ИНФОРМАТИКА, ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И УПРАВЛЕНИЕ

DOI: 10.18698/0236-3933-2016-3-65-87

УДК 004.5

Верификация свойств интеллектуальных интерфейсов в логике тайлов

В.В. Девятков

МГТУ им. Н.Э. Баумана, Москва, Российская Федерация
e-mail: deviatkov@bmstu.ru

Рассмотрено развитие методики формальной верификации свойств мультимодальных интеллектуальных интерфейсов, обеспечивающих естественное интуитивное взаимодействие информационных систем с человеком. В качестве языка для решения задач формальной верификации интерфейсов выбрана логика тайлов, а в качестве языка для формулировки этих свойств — временна́я модальная логика. Обоснован такой выбор, рассмотрены принципы верификации (доказательства) свойств интерфейсов как проверки правильности взаимодействия агентов в мультиагентной системе. Автоматизацию проверки правильности взаимодействия предложено осуществлять логическими программами, получаемыми в результате перехода от описания свойств интерфейсов в логике тайлов и требований правильности взаимодействия на языке модальной логики к программе проверки правильности на языке логического программирования Visual Prolog. Методика формальной верификации свойств мультимодальных интеллектуальных интерфейсов проиллюстрирована на примере получения описания поведения агентов в логике тайлов, свойства обязательной реакции в модальной логике и логической программой для случая трех взаимодействующих агентов (руководителя и двух исполнителей). Рассмотрены перспективы использования предлагаемой методики для других свойств и приложений.

Ключевые слова: *свойства интеллектуальных интерфейсов, логика тайлов, модальная логика, верификация свойств, интеллектуальные агенты, язык логического программирования Visual Prolog.*

Verification of Intelligent Interface Properties in the Tiles Logic

V.V. Devyatkov

Bauman Moscow State Technical University, Moscow, Russian Federation
e-mail: deviatkov@bmstu.ru

In this work we developed the methodology for formal verification of properties of multimodal intelligent interfaces, providing a natural intuitive interaction of information systems with the person. We selected the tiles logic as the language for solving problems of formal verification of interfaces and temporal modal logic as the

language for formulating these properties. The article explains the choice, considers the principles of verification (proof) of the interfaces properties as the validation of the agents interaction in the multi-agent system. To automate the validation of the agents interaction, we use the logic programs obtained as a result of the transition from the agents interaction description in the tiles logic and requirements to proper interaction in modal logic language to the validation program in logic programming language Visual Prolog. The methodology for formal verification of properties of multimodal intelligent interfaces is illustrated by the description of the agents behavior in the tiles logic, interfaces properties in the modal logic and logic program in the case of the three interacting agents (the leader and two executors). Of particular interest are the prospects of using the proposed methodology for other properties and applications.

Keywords: *intelligent interface properties, tiles logic, modal logic, verification of properties, intelligent agents, logic programming language Visual Prolog.*

Введение. Свойства мультимодальных интеллектуальных интерфейсов, реализация которых обеспечивает естественное «безбарьерное» взаимодействие пользователя с машиной, рассмотрены в работе [1]. Какие из этих свойств необходимо воплощать, зависит от конкретной системы, где требуется соответствующий интерфейс. В любом случае, если какое-либо свойство интерфейса должно быть реализовано, то, во-первых, требуется его точная спецификация, а во-вторых, возможность формальной проверки этой спецификации применительно к интерфейсу конкретной системы. Поскольку речь идет о программной системе, упомянутая спецификация свойств интерфейса и их формальная проверка могут рассматриваться как специфическая область верификации программных систем в целом. В настоящее время существует три основных подхода к верификации программных систем: 1) тестирование наличия у системы требуемых свойств; 2) формальное доказательство наличия свойств в рамках некоторого исчисления; 3) моделе-ориентированное доказательство наличия свойств. Различие этих подходов заключается в следующем.

Тестирование наличия у системы требуемых свойств. При тестировании на вход модели подаются заранее подготовленные и представленные в каком-либо языке последовательности действий (восприятий и реакций агентов) и проверяется, проявляет ли модель ожидаемые свойства. Недостаток тестирования: мощность множества тестирующих последовательностей действий (нитей) может быть слишком велика, что приводит к комбинаторному взрыву и вычислительным проблемам тестирования. Кроме того, множество тестирующих последовательностей формируется экспертом и нет никакой гарантии полноты этого множества для верификации.

Формальное доказательство наличия свойств в рамках некоторого исчисления. При формальном доказательстве эти свойства формулируются в виде теорем (целей) на языке соответствующего исчисления, которые затем доказываются (выводятся) из начальных знаний (аксиом), представленных на том же языке с использованием специальных правил

вывода. Если стратегия вывода полна, то успех доказательства наличия свойств зависит от полноты аксиоматизации начальных знаний. Кроме того, полная стратегия вывода может оказаться вычислительно сложной. Однако главное, что ограничивает использование исчислений, — сложность интерпретации и аксиоматизации на языке этих исчислений необходимых сведений участниками процесса верификации, которые, как правило, не являются специалистами в области исчислений.

Моделе-ориентированное доказательство наличия свойств. В отличие от формального доказательства в рамках исчисления моделе-ориентированное доказательство наличия свойств использует модели архитектуры и поведения системы, в которой реализуется интерфейс. Если разработчик этих моделей полагает, что, создавая архитектуру и описывая поведение системы, он выполнил все предъявляемые к ним требования, то для верификации модели остается только доказать, что ее архитектура и поведение этим требованиям удовлетворяют. Для подобного доказательства также применяют правила вывода, но семантика этих правил существенно ориентирована на привычные для тех или иных приложений понятия и смыслы.

Настоящая работа основана на моделе-ориентированном доказательстве свойств систем. Одни из самых популярных моделей, используемых для этих целей, — процессные модели описания архитектуры и поведения систем. Прародителем класса таких моделей является, например, язык процессных выражений процессной алгебры (пи-исчисления) Р. Милнера [2]. Процессные алгебры предлагают достаточно естественный путь описания параллельных систем, состоящих из агентов, которые взаимодействуют по общим каналам. Описание динамического поведения агентов обычно использует операционную семантику, представляемую в виде правил. К сожалению, языки процессных алгебр хотя и изящны с математической точки зрения, но зачастую слишком абстрактны для применения, а управление правилами вывода в этих языках требует дополнительных усилий, не всегда следующих непосредственно из контекста.

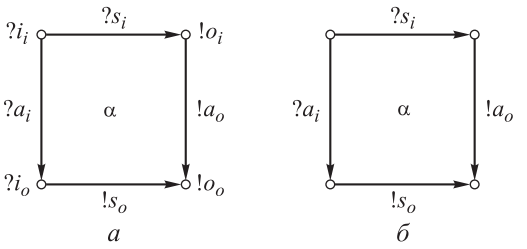
В связи с этим популярным является использование так называемых языков архитектурного описания (Architecture Description Language, ADL), которые позволяют на разных уровнях абстракции описывать структуру и поведение систем [3]. В настоящее время предложено довольно большое число различных языков архитектурного описания [4–7], рассчитанных, прежде всего, на описание и проверку программных систем.

В настоящей статье для описания поведения систем и доказательства свойств их интерфейсов предложено применять формальные системы, основанные на правилах — тайлах (*tile*). На русский язык термин *tile* переводится как черепица, плитка, изразец, что не очень естественно использовать для рассматриваемых целей. Поэтому здесь будем использовать англицизм «логика тайлов» (*tile logic*). Принципы доказательства

каких-либо свойств информационных систем на основе логик тайлов известны достаточно давно [8]. Для того чтобы логику можно было использовать для того или иного приложения, необходимо создать множество тайлов, соответствующих этому приложению.

