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

Одним из наиболее важных качеств программного обеспечения следует считать свойство адаптивности (adaptive software). За последние 3–4 года начало складываться направление, обращенное на разработку технологий создания и поддержки адаптируемых и адаптивных программных систем [8].

Современные методологии проектирования программного обеспечения рассматривают программиста в качестве ключевой фигуры. В результате этого процессы, реализуемые программными системами, пока не имеют формальных средств описания, характеризуемых достаточной математической строгостью. Начиная с 2004 года, сформировалось направление, известное как программная кибернетика, которое ставит цель решить эти проблемы [11]. В связи с этим большой интерес представляют модели развития программного продукта, допускающие строгое математическое описание.

Одним из наиболее употребляемых паттернов рефакторинга архитектуры программного обеспечения является паттерн "выделение слоев" [3]. Для слоистых структур можно сформулировать относительно простые механизмы реструктуризации, построить модель развития систем и среды существования продукта.

В статье рассмотрена теоретическая модель адаптивной программной системы, композиция которой основана на слоях. Модель позволяет проанализировать основные понятия процесса адаптации и принципы проектирования адаптивных программных систем.

Показано, что адаптивные программные системы не могут рассматриваться в отрыве от среды своего существования. Предложена методика проектирования среды поддержки адаптивного программного продукта как многоагентной системы.

1. Интерфейсная модель адаптации

С тем, чтобы формализовать взаимодействие информационной системы с информационной средой бизнес-процесса, будем исходить из интерфейсной модели обмена сообщениями между подсистемами.

Рассмотрим процесс адаптации *s*-элемента в паре *S(m,s)*. Соотнесем *m* с бизнес-процессом, а *s* – с информационной системой. 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. Далее будем говорить об *m*-элементе (или главном компоненте) и *s*-элементе (или подчиненном компоненте) пары. 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.

Под интерфейсом будем понимать кортеж $I = (f_0, f_1, \ldots, f_n), f_i$ – элементы интерфейса. Элемент интерфейса – это, в терминологии объектно-ориентированного подхода, совокупность свойств и методов объекта, предназначенных для решения

одной задачи. Элементы интерфейса можно получить *методом делегирования информационных функций* от бизнес-процесса к информационной системе. Некоторые элементы интерфейса имеют особое значение и реализуют основную функцию программной системы. Их можно рассматривать как таксономические признаки. Далее будем называть эти элементы интерфейса *атрибутными*. Допустим, что только один элемент интерфейса является атрибутным; обозначим его как f_0. Остальные элементы интерфейса будем называть *вариантными*. Они могут присутствовать в интерфейсе бизнес-процесса, но могут и отсутствовать.

Например, для бизнес-процесса «Хранение» атрибутный элемент интерфейса будет обрабатывать сообщения ПОСТУПЛЕНИЕ, ВЫДАЧА, ХРАНИМОЕ_КОЛИЧЕСТВО и запрос СООБЩИТЬ ХРАНИМОЕ_КОЛИЧЕСТВО. Решаемая задача – задача управления, т.е. выдача команд на выполнение функций ПРИНЯТЬ, ВЫДАТЬ. Вариантным элементом интерфейса может быть совокупность методов обработки сообщений для задач снабжения, сбыта или отчетности, т.к. необходимость решения этих задач определяется характером взаимодействия бизнес-процесса с внешней средой.

Обозначим интерфейс m-элемента пары как $I(m) = (f_0, f_1, ... ,f_n)$. 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, т.е. составом и последовательностью появления элементов в интерфейсе $I(m) = (f_0, f_1, ... ,f_n)$. Далее допустим, что адаптация в семантическом и синтаксическом аспектах (*параметрическая адаптация*) происходит мгновенно (по сравнению со временем реорганизации s-элемента). Каждому элементу интерфейса $f \in I$ сопоставим идентификатор, будем называть его *прагматикой* и обозначать той же буквой.

