирует в системе динамического программирования типа CLOS, Python, Open Java. Инсталляция программной системы имеет место при загрузке соответствующей базы конфигураций в оболочку s программной системы и ее активизацию при выборе начальной спецификации Z. Адаптивная система в режиме эксплуатации производит модификацию своей архитектуры таким образом, чтобы ее функции соответствовали меняющейся спецификации Z.

Адаптивная система может выступать как в роли эксплуатируемой системы, так и в роли системы разработки. Роли могут изменяться. Жизненный цикл эксплуатируемой системы соответствует спиральной модели и включает этапы: разработка требований, проектирование, конструирование, тестирования; сопровождение. Все этапы, кроме разработки требований, носят автоматический характер. Разработка требований - единственный процесс, требующий присутствия инженера.

Разработка требований состоит из сбора требований и анализа требований. Сбор требований выполняется согласно общей методологии (например, методом А. Джекобсона). Анализ требований (определение объектов системы), напротив, автоматичен. Точка соприкосновения лежит на соответствии Use cases (неформальное описание) с их спецификацией (формальное описание). Спецификация Use cases осуществляется инженером на основе словаря онтологии требований. Обновление и расширение словаря онтологии требований – функция среды поддержки продукта.

Сопровождение включает (а) коммуникации посредством MAC (в т.ч. формирование вторичных сообществ), (б) получение другой базы конфигураций и выполнение операции соединения (при необходимости) и (в) контроль регенерации системы. Сопровождение выполняется в автоматическом режиме.

Жизненный цикл системы, как системы разработки продукта, соответствует спиральной модели. Кроме того, включает процесс рассылки обновления для словаря онтологии требований. Этот процесс можно назвать *презентацией решения*. Подчеркнем, что внесение изменений в словарь онтологии требований допускается только при модификации базы конфигураций.

Весь процесс адаптации предполагает минимальное участие человеческого фактора, что обеспечивает надежность и низкие стоимостные показатели модификации программной системы.

Выводы

В статье рассмотрена методология проектирования адаптивных программных систем, архитектура которых основана на методе расширения ядра. Показано, что адаптивный программный продукт должен рассматриваться только вместе с распределенной системой поддержки продукта. Таким образом, адаптивная программная система представляет собой социотехническую систему, целостность которой поддерживается агентными технологиями.

Приведенная выше методология может быть применена для разработки приложений на основе языков, допускающих динамическую рекомпозицию. Рассмотренная методология имеет много общего с методологией, известной как «фабрика приложений» (Software Factories). Концептуальные отличия следующие: (а) фиксированный архитектурный каркас, (б) использование онтологий вместо базы рекомендаций и (в) определение архитектуры фабрики в контексте среды существования продукта.

Следующим шагом развития методологии может стать перенос методов в сферу проектирования «больших» программных систем. Очевидный путь - формулировка методов для программ, использующих платформу промежуточного слоя. Однако более продуктивным и более естественным обобщением методологии вероятнее всего будет определение методов для платформ агентного программирования.

Литература

1. ГУРЬЯНОВ В.И., *Адаптивная сборка класса* / Современные информационные технологии в науке, образовании и практике. Материалы IV всероссийской конференции. – Оренбург: ИПК ГОУ ОГУ, 2007. С. 31-32.
2. ГУРЬЯНОВ В.И., *Адаптивная информационная система для модельной задачи управления стеком* / Информационные технологии моделирования и управления. № 1(44). – Воронеж, Изд-во «Научная книга», 2008 - С. 52-61.
3. КСЕНЗОВ М.В. *Рефакторинг архитектуры программного обеспечения*, М.: ИСП РАН, Препринт 4, Москва, 2004
4. ТАРАСОВ В. Б. *От многоагентных систем к интеллектуальным организациям: философия, психология, информатика*. – М.: Эдиториал УРСС, 2002. - 352 с.
5. ЧЕРНЕЦКИ К., АЙЗЕНЕКЕР У. *Порождающее программирование: методы, инструменты, применение* / Пер. с англ. СПб: Питер, 2005. 736 с.
6. FERBER J., JACOPIN E. *The Framework of Eco-Problem Solving* // Decentralized Artificial Intelligence II / Ed. by Y.Demazeau, J.-P.Muller. – Amsterdam: Elsevier North-Holland, 1991.
7. FERBER J. *Les systemes multi-agents. Vers une intelligence collective*. – Paris: InterEditions, 1995.
8. MCKINLEY P., SADJADI S.M., KASTEN E., CHENG B. *Composing Adaptive Software* // IEEE Computer Society, Vol. 37, No 7, July 2004. P.27-30.
9. SMITH R.G. *The Contract Net Protocol: High Level Communication and Control in a Distrubuted Problem Solver* // IEEE Transactions on Computers. – 1980. – Vol. 29, №12. – P.1104-1113.
10. SHOHAM Y. *Agent–oriented programming*.- Artif.Intell. 1993, 60, №1.– P.51–92.
11. *The Third IEEE International Workshop on Software Cybernetics* // IWSC 2006. Chicago, September 18-21, 2006; *The Fourth IEEE International Workshop on Software Cybernetics* // IWSC 2007 Beijing, China, July 24, 2007.