В рассматриваемом случае приложением является проверка (доказательство) свойств интеллектуальных интерфейсов. Далее будут приведены основные понятия, связанные с тайлами. Затем на простом примере одного из свойств интеллектуальных интерфейсов, называемым свойством обязательной реакции [1], будет введено необходимое для примера множество тайлов. Будет показано, как это множество используется для доказательства свойства обязательной реакции, сформулированного на языке модальной логики. В заключение приведены направления дальнейших работ в области проверки свойств интеллектуальных интерфейсов.

Тайлы и операции над ними. Достоинства логики тайлов связаны со структурой ее правил вывода, каждое из которых может быть представлено в виде тайла (рис. 1, *а*).



а *б*

Рис. 1. Структуры тайла (*а*) и секвента (*б*):

α — имя тайла; *?*i_i — начальный входной интерфейс тайла; *?*s_i — начальная конфигурация тайла; *?*o_i — начальный выходной интерфейс тайла; *!*i_o — конечный входной интерфейс тайла; *!*s_o — конечная конфигурация тайла; *!*o_o — конечный выходной интерфейс тайла; *?*a_i — восприятие тайла; *!*a_o — реакция тайла

Тайл позволяет описывать поведение частично определенных компонентов системы, содержащих переменные и называемых конфигурациями. Это поведение описывается в терминах возможных взаимодействий с внешней или внутренней средой. Поведение системы выглядит как скоординированное взаимодействие отдельных тайлов. Восприятия и реакции также называют действиями. Интерфейсы тайла задают форматы его взаимодействия с внешней средой. Тайл α (см. рис. 1, *а*) в ответ на восприятие *?*a_i задает преобразование начальной конфигурации *?*s_i в конечную конфигурацию *!*s_o, производя реакцию *!*a_o. Такое преобразование возможно тогда, когда тайл не только осуществил восприятие *?*a_i, но и когда форматы данных внешней среды тайла удовлетворяют форматам интерфейсов тайла.

Тайлы, у которых все интерфейсы одинаковы, называют секвентами (рис. 1, *б*). Обозначения интерфейсов в секвентах опускают. Секвент обычно изображают не графически, а в следующем виде:

$$\alpha: ?s_i \frac{?a_i}{!a_o} !s_o.$$

Тайлы могут иметь горизонтальную, вертикальную и параллельную композиции для задания более сложного поведения систем. Операцию горизонтальной композиции тайлов (рис. 2) обозначают символом «*».

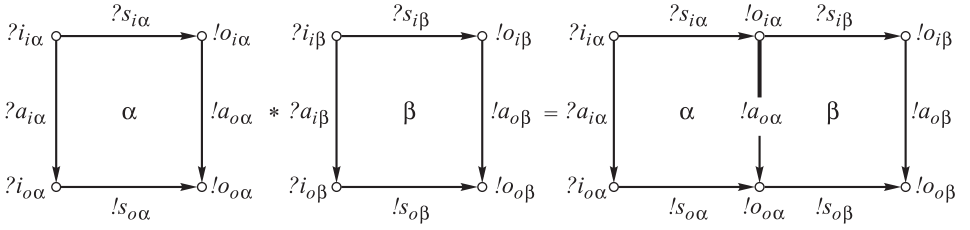


Рис. 2. Горизонтальная композиция тайлов

Горизонтальная композиция двух тайлов α и β возможна только при условии, что $!a_{o\alpha} = !a_{i\beta}$, $!o_{i\alpha} = ?i_{i\beta}$, $!o_{o\alpha} = ?i_{o\beta}$. Смысл горизонтальной композиции тайлов заключается в моделировании формирования более сложных конфигураций $?s_{i\alpha}; ?s_{i\beta}$ и $!s_{o\alpha}; !s_{o\beta}$ из более простых при условии выполнения равенств $!a_{o\alpha} = !a_{i\beta}$, $!o_{i\alpha} = ?i_{i\beta}$, $!o_{o\alpha} = ?i_{o\beta}$.

Горизонтальную композицию двух секвентов α и β представляют правилом

$$\frac{\alpha: ?s_{i\alpha} \frac{?a_{i\alpha}}{!a_{o\alpha}} !s_{o\alpha},\ \beta: ?s_{i\beta} \frac{?a_{i\beta}}{!a_{o\beta}} !s_{o\beta},\ !a_{o\alpha} = ?a_{i\beta}}{\alpha * \beta: ?s_{i\alpha}; ?s_{i\beta} \frac{?a_{i\alpha}}{!a_{o\beta}} !s_{o\alpha}; !s_{o\beta}}.$$

Операцию вертикальной композиции (рис. 3) обозначают символом «•».

Вертикальная композиция моделирует последовательное преобразование конфигурации $?s_{i\alpha}$ в конфигурацию $!s_{o\beta}$ сначала с помощью пары восприятие $?a_{i\alpha}$, реакция $!a_{o\alpha}$, а затем пары восприятие $?a_{i\beta}$, реакция $!a_{o\beta}$, и при условии, что $!s_{o\alpha} = ?s_{i\beta}$, $!i_{o\alpha} = ?i_{i\beta}$, $!o_{o\alpha} = ?o_{i\beta}$.

Вертикальную композицию двух секвентов α и β представляют правилом

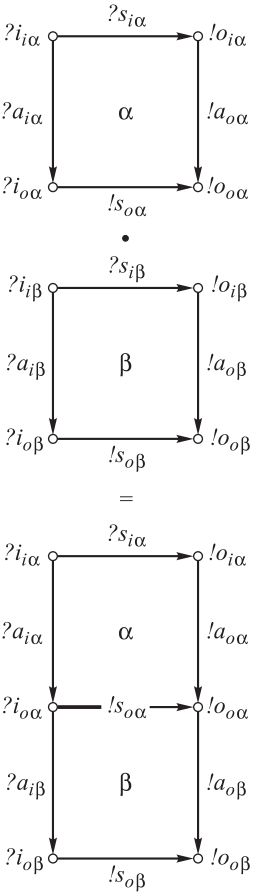


Рис. 3. Вертикальная композиция тайлов

$$\frac{\alpha\colon ?s_{i\alpha}\,\dfrac{?a_{i\alpha}}{!a_{o\alpha}}\,!s_{o\alpha},\ \beta\colon ?s_{i\beta}\,\dfrac{?a_{i\beta}}{!a_{o\beta}}\,!s_{o\beta},\ !s_{o\alpha}=?s_{i\beta}}{\alpha\bullet\beta\colon ?s_{i\alpha}\,\dfrac{?a_{i\alpha};\ ?a_{i\beta}}{!a_{o\alpha};\ !a_{o\alpha}}\,!s_{o\beta}}.$$

Операцию параллельной композиции (рис. 4) обозначают символом «⊗».

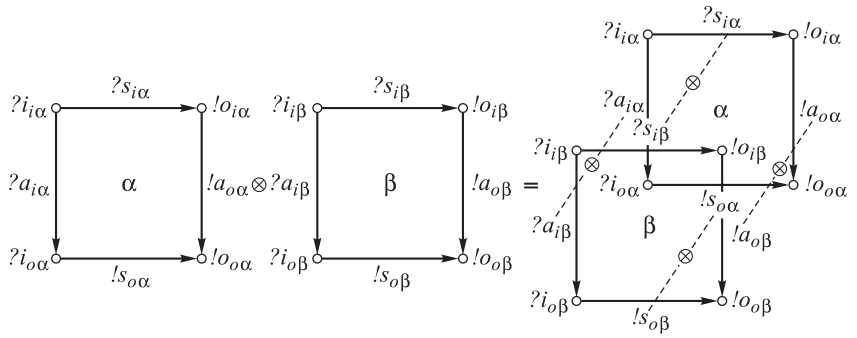


Рис. 4. Параллельная композиция тайлов

Параллельная композиция двух тайлов α и β возможна только при условии параллельного выполнения соответственно пар восприятий $?a_{o\alpha}$, $?a_{i\beta}$, реакций $!a_{o\alpha}$, $!a_{i\beta}$, начальных конфигураций $?s_{i\alpha}$, $?s_{i\beta}$, конечных конфигураций $!s_{o\alpha}$, $!s_{o\beta}$.