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

Процесс реструктуризации программной системы выполняется методами композиционной адаптации [8]. 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]. Слои определим методом расслоения класса по прагматикам спецификации $Z = (f_0, f_1, ... ,f_n)$. Метод основан на последовательном применении наследования к классу продукта T, причем с каждым слоем будем связывать одну прагматику. 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 хорошо работает для небольших офисных приложений. Предлагаемое решение позволяет создавать адаптивные программные системы, удобные для теоретического анализа.

Обозначим слои буквами алфавита $V_T = \{a_0, a_1, ... , a_n\}$ некоторого формального языка Lu, множество прагматик обозначим как P. Сопоставим каждой букве алфавита $a_i \in V_T$ прагматику $f_j \in P$ и обозначим множество пар (a_i, f_j) как F, а фактор-множество как $V_T/F = \{f_0|\{a_1, a_2, \dots a_k\}, f_1|\{b_1, b_2, \dots b_l\}, \dots, f_N|\{z_1, z_2, \dots z_m\}\}$, где $a_j, b_j, \dots, z_j \in V_T$. Далее, сопоставим каждой букве $a_i \in V_T$ упорядоченную пару $L_i = (\{l_1,...,l_{Ni}\}, \{r_1,...,r_{Mi}\})$, 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. С позиций порождающего программирования $J(V_T, V_L, P, F, L)$ представляет собой характеристическую модель домена предметной области (т.е. адаптивной программной системы) [5].

Процесс адаптации представляет собой процесс развития системы и в общем случае требует описания в рамках теории развития сложных систем. Приведенная выше пятерка $J(V_T, V_L, P, F, L)$ определяет только множество Lu допустимых структур и ничего не говорит о том, в каком порядке следует производить сборку слов. Допустимые изменения структуры задаются законом развития системы.

В случае слоистых структур закон развития может быть задан грамматиками Хомского специального вида G^A (V_T, V_N, R, S), причем $L(G^A) \subseteq Lu$. Тогда всякий вывод в G^A описывает эволюцию системы, а формальная грамматика определяет генеративную модель домена.

Ниже приведено решение этой задачи для частного случая, когда процесс развития может быть представлен орграфами. Для описания ориентированного леса деревьев сборки удобна следующая реляционная запись.

Пусть класс T продукта s имеет множественный интерфейс $I(T) = \{I_\lambda\}_{\lambda \in J}$, J – множество индексов. Создать такой класс для программной системы можно только в некоторых частных случаях. 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 сборки* или *метакодом* объекта s. При этом мы предполагаем, что последовательность сборки d_λ верифицирована в G^A (т.е. программная система протестирована на каждом этапе разработки). Тогда можно говорить об отображении X: $D \rightarrow I(T)$. 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* метакода D. С точки зрения объектно-ориентированного программирования имеет место динамическое порождение класса T' с единственным интерфейсом $I(T') = Z$. В терминах порождающего программирования [5] метакод определяет генеративную модель домена адаптивной программной системы.

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 «регенерация», который синтезирует класс T' программной системы (собирает слово w) с интерфейсом $I(w)=\{f_0, f_{\lambda 1}, \ldots, f_{\lambda q}\}$, $\lambda \in J$. По завершению процесса сборки класса производится перезагрузка программной системы с сохранением актуального состояния. В итоге, изменение спецификации Z приводит к изменению структуры слова w так, что равенство $I(w) = Z$ будет выполнено, что и позволяет говорить об адаптивности s.

Реляционная запись процесса сборки слов (т.е. классов) позволяет свести процесс адаптации к выполнению автоматических процедур. С тем, чтобы отличать автоматический режим проектирования программной системы от процесса с участием инженера, будем использовать термин *автоинжиниринг*. Следует подчеркнуть, что механизм сборки и механизм проектирования, в этом случае, являются составляющими частями самого изделия.

2. Среда существования продукта как многоагентная система

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

Предположим, что в общем случае производство метакода должно быть результатом кооперации всех пользователей системы s. Тогда централизованная и децентрализованная разработка программного обеспечения будет частным случаем такой кооперации. Методология проектирования такой системы будет определяться парадигмой агентного программирования [10].