Параллельную композицию двух секвентов α и β представляют правилом

$$\frac{\alpha\colon ?s_{i\alpha}\,\dfrac{?a_{i\alpha}}{!a_{o\alpha}}\,!s_{o\alpha},\ \beta\colon ?s_{i\beta}\,\dfrac{?a_{i\beta}}{!a_{o\beta}}\,!s_{o\beta}}{\alpha\otimes\beta\colon ?s_{i\alpha}\otimes ?s_{i\beta}\,\dfrac{?a_{i\alpha}\otimes ?a_{i\beta}}{!a_{o\alpha}\otimes !a_{o\alpha}}\,!s_{o\alpha}\otimes !s_{o\beta}}.$$

Та или иная логика тайлов получается в результате введения множества базовых тайлов и некоторой совокупности вспомогательных тайлов, над которыми могут совершаться введенные выше операции, приводящие к различным преобразованиям тайлов, в частности, в целях доказательства свойств интеллектуальных интерфейсов.

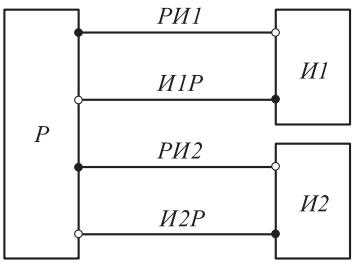


Рис. 5. Простая архитектура, состоящая из руководителя и исполнителей

Принципы секвентного описания поведения взаимодействующих агентов. Для демонстрации принципов секвентного описания поведения взаимодействующих агентов воспользуемся простой архитектурой, состоящей из агента-руководителя и двух агентов-исполнителей (рис. 5), которые назовем руководителем и исполните-

лями. Каждый *i*-й агент-исполнитель имеет два канала: входной канал *РИi*, являющийся выходным для руководителя, и выходной канал *ИiР*, являющийся входным для руководителя.

Во входной канал исполнителя руководитель может помещать сообщение, которое может забирать исполнитель. В свой выходной канал исполнитель может помещать сообщение для руководителя. Каждый исполнитель после получения сообщения от руководителя осуществляет внутреннее поведение в соответствии с собственными убеждениями. В рамках настоящей статьи это поведение не рассмотрено. Взаимодействие исполнителей и руководителя начинается после получения от руководителя уведомления о начале взаимодействия, которое одновременно рассылается всем исполнителям. Если хотя бы один исполнитель дал согласие на взаимодействие, то оно продолжается. В противном случае взаимодействие прекращается. Руководитель прекращает взаимодействие путем отправки соответствующего сообщения исполнителям. Описанное взаимодействие агентов будем представлять секвентами, структура которых рассмотрена выше.

Рассмотрим состояния (конфигурации) каналов, которые использует руководитель:

1) все каналы открыты, и руководитель готов начать взаимодействие с исполнителями

$$\textbf{\textit{Готовность}} = (РИ1(открыт), И1Р(открыт), РИ2(открыт), И2Р(открыт));$$

2) в выходные каналы руководителя поступают приглашения на взаимодействие всем исполнителям

$$\textbf{\textit{Приглашение}} = (РИ1(приглашение), И1Р(открыт), РИ2(приглашение), И2Р(открыт));$$

3) в выходные каналы исполнителей поступают подтверждения о получении приглашения на взаимодействие

$$\textbf{\textit{Принятие}} = (РИ1(приглашение), И1Р(приглашен), РИ2(приглашение), И2Р(приглашен));$$

4) в выходные каналы руководителя поступают сообщения о начале выполнения заданий исполнителями

$$\textbf{\textit{Управление}} = (РИ1(задание), И1Р(приглашен), РИ2(задание), И2Р(приглашен));$$

5) в выходные каналы исполнителей поступают сообщения о согласии на выполнение ими заданий

$$\textbf{\textit{Выполнение}} = (РИ1(задание), И1Р(согласен), РИ2(задание), И2Р(согласен));$$

6) в выходные каналы руководителя поступают сообщения об ожидании выполнения заданий

Окончание = (*РИ1*(*ожидание*), *И1Р*(*согласен*), *РИ2*(*ожидание*), *И2Р*(*согласен*));

7) в выходные каналы исполнителей поступают сообщения о завершении выполнения заданий

Завершение = (*РИ1*(*ожидание*), *И1Р*(*завершено*), *РИ2*(*ожидание*), *И2Р*(*завершено*));

8) во входные каналы исполнителей поступают сообщения о закрытии этих каналов

Закрытие = (*РИ1*(*закрыт*), *И1Р*(*завершено*), *РИ2*(*закрыт*), *И2Р*(*завершено*)).

Руководитель выполняет следующие действия:

1) поместить сообщения *сообщение1, сообщение2* в каналы *РИ1*, *РИ2*

Поместить(*РИ1*(*сообщение1*), *РИ2*(*сообщение2*));

2) получить сообщения *сообщение1, сообщение2* из каналов *И1Р*, *И2Р*

Получить(*И1Р* (*сообщение1*), *И2Р* (*сообщение2*)).

Используя введенные состояния каналов и действия руководителя, его поведение может быть описано приведенными ниже секвентами.

Подготовка:

$$\textbf{\textit{Готовность}}\ \frac{\textit{Получить}(\textit{И1Р}(\textit{открыт}),\ \textit{И2Р}(\textit{открыт}))}{\textit{Поместить}(\textit{РИ1}(\textit{приглашение}),\ \textit{РИ2}(\textit{приглашение}))}\ \textbf{\textit{Приглашение}}.$$

Воспринимая состояние каналов ***Готовность***, руководитель решает поместить сообщение *приглашение* во все свои выходные каналы. В результате состоянием каналов становится ***Приглашение***.

Реакция:

$$\textbf{\textit{Приглашение}}\ \frac{\textit{Поместить}(\textit{РИ 1}(\textit{приглашение}),\textit{РИ 2}(\textit{приглашение}))}{\textit{Получить}(\textit{И 1Р}(\textit{приглашен}),\ \textit{И2Р}(\textit{приглашен}))}\ \textbf{\textit{Принятие}}.$$

Воспринимая состояние каналов ***Приглашение***, руководитель ожидает реакции исполнителей. После их реакции состоянием каналов становится ***Принятие***.

Уведомление:

$$\textbf{\textit{Принятие}}\ \frac{\textit{Получить}(\textit{И1Р}(\textit{приглашен}),\textit{И2Р}(\textit{приглашен})}{\textit{Поместить}\,(\textit{РИ1}(\textit{задание}),\ \textit{РИ2}(\textit{задание}))}\ \textbf{\textit{Управление}}.$$

Воспринимая состояние каналов ***Принятие*** руководитель помещает задания во входные каналы исполнителей. Состоянием каналов становится ***Управление***.

Информирование:

$$\textbf{\textit{Управление}}\frac{\textit{Поместить}(\textit{РИ1}(\textit{задание}),\textit{РИ2}(\textit{задание}))}{\textit{Получить}(\textit{И1Р}(\textit{согласен}),\ \textit{И2Р}(\textit{согласен}))}\textbf{\textit{Выполнение}}.$$

Воспринимая состояние каналов ***Управление***, руководитель получает от исполнителей сообщение о согласии на выполнение заданий. Состоянием каналов становится ***Выполнение***.

Заключение:

$$\textbf{\textit{Выполнение}}\frac{\textit{Получить}(\textit{И1Р}(\textit{согласен}),\ \textit{И2Р}(\textit{согласен}))}{\textit{Поместить}(\textit{РИ1}(\textit{ожидание}),\textit{РИ2}(\textit{ожидание}))}\textbf{\textit{Окончание}}.$$

Воспринимая состояние каналов ***Выполнение***, руководитель получает от всех исполнителей сообщение о согласии на выполнение заданий. Состоянием каналов становится ***Окончание***.

Сокрытие:

$$\textbf{\textit{Окончание}}\frac{\textit{Поместить}(\textit{РИ1}(\textit{ожидание}),\textit{РИ2}(\textit{ожидание}))}{\textit{Получить}(\textit{И1Р}(\textit{завершено}),\ \textit{И2Р}(\textit{завершено}))}\textbf{\textit{Завершение}}.$$

Воспринимая состояние каналов ***Окончание***, руководитель получает сообщения о завершении заданий. Состоянием каналов становится ***Завершение***.

Прекращение:

$$\textbf{\textit{Завершение}}\frac{\textit{Получить}(\textit{И1Р}(\textit{завершено}),\ \textit{И2Р}(\textit{завершено}))}{\textit{Поместить}(\textit{РИ1}(\textit{закрыт}),\textit{РИ2}(\textit{закрыт}))}\textbf{\textit{Закрытие}}.$$

Воспринимая состояние каналов ***Завершение***, руководитель направляет всем исполнителям сообщение о закрытии их входных каналов. Состоянием каналов становится ***Закрытие***.

Состояниями (конфигурациями) каналов, которые доступны *i*-му исполнителю, являются следующие состояния:

Иi-Готов=(*РИi*(*открыт*), *ИiР*(*открыт*));

Иi-Приглашен=(*РИi*(*приглашение*), *ИiР*(*открыт*));

Иi-Принято=(*РИi*(*приглашение*), *ИiР*(*приглашен*));

Иi-Управление=(*РИi*(*задание*), *ИiР*(*приглашен*));

Иi-Выполнено=(*РИi*(*задание*), *ИiР*(*согласен*));

Иi-Завершено=(*РИi*(*ожидание*), *ИiР*(*согласен*));

Иi-Закрыт=(*РИi*(*ожидание*), *ИiР*(*закрыт*));

Действиями *i*-го исполнителя являются:

1) поместить *сообщение* в выходной канал *ИiР* ***Иi-Поместить***(*ИiР*(*сообщение*));
2) получить *сообщение* из входного канала *РИi* ***Иi-Получить***((*РИi*(*сообщение*)).

Аналогично описанию поведения руководителя каждый исполнитель может вести себя в соответствии с перечисленными ниже секвентами.

И*i*-Подготовка:

$$\textbf{\textit{Иi-Готов}}\ \frac{Иi\text{-}Поместить(ИiР(открыт))}{Иi\text{-}Получить(РИi(приглашение))}\ \textbf{\textit{Иi-Приглашен}}.$$

И*i*-Реакция:

$$\textbf{\textit{Иi-Приглашен}}\ \frac{Иi\text{-}Получить(РИi(приглашение))}{Иi\text{-}Поместить(ИiР(приглашен))}\ \textbf{\textit{Иi-Принято}}.$$

И*i*-Уведомление:

$$\textbf{\textit{Иi-Принято}}\ \frac{Иi\text{-}Поместить(ИiР(приглашен))}{Иi\text{-}Получить(РИi(задание))}\ \textbf{\textit{Иi-Управление}}.$$

И*i*-Информирование:

$$\textbf{\textit{Иi-Управление}}\ \frac{Иi\text{-}Получить(РИi(задание))}{Иi\text{-}Поместить(ИiР(согласен))}\ \textbf{\textit{Иi-Выполнено}}.$$

***Иi*-Завершение:**

$$\textbf{\textit{Иi-Выполнено}}\ \frac{Иi\text{-}Поместить(ИiР(согласен))}{Иi\text{-}Получить(РИi(ожидание))}\ \textbf{\textit{Иi-Завершено}}.$$

И*i*-Закрытие:

$$\textbf{\textit{Иi-Завершено}}\ \frac{Иi\text{-}Получить(РИi(ожидание))}{Иi\text{-}Поместить(ИiР(закрыт))}\ \textbf{\textit{Иi-Закрыт}}.$$

Формулировка свойств интеллектуального интерфейса на языке модальной логики. Для формулировки требуемых свойств интеллектуальных интерфейсов в работе [1] предложено использовать широко применяемый для подобных целей язык временно́й модальной логики. Этот язык хорошо изучен и его применение для верификации также широко известно, в настоящей статье не будем уделять ему слишком много внимания. Рассмотрим принципы формулировки свойств интеллектуальных интерфейсов с привязкой к понятиям тайлов (секвентов), с помощью операций над которыми эти свойства будут проверяться. В качестве примера одного из таких свойств выберем простейшее свойство обязательной реакции. Суть этого свойства применительно к взаи-

модействию руководителя и некоторого исполнителя состоит в следующем. На каждую реакцию руководителя, в результате которой в его выходных каналах, являющихся одновременно входными каналами для какого-либо исполнителя, окажутся сообщения, предназначенные для последнего, исполнитель обязательно должен получить эти сообщения и реагировать на них помещением в свои выходные каналы, являющиеся одновременно входными для руководителя, ответных сообщений.

На языке модальной логики формулировка этого свойства для рассматриваемого примера может выглядеть так:

$$\begin{aligned}
&\Box[sequent(\alpha_{P_1}, ?s_{P_1},\ f(Поместить(РИ1(x), РИ2(y))),\ !s_{P_1})\\
&\wedge sequent(\alpha_{И1_1}, ?s_{И1_1},\ f(И1\text{-}Получить(РИ1(x)),\ !s_{И1_1})\\
&\wedge sequent(\alpha_{И2_1}, ?s_{И2_1},\ f(И2\text{-}Получить(РИ1(x)),\ !s_{И2_1})\\
&\wedge sequent(\alpha_{И1_2}, ?s_{И1_2},\ f(И1\text{-}Поместить(И1P(v)),\ !s_{И1_2},)\\
&\wedge sequent(\alpha_{И2_2}, ?s_{И2_2},\ f(И2\text{-}Поместить(И2P(w)),\ !s_{И2_2})\\
&\wedge sequent(\alpha_{P_2}, ?s_{P2},\ f(Получить(И1P(v), И2P(w)),\ !s_P)].
\end{aligned}$$

Здесь каждый предикат $sequent(\alpha, ?s_i,\ f(!a_o),\ !s_o))$ представляет секвент $\alpha: ?s_i \dfrac{?a_i}{!a_o} !s_o$, где $f(?a_i,\ !a_o)$ — функтор, структура которого может быть достаточно сложной в зависимости от структуры реакции $!a_o$ в конкретном приложении; $?s_i$, $!s_o$ — конфигурации.

Процедура проверки свойств интеллектуального интерфейса. Формулировка любого свойства интеллектуального интерфейса на языке модальной логики — утверждение, которое требует определенной процедуры доказательства. В частности, свойство обязательной реакции предполагает, что в любом макросостоянии, являющемся совокупностью состояний руководителя и исполнителей, это утверждение должно быть истинным. Следовательно, процедура доказательства может быть разбита на два этапа: 1) генерация всех возможных макросостояний; 2) проверка в каждом макросостоянии истинности свойства обязательной реакции. Генерация макросостояний может быть осуществлена с помощью правила параллельной композиции, которое для рассматриваемого примера будет иметь вид

$$\frac{\alpha: ?s_{i\alpha} \dfrac{?a_{i\alpha}}{!a_{o\alpha}} !s_{o\alpha}, \beta: ?s_{i\beta} \dfrac{?a_{i\beta}}{!a_{o\beta}} !s_{o\beta}, \delta: ?s_{i\delta} \dfrac{?a_{i\delta}}{!a_{o\delta}} !s_{o\delta}}{\alpha\otimes\beta\otimes\delta: ?s_{i\alpha}\otimes ?s_{i\beta}\otimes ?s_{i\delta} \dfrac{?a_{i\alpha}\otimes ?a_{i\beta}\otimes ?a_{i\delta}}{!a_{o\alpha}\otimes !a_{o\beta}\otimes !a_{o\delta}} !s_{o\alpha}\otimes !s_{o\beta}\otimes !s_{o\delta}}.$$