Несмотря на множество работ в области программной кибернетики, модель среды существования адаптивной программной системы, как самостоятельная проблема, еще не рассматривалась [11]. Если среда существования продукта рассматривается в контексте обеспечения инжиниринга, то будем говорить о *среде поддержки продукта*.

Проектирование многоагентной системы (далее МАС) для среды существования продукта имеет ряд особенностей. Рассмотрим ключевые отличия.

Разработка МАС обычно выполняется по методологии восходящего проектирования. Определение генеративной модели домена фактически определяет основной набор агентов и их архитектуру, т.к. в роли агентов рассматриваются элементы адаптивной программной системы. С другой стороны существует важный аспект проектирования, характерный именно для адаптивных систем. Это связано с тем, что специфицирование онтологий является заключительным этапом проектирования, что в нашем случае невозможно.

Будем выделять *онтологию требований* (структура и смысл элементов интерфейса f_i) и *онтологию сборки*.

Онтология должна обеспечивать словарную поддержку, предлагая программно интерпретированный ответ о лексике данной проблемной области. 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 определения обязательств агентов.

Обозначим множество пар $S(m_\mu,s_\mu)$, $\mu \in H$ (H – множество индексов), имеющих общую атрибутивную прагматику f_0 , как Ξ. Пусть на множестве Ξ существует подмножество θ активных пользователей, т.е. таких, которые могут вносить изменения либо в метакод D системы, либо в ее элементы $a \in V_T$. Допустим, что между элементами Ξ существует возможность передачи метакода и может быть определена некоторая операция объединения метакодов. Это не всегда можно сделать ввиду того, что новый метакод должен сохранять адаптационные признаки эксплуатируемой системы s_μ в паре $S(m_\mu,s_\mu)$. В случае слоистых структур можно ввести операцию перекрестного соединения $\aleph: (D_\mu^+, D_v^-) \to (D_\mu^{+\prime}, D_v^{-\prime})$ [1]. Регенерация системы по метакоду $D_\mu^{+\prime} = \aleph_1(D_\mu^+, D_v^-)$ 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 метакод удобен для транспортировки базы сборки (и, тем самым, 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 интенциональных агента $S(m, s)$ и $S'(m, s)$ в роли заказчика и в роли исполнителя соответственно. Будем рассматривать многоагентные системы с одноуровневой (полностью децентрализованной) архитектурой. Взаимодействие агентов D^+ и D^- определяется операцией соединения, и поддерживается языком типа KIF (Knowledge Interchange Format).

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 этом случае S (m, s) выступает в роли агента-менеджера. Цена определяется в первую очередь уровнем соответствия $I(s) = Z$. Возможно также

использование других протоколов, например, протокола для планирования встреч. При этом S (m, s) выступает в качестве инициатора встречи, а контакты устанавливаются по списку предложений.

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 некоторого оптимального метакода, который обеспечит существенность предметной онтологии. С тем, чтобы исследовать возможные варианты эволюции, составим математическую модель МАС. Будем придерживаться подхода, использованного в работе [6].

Рассмотрим все возможные варианты метакода, 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 метакод отсутствует, будем отмечать индексом 0 и обозначим символом $D_0 \in \Omega$. Если метакод $D = (d_1)$ определяет слово из одной буквы $d_1=\{(f_0, a)\}$, соотнесенной с атрибутной прагматикой f_0, то положим, что $D \in \Omega_{dist}$ 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.

В ожидаемом конечном состоянии развития среды метакод будет содержать сегменты, которые определят слова w, $I(w) = I_k$, (или $X^{-1}(I_k) \neq \varnothing$). Обозначим эти подмножества как $\Omega_k \subset \Omega$, $\Lambda_k \subset \Lambda$, $k \in K$. Отметим, что этот же метакод может содержать сегменты, определяющие слова с другим интерфейсом.