Это правило позволяет генерировать все макросостояния $\alpha\otimes\beta\otimes\delta$, требуемые для проверки свойства обязательной реакции. В рамках настоя-

щей статьи генерация всех макросостояний потребует слишком много места, поэтому покажем только два шага такой генерации:

Подготовка:

$$\textbf{\textit{Готовность}}\frac{\textit{Получить}(\textit{И1Р}(\textit{открыт}),\ \textit{И2Р}(\textit{открыт}))}{\textit{Поместить}(\textit{РИ1}(\textit{приглашение}),\ \textit{РИ2}(\textit{приглашение}))}\textit{Приглашение},$$

$$\textbf{И1-Подготовка:}\ \textit{И1-Готов}\frac{\textit{И1-Поместить}(\textit{И1Р}(\textit{открыт}))}{\textit{И1-Получить}(\textit{РИ1}(\textit{приглашение}))}\textit{И1-Приглашен},$$

$$\textbf{И2-Подготовка:}\ \textit{И2-Готов}\frac{\textit{И2-Поместить}(\textit{И2Р}(\textit{открыт}))}{\textit{И2-Получить}(\textit{РИ2}(\textit{приглашение}))}\textit{И2-Приглашен}$$

Подготовка ⊗ И1-Подготовка ⊗ И2-Подготовка :

$$\textbf{\textit{Готовность}} \otimes \textbf{\textit{И1-Готов}} \otimes \textbf{\textit{И2-Готов}}\frac{\begin{array}{c}\textit{Получить}(\textit{И1Р}(\textit{открыт}),\ \textit{И2Р}(\textit{открыт}))\\ \otimes\textit{И1-Поместить}(\textit{И1Р}(\textit{открыт}))\\ \otimes\textit{И2-Поместить}(\textit{И2Р}(\textit{открыт}))\end{array}}{\begin{array}{c}\textit{Поместить}(\textit{РИ1}(\textit{приглашение}),\ \textit{РИ2}(\textit{приглашение}))\\ \otimes\ \textit{И1-Получить}(\textit{РИ1}(\textit{приглашение}))\\ \otimes\ \textit{И2-Получить}(\textit{РИ2}(\textit{приглашение}))\end{array}}\otimes$$

$$\otimes\ \textbf{Приглашение} \otimes \textbf{\textit{И1-Приглашен}} \otimes \textbf{\textit{И1-Приглашен}}$$

$$\textbf{Реакция:}\ \textit{Приглашение}\frac{\textit{Поместить}(\textit{РИ1}(\textit{приглашение}),\textit{РИ2}(\textit{приглашение}))}{\textit{Получить}(\textit{И1Р}(\textit{приглашение}),\textit{И2Р}(\textit{приглашение}))}\textbf{\textit{Принятие}},$$

$$\textbf{И1-Реакция:}\ \textit{И1-Приглашен}\frac{\textit{И1-Получить}(\textit{РИ1}(\textit{приглашение}))}{\textit{И1-Поместить}(\textit{И1Р(приглашен}))}\textbf{\textit{И1-Принято}}$$

$$\textbf{И2-Реакция:}\ \textit{И2-Приглашен}\frac{\textit{И2-Получить}(\textit{РИ2}(\textit{приглашение}))}{\textit{И2-Поместить}(\textit{И2Р}(\textit{приглашен}))}\textbf{\textit{И2-Принято}}$$

Реакция ⊗ И1-Реакция ⊗ И2 -Реакция:

$$\textbf{\textit{Приглашение}} \otimes \textbf{\textit{И1-Приглашен}} \otimes \textbf{\textit{И1-Приглашен}}\frac{\begin{array}{c}\textit{Поместить}(\textit{РИ}\backslash(\textit{приглашение}),\ \textit{РИ2}(\textit{приглашение}))\\ \otimes\ \textit{И1-Получить}(\textit{РИ1}(\textit{приглашение}))\\ \otimes\ \textit{И2-Получить}(\textit{РИ2}(\textit{приглашение}))\end{array}}{\begin{array}{c}\textit{Получить}(\textit{И1Р}(\textit{приглашение}),\textit{И2Р}(\textit{приглашение})\\ \otimes\ \textit{И1-Поместить}(\textit{И1Р}(\textit{приглашен})\\ \otimes\ \textit{И1-Поместить}(\textit{И1Р}(\textit{приглашен}))\end{array}}$$

$$\textit{Принятие} \otimes \textit{И1-Притглашен} \otimes \textit{И2-Приглашен}$$

Очевидно, что для реальных систем большой размерности, включающих n взаимодействующих агентов, каждый из которых связан друг с другом m каналами, число пар наборов восприятий и реакций может быть оценено величиной $O(mn^2)$ макросостояний. При больших числах m и n генерация наборов восприятий и реакций, а также проверка на них требуемых свойств интеллектуального интерфейса вручную трудно осуществима, в связи с чем необходима автоматизация подобной проверки. Перспективный способ автоматизации — использование логического программирования, например, на основе языка логического программирования Visual Prolog [9].

Логическая программа проверки свойства обязательной реакции для рассматриваемого примера может выглядеть следующим образом (здесь названия секвентов, конфигураций, каналов и сообщений представлены на английском языке):

```
domains
rlp = rlp(receive, put)
plr = plr(put, receive)
          receive = receive(p1l, p2l)
                    p1l = p1l(symbol)
                    p2l = p2l(symbol)
          put = put(lp1, lp2)
                    lp1 = lp1(symbol)
                    lp2 = lp2(symbol)
rp1p = rp1p(p1receive, p1put)
pp1r = pp1r(p1put, p1receive)
       p1receive = p1receive(lp11)
      p1put = p1put(p1l1)
              lp11 = lp11(symbol)
              p1l1 = p1l1(symbol)
rp2p = rp2p(p2receive, p2put)
pp2r = pp2r(p2put, p2receive)
        p2receive = p2receive(lp22)
        p2put = p2put(p2l2)
                lp22 = lp22(symbol)
                p2l2 = p2l2(symbol)
predicates
nondeterm sequentRlp (symbol, symbol, rlp, symbol)
nondeterm sequentPlr (symbol, symbol, plr, symbol)
nondeterm sequentRp1p (symbol, symbol, rp1p, symbol)
nondeterm sequentPp1r (symbol, symbol, pp1r, symbol)
nondeterm sequentRp2p (symbol, symbol, rp2p, symbol)
nondeterm sequentPp2r (symbol, symbol, pp2r, symbol)
nondeterm execute(symbol,symbol,symbol,symbol,symbol,symbol)

clauses
sequentRlp(preparation, readiness, rlp(receive(p1l(opened),
p2l (opened)), put(lp1(invitation), lp2(invitation))), invit-
ing).
sequentRlp(notification, accepting, rlp(receive(p1l(invited),
p2l (invited)), put(lp1(quest), lp2(quest))), controlling).
sequentRlp(completion, implementing, rlp(receive(p1l(agreed),
p2l (agreed)), put(lp1(expectation), lp2(expectation))),
     ending).
sequentRlp(cessation, terminating,
rlp(receive(p1l(completed), p2l (completed)),
put(lp1(closed), lp2(closed))), closing).
sequentPlr(reaction, inviting, plr(put(lp1(invitation),
lp2(invitation)), receive(p1l(invited), p2l (invited))),
    accepting).
sequentPlr(information, controlling, plr(put(lp1(quest),
lp2(quest)), receive(p1l(agreed), p2l (agreed))),
    implementing).
```

```
sequentPlr(encapsulation, ending, plr(put(lp1(expectation),
lp2(expectation)), receive(p1l(completed), p2l (completed))),
terminating).
sequentPp1r(p1preparation, p1ready, pp1r(p1put(p1l1
(opened)), p1receive(lp11(invitation))), p1invited).
sequentPp1r(p1notification, p1accepted,
pp1r(p1put(p1l1(invited)), p1receive(lp11(quest))),
p1controlled).
sequentPp1r(p1complition, p1implemented,
pp1r(p1put(p1l1(agreed)), p1receive(lp11(expectation))),
p1ended).
sequentPp1r(p1cessation, p1terminated,
pp1r(p1put(p1l1(completed)), p1receive(lp11(closed))),
p1closed).
sequentRp1p(p1information, p1controlled,
rp1p(p1receive(lp11(quest)), p1put(p1l1(agreed))),
p1implemented).
sequentRp1p(p1encapsulation, p1ended,
rp1p(p1receive(lp11(expectation)), p1put(p1l1 (completed))),
p1terminated).
sequentRp1p(p1reaction, p1invited,
rp1p(p1receive(lp11(invitation)), p1put(p1l1(invited))),
p1accepted).

sequentPp2r(p2preparation, p2ready, pp2r(p2put(p2l2(opened)),
p2receive(lp22(invitation))), p2invited).
sequentPp2r(p2notification, p2accepted,
pp2r(p2put(p2l2(invited)), p2receive(lp22(quest))),
p2controlled).
sequentPp2r(p2complition, p2implemented,
pp2r(p2put(p2l2(agreed)), p2receive(lp22(expectation))),
p2ended).
sequentPp2r(p2cessation, p2terminated,
pp2r(p2put(p2l2(completed)), p2receive(lp22(closed))),
p2closed).
sequentRp2p(p2reaction, p2invited,
rp2p(p2receive(lp22(invitation)), p2put(p2l2(invited))),
p2accepted).
sequentRp2p(p2information, p2controlled,
rp2p(p2receive(lp22(quest)), p2put(p2l2(agreed))),
p2implemented).
sequentRp2p(p2encapsulation, p2ended,
rp2p(p2receive(lp22(expectation)), p2put(p2l2(completed))),
p2terminated).

execute(XL11,XP111,XP221,XL12,XP112,XP222):-
sequentRlp(XL11, S1L11, rlp(receive(p1l(M1L11), p2l(M1L21)),
put(lp1(M2L11), lp2(M2L21))), S2L11),
sequentPp1r(XP111, S1P11, pp1r(p1put(p1l1(M1L11)),
p1receive(lp11(M2L11))), S2P11),
sequentPp2r(XP221, S1P21, pp2r(p2put(p2l2(M1L21)),
p2receive(lp22(M2L21))), S2P21),
sequentPlr(XL12, S1L12, plr(put(lp1(M1L12), lp2(M1L22)),
```

```
receive(p1l(M2L12), p2l(M2L22))), S2L12),
sequentRp1p(XP112, S1P12, rp1p(p1receive(lp11(M1L12)),
p1put(p1l1(M2L12))), S2P12),
sequentRp2p(XP222, S1P22, rp2p(p2receive(lp22(M1L22)),
p2put(p2l2(M2L22))), S2P22),
nl, nl,
write("=============================================="), nl,
write("sequentRlp(XL11, S1L11, rlp(receive(p1l(M1L11),
p2l(M1L21)), put(lp1(M2L11), lp2(M2L21))), S2L11)"), nl,
write("XL11 = ", XL11), write(", S1L11 = ", S1L11), write(",
M1L11 = ", M1L11), write(", M1L21 = ", M1L21),
write(", M2L11 = ", M2L11), write(", M2L21 = ", M2L21),
write(", S2L11 = ", S2L11), nl, nl,
write("sequentPp1r(XP111, S1P11, pp1r(p1put(p1l1(M1L11)),
p1receive(lp11(M2L11))), S2P11)"), nl,
write("XP111 = ", XP111), write(", S1P11 = ", S1P11),
write(", M1L11 = ", M1L11), write(", M2L11 = ", M2L11),
write(", S2P11 = ", S2P11), nl, nl,
write("sequentPp2r(XP221, S1P21, pp2r(p2put(p2l2(M1L21)),
p2receive(lp22(M2L21))), S2P21)"), nl,
write("XP221 = ", XP221), write(", S1P21 = ", S1P21),
write(", M1L21 = ", M1L21), write(", M2L21 = ", M2L21),
write(", S2P21 = ", S2P21), nl, nl,
write("sequentPlr(XL12, S1L12, plr(put(lp1(M1L12),
lp2(M1L22)), receive(p1l(M2L12), p2l(M2L22))), S2L12)"), nl,
write("XL12 = ", XL12), write(", S1L12 = ", S1L12), write(",
M1L12 = ", M1L12), write(", M1L22 = ", M1L22),
write(", M2L12 = ", M2L12), write(", M2L22 = ", M2L22),
write(", S2L12 = ", S2L12), nl, nl,
write("sequentRp1p(XP112, S1P12, rp1p(p1receive(lp11(M1L12)),
p1put(p1l1(M2L12))), S2P12)"), nl,
write("XP112 = ", XP112), write(", S1P12 = ", S1P12),
write(", M1L12 = ", M1L12), write(", M2L12 = ", M2L12),
write(", S2P12 = ", S2P12), nl, nl,
write("sequentRp2p(XP222, S1P22, rp2p(p2receive(lp22(M1L22)),
p2put(p2l2(M2L22))), S2P22)"), nl,
write("XP222 = ", XP222), write(", S1P22 = ", S1P22),
write(", M1L22 = ", M1L22), write(", M2L22 = ", M2L22),
write(", S2P22 = ", S2P22), nl, nl,
write("=============================================="),!,
nl.

goal
execute(XL11,XP111,XP221,XL12,XP112,XP222).
```

Приведенная программа содержит четыре раздела `domains`, `predicates`, `clauses`, `goal`. Раздел `domains` включает в себя описание структуры всех используемых функторов, раздел `predicates` — описание структуры всех используемых предикатов, представляющих

секвенты, с предикатными символами, начинающимися с префикса `sequent`, и правило обязательной реакции, в левой части которого находится целевой предикат `execute(XL11, XP111, XP221, XL12, XP112, XP222)` с предикатным символом `execute`. Термами этого предикатного символа являются переменные, соответствующие именам секвентов, которые должны обязательно использоваться при взаимодействии агентов для выполнения свойства обязательной реакции. В правой части этого правила указывается перечень всех секвентов с этими именами и соответствующими термами.

Раздел `clauses` содержит единственный предикат, каждое истинное значение которого выдает список имен всех секвентов, участвующих в том или ином взаимодействии агентов и не нарушающих свойство обязательной реакции. Кроме имен также выдаются соответствующие этим взаимодействиям значения термов. В связи с ограниченным объемом статьи в конце правила с левой частью `execute(XL11, XP111, XP221, XL12, XP112, XP222)` поставлено отсечение (!), позволяющее выдавать информацию только об одном макросостоянии взаимодействия агентов, правильном с позиции свойства обязательной реакции. В соответствии с приведенной логической программой это будет следующая информация:

```
sequentRlp(XL11, S1L11, rlp(receive(p1l(M1L11), p2l(M1L21)),
put(lp1(M2L11), lp2(M2L21))), S2L11)
XL11 = preparation, S1L11 = readiness, M1L11 = opened,
M1L21 = opened, M2L11 = invitation, M2L21 = invitation,
S2L11 = inviting

sequentPp1r(XP111, S1P11, pp1r(p1put(p1l1(M1L11)),
p1receive(lp11(M2L11))), S2P11)
XP111 = p1preparation, S1P11 = p1ready, M1L11 = opened, M2L11
= invitation, S2P11 = p1invited

sequentPp2r(XP221, S1P21, pp2r(p2put(p2l2(M1L21)),
p2receive(lp22(M2L21))), S2P21)
XP221 = p2preparation, S1P21 = p2ready, M1L21 = opened, M2L21
= invitation, S2P21 = p2invited

sequentPlr(XL12, S1L12, plr(put(lp1(M1L12), lp2(M1L22)),
receive(p1l(M2L12), p2l(M2L22))), S2L12)
XL12 = reaction, S1L12 = inviting, M1L12 = invitation,
M1L22 = invitation, M2L12 = invited, M2L22 = invited,
S2L12 = accepting

sequentRp1p(XP112, S1P12, rp1p(p1receive(lp11(M1L12)),
p1put(p1l1(M2L12))), S2P12)
XP112 = p1reaction, S1P12 = p1invited, M1L12 = invitation,
M2L12 = invited, S2P12 = p1accepted
```