Остальные индексы будут обозначать метакоды, определяющие слова w с частичным покрытием интерфейса ($I(w) \subset I_k$) или с ненасыщенными связями (связи между слоями имеют дефект), т.е. метакодам, находящимся в процессе развития.

Пусть $u_i^{(k)}$ количество s – элементов имеющих метакод 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 ($v_i^{(k)} = u_i^{(k)}/N_k$, $u_i^{(k)} = v_i^{(k)} N_k$, $v_i^{(k)} \rightarrow u_i^{(k)}$)

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

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

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 метакодом i получает метакод j от некоторой пары $S(m_\beta,s_\beta)$ и возможна замена метакода i на j. Интенсивность этого процесса $b_{ij}^{(k)}$, $l = k$ (*матрица предпочтений*). Возможна диффузия метакода за границы Ξ_k. Интенсивность диффузии метакода 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 метокода $i \rightarrow j$. Этот процесс опишем *матрицей модификаций* a_{ij}.

Универсальному метакоду $D_u = \{d_1, d_2,\ldots,d_M\}$соответствует устойчивое стационарное состояние 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 систе-

мы должны проектироваться и эксплуатироваться как распределенные системы поддержки инжиниринга.

Итак, в терминологии порождающего программирования [5] процесс проектирования адаптивной программной системы со слоистой архитектурой будет включать следующие процессы.

Методология прикладной инженерии сводится к определению процесса регенерации программной системы, управляемой метакодом.

Методология инженерии предметной области включает три процесса: (а) модификация характеристической модели, (б) модификация порождающей модели и (в) конструирование (или рефакторинг) компонентных систем. Генеративные свойства порождающей модели инкапсулированы в объекте метакод. Пример реализации этой методологии в среде динамического программирования Smalltalk представлен в работе автора [2].

С другой стороны, как было показано выше, эти три процесса должны быть реализованы как распределенная система поддержки программного продукта. Таким образом, методология проектирования адаптивных систем, наряду с традиционными для порождающего программирования разделами инженерии, должна включать также методологию проектирования многоагентной системы.

Параметрами проектирования выступают внешние факторы МАС, которые накладывают разного рода ограничения на характер обмена метакодами. Шаблон проектирования, сформулированный выше, призван обеспечить необходимые свойства онтологии: существенность, непротиворечивость, декларативность, расширяемость.

Обязательное использование шаблона МАС делает более удобным использование методологии нисходящего проектирования.

Нисходящее проектирования МАС можно представить в виде следующей цепочки: «выбор социальных критериев для характеризации сообщества МАС – определение типа искусственного сообщества – синтез структуры МАС – выбор типов агентов – проектирование архитектуры агента» [4]. Как правило, задача сводится к определению структуры МАС.

Выбор социальных критериев. 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. Социальные критерии будем определять исходя из базовых типов взаимодействия агентов [6].

Вернемся к динамической системе, описывающую эволюцию онтологий в МАС. Конечное состояние среды определяется набором устойчивых стационарных состояний. Однако их достижение может потребовать асимптотически большого 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» – общий метакод в границах Ξ_k и разный для разных $k \in K$ и (б) «полоса» – общий для всех Ξ_k универсальный метакод D_u. И в том и в другом случае метакоды равноценны с точки зрения прагматики. Однако последний вариант метакода 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.

Проектируемая среда поддержки продукта будет находиться между этими двумя крайними случаями. Задача проектирования среды поддержки продукта сводится к определению ограничений на коэффициенты матрицы предпочтений и модификаций, так, чтобы достигалось некоторое заданное квазистационарное состояние.

Определение типа искусственного сообщества. Для базовых типов взаимодействия агентов выбирают следующие критерии: а) совместимость целей агентов; б) потребность в чужом опыте; в) совместное использование ресурсов [6]. Тип сообще-

ства выбирается по наличию или отсутствию этих признаков. Это дает восемь базовых типов: координируемое сотрудничество, простое сотрудничество, непродуктивное сотрудничество, безразличие (независимость), коллективное соперничество за ресурсы, чистое коллективное (командное) соперничество, индивидуальное соперничество за ресурсы, чистое индивидуальное соперничество.