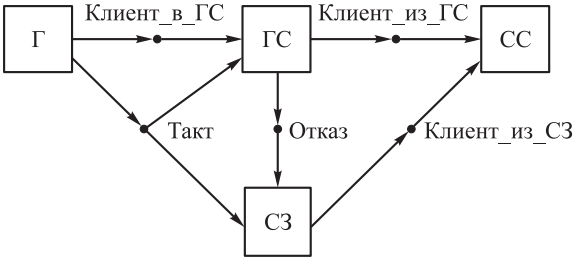
```
sequentRp2p(XP222, S1P22, rp2p(p2receive(1p22(M1L22)),
p2put(p2l2(M2L22))), S2P22)
XP222 = p2reaction, S1P22 = p2invited, M1L22 = invitation,
M2L22 = invited, S2P22 = p2accepted

=====================================================
XL11=preparation, XP111=p1preparation, XP221=p2preparation,
XL12=reaction, XP112=p1reaction, XP222=p2reaction
1 Solution
```

Заключение. Кратко изложены принципы создания методики формальной верификации свойств мультимодальных интеллектуальных интерфейсов на основе использования логики тайлов, временно́й модальной логики и логического программирования. Методика формальной верификации свойств мультимодальных интеллектуальных интерфейсов проиллюстрирована на примере проверки свойства обязательной реакции простейшей системы, состоящей из трех агентов. В этом случае для проверки свойства обязательной реакции достаточно использования параллельной композиции секвентов. Для верификации свойств мультимодальных интеллектуальных интерфейсов больших и сложных систем требуется представлять систему как иерархическое множество независимых процессов (агентов) и каналов связи между этими процессами, которые могут реконфигурироваться. Логика тайлов в наибольшей степени подходит для верификации свойств мультимодальных интеллектуальных интерфейсов больших и сложных систем, поскольку интерфейсы являются базовыми объектами, которыми эта логика способна манипулировать. Различные схемы соединений интерфейсов могут создаваться в логике тайлов вследствие параллельной и горизонтальной композиций, порождающих различные системные конфигурации. Каждая такая конфигурация задает пространственное представление общего состояния системы, состоящего из локальных состояний. Горизонтальная или параллельная композиция также обеспечивает возможность представления контекстной зависимости данных вследствие наличия интерфейсов. Поведение системы в целом определяется множеством тайлов, задающих локальные изменения конфигураций системы, которые являются локальными состояниями системы, в результате вертикальной композиции. Для того чтобы изложенное выше было более понятным, введем следующие обозначения и рассмотрим пример.

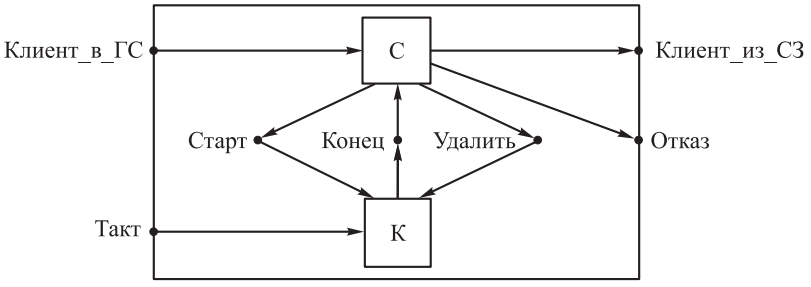
Архитектуру интерфейсной системы любого уровня иерархии представим четверкой $S = \{P, L, I, O\}$, где P — множество процессов системы, изображаемых прямоугольниками; L — множество каналов системы, соединяющих процессы; I — множество подмножеств $I(p) \subset L$ входных каналов процессов $p \in P$; O — множество подмножеств $O(p) \subset L$ выходных каналов процесса $p \in P$.

Каждый процесс $p \in P$ верхнего уровня интерфейсной системы может задаваться интерфейсной системой более низкого уровня $S(p) = \{P(p), L(p), I(p), O(p), ei(p), eo(p)\}$, где $P(p)$, $L(p)$, $I(p)$, $O(p)$ имеют тот же смысл, что и параметры P, L, I, O, но только как внутренняя структура процесса p; $ei(p)$ — функция, отображающая внутренние каналы процесса p в его внешние входные каналы; $eo(p)$ — функция, отображающая внутренние каналы процесса p в его внешние выходные каналы.

Рассмотрим пример, иллюстрирующий предлагаемые принципы верификации свойств интерфейса. Этот пример в некоторой степени использует интерфейс клиент-серверной системы, заимствованной из работы [10]. Система состоит из главного сервера, отвечающего за интерфейс с клиентами, сервера задержки, перераспределяющего клиентов в случае их длительного обслуживания главным сервером, генератора, фиксирующего клиентов и направляющего их главному серверу, и сервера статистики, фиксирующего обслуженных клиентов и серверы, их обслужившие. Клиенты, слишком долго ожидающие обслуживания главным сервером, могут отказаться от его услуги и обратиться к серверу задержек. Сервер задержек является конкурентом главного сервера и может оказывать те же услуги, что и главный сервер, но за более длительное время. Каждый клиент решает, следует ли ему отказываться от услуг главного сервера без их предоставления и переходить на обслу-



а



б

Рис. 6. Архитектуры клиент-серверной системы S (*а*) и главного сервера (*б*):

ГС — главный сервер; СЗ — сервер задержки; Г — генератор; СС — сервер статистики; С — буфер; К — клиентский сервис

живание к серверу задержек или нет. Пример архитектуры клиент-серверной системы *S* верхнего уровня представлен на рис. 6, *а*.

Согласно приведенным выше определениям, для клиент-серверной системы первого уровня, представленной на рис. 6, *б*, имеем

- $P = \{Г, ГС, СС, СЗ\}$;
- $L = \{Клиент\_в\_ГС, Клиент\_из\_ГС, Такт, Отказ, Клиент\_из\_СС\}$;
- $I = \{I(ГС), I(СС), I(СЗ)\}$,
  - $I(Г) = \varnothing$,
  - $I(ГС) = \{Клиент\_в\_ГС, Такт\}$,
  - $I(СС) = \{Клиент\_из\_ГС, Клиент\_из\_СС\}$,
  - $I(СЗ) = \{Такт, Отказ\}$;
- $O = \{ O(Г), O(ГС), O(СЗ)\}$,
  - $O(Г) = \{Клиент\_в\_ГС, Такт\}$,
  - $O(ГС) = \{Клиент\_из\_ГС, Отказ\}$,
  - $O(СС) = \varnothing$,
  - $O(СЗ) = \{Клиент\_из\_СЗ\}$.

Пример архитектуры второго уровня для главного сервера, представленного на рис. 6, *а*, показан на рис. 6, *б*.