Синтез структуры МАС предполагает определение функционально-структурной единицы и определение протоколов. Для дополнительных агентов протоколы определяются, для основных – изменяются или дополняются. Функционально-структурные единицы будут следующие: Заказчик–Исполнитель, Заказчик–Координатор–Исполнитель, Заказчик – Субординатор – Исполнитель и Заказчик – Субординатор – Координатор – Исполнитель.

Подбор типов виртуальных агентов. В зависимости от типа выбранной структуры определяем дополнительных агентов; в нашем случае их два: координатор (доска объявлений) и субординатор.

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 $\{I_1, I_2\}$, $I_1=(f_0, f_1)$, $I_2=(f_0, f_2)$. Распределим пары по группам согласно их интерфейсам, фактор-множество будет $\{ I_1|\Xi_1, I_2|\Xi_2\}$.

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

Пусть в начальном состоянии все пары имеют метакод D_1. Проследим за изменением метакода одной пары $S(m_\mu,s_\mu)$, входящей, например, в группу Ξ_1. Если в Ξ_1 уже существуют адаптированные системы и их метакод доступен (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 метакод 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 универсального метакода 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. В шаблон МАС следует внести следующие изменения: (а) добавить агента-координатора (посредника) и определить его протокол, (б) переопределить протоколы исполнителя и заказчика так, чтобы вместо поддержки непосредственных коммуникаций, передача информации происходила через посредника. Определение архитектуры посредника сводится к использованию технологии доски объявлений.

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

В заключение приведем эталонную модель проектирования. Адаптивная программная система представляет собой автоинжиниринговую систему с распределенной средой поддержки продукта. Экземпляры системы расположены в узлах сети. Программная система функционирует в системе динамического программирования типа CLOS, Python, Open Java. Адаптивная система в течение определенного времени (порционный метод регенерации) производит рекомпозицию своей структуры таким образом, чтобы ее функции соответствовали измененной спецификации.

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. Тем самым имеет место три параллельных процесса: (а) разработка требований, (б) эксплуатация и (в) сопровождение.

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, автоматизирован. Точка соприкосновения лежит на соответствие сценариев (неформальное описание) с их спецификацией (формальное описание). Для большинства функциональных требований может быть зафиксирован некоторый список дескрипторов, которые используются для наименования и раскрытия семантики. Обновление словаря или базы знаний онтологии требований – функция среды поддержки продукта.

Сопровождение включает (а) коммуникации посредством МАС, (б) получение метакода и выполнение операции соединения метакодов (при необходимости) и (в) контроль регенерации системы.

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 метакода.

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

Выводы

В статье рассмотрена модель адаптивной программной системы, архитектура которой 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.

Методология проектирования адаптивной программной системы состоит из двух аспектов: методологии проектирования порождающей модели домена программного продукта и методологии нисходящего проектирования многоагентной системы среды поддержки.

В методологии проектирования МАС, в целях обеспечения целостности онтологий, необходимо использовать шаблон проектирования. Шаблон включает унифицированные программные агенты четырех основных типов и основан на одноуровневой архитектуре МАС. Задача проектирования сводится к определению структуры МАС. Структура определяется либо введением дополнительных агентов, либо изменением спецификаций протокола переговоров, либо их сочетанием. Параметрами проектирования структуры МАС являются внешние по отношению к МАС факторы, которые определяют тип вторичного сообщества виртуальных агентов.

Следующим шагом развития методики может стать перенос концепций проектирования на методы композиционной адаптации программ, использующих платформу промежуточного слоя. Другой путь сближения с практическим программированием возможен на основе развития платформ агентного программирования, приближающих среду программного продукта к языкам, допускающим динамическую рекомпозицию.

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., *Структурная адаптация системы управления стеком* / Современные проблемы информатизации в

анализе и синтезе технологических и программно – телекоммуникационных систем: Сб. трудов, Вып.13. – Воронеж, «Научная книга», 2008. С.343-345
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.