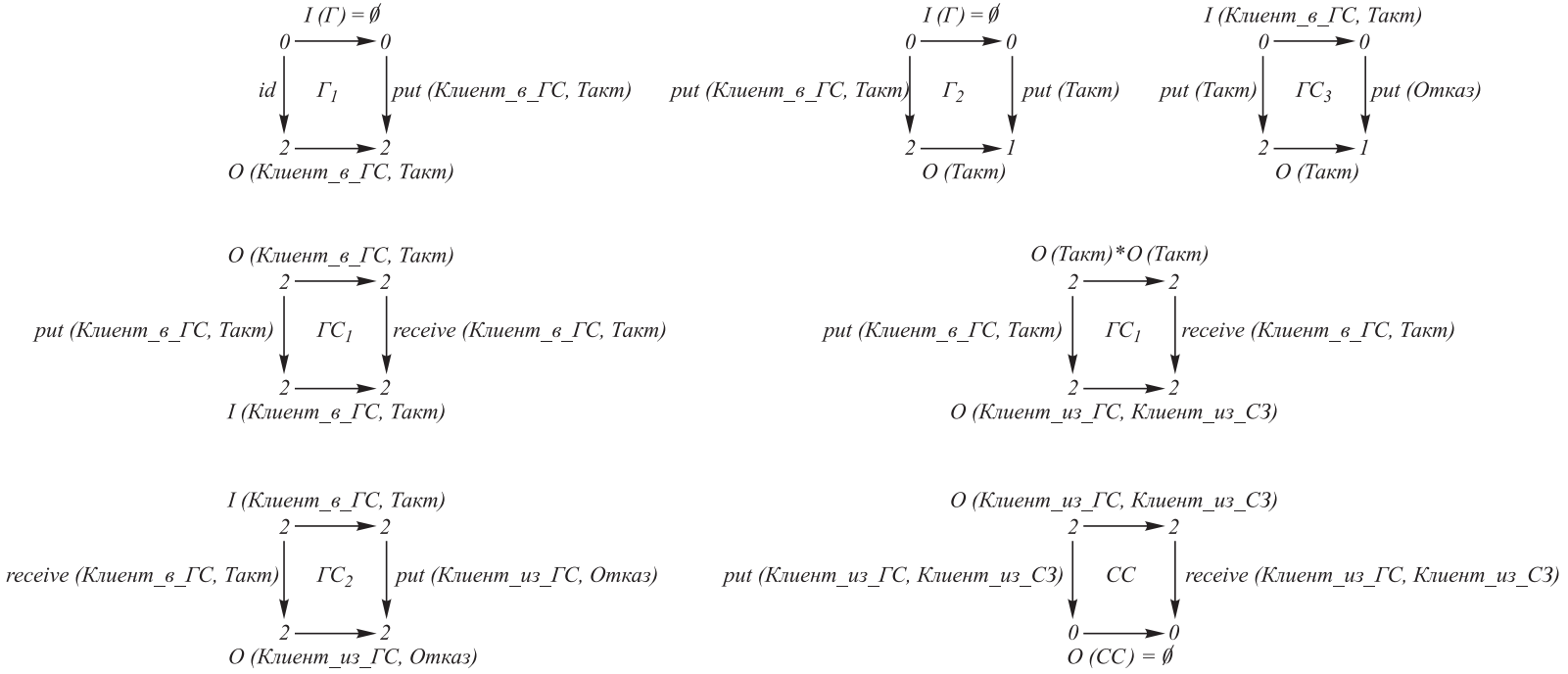
Для архитектуры главного сервера (см. рис. 6, *б*) будем иметь

- $P(ГС) = \{C, K\}$;
- $L(ГС) = \{Старт, Конец, Удалить\}$;
- $I(ГС) = \{I(C), I(K)\}$,
  - $I(C) = \{Клиент\_в\_ГС, Конец\}$
  - $I(K) = \{Такт, Старт, Удалить\}$;
- $O(ГС) = \{O(C), O(K)\}$,
  - $O(C) = \{Клиент\_из\_ГС, Отказ\}$
  - $O(K) = \{Конец\}$;
- $ei(ГС) = I(ГС) = \{Клиент\_в\_ГС, Такт\}$;
- $eo(ГС) = O(ГС) = \{Клиент\_в\_ГС, Отказ\}$.

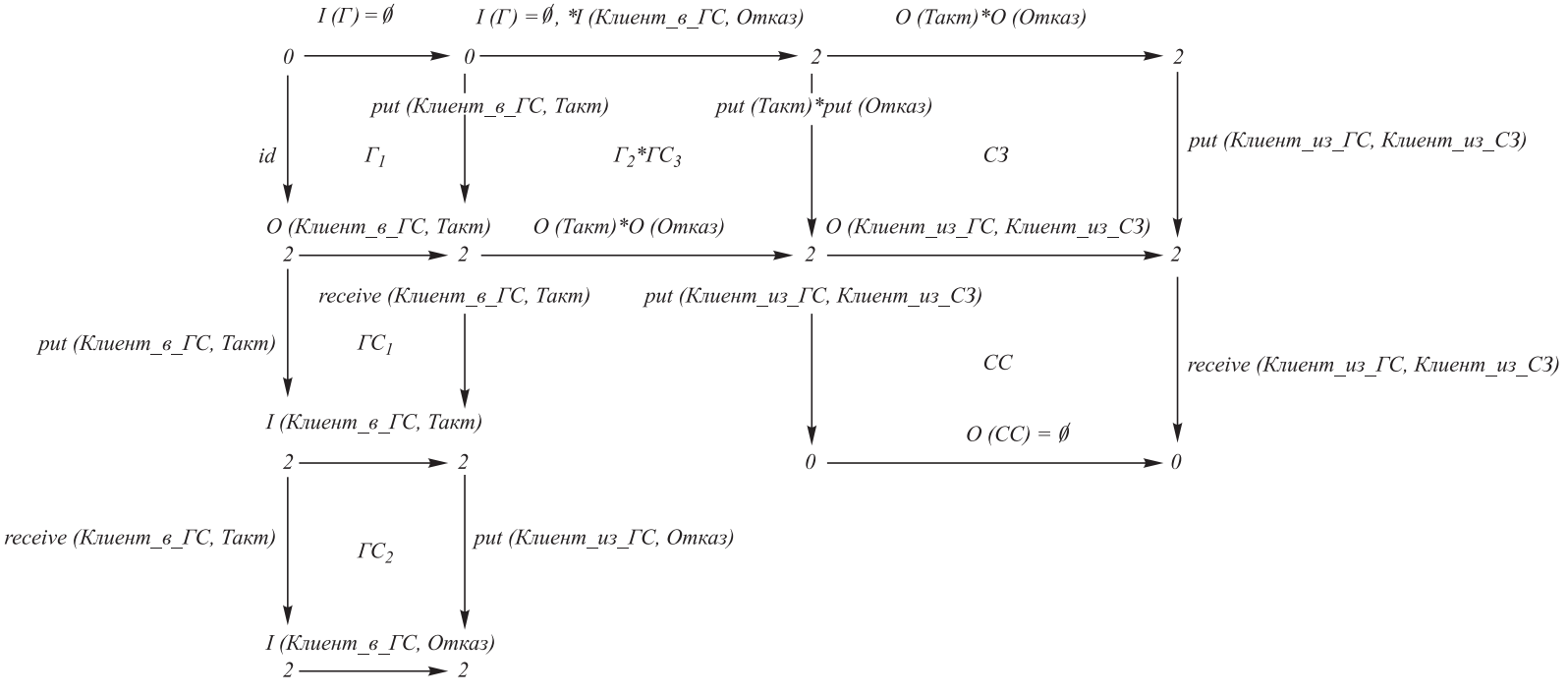
Для простоты тайлы будем именовать индексированными именами процессов, конфигурации тайлов обозначим подмножествами их каналов, а действия над ними по помещению и получению конкретных сообщений в каналы, — подмножествами имен каналов с указанием действий поместить (*put*) или получить (*receive*), совершаемых над ними.

Множество тайлов, задающих локальное поведение процессов первого уровня системы *S*, показано на рис. 7, *а*.

Композиция тайлов, задающая общее поведение процессов первого уровня системы *S*, приведена на рис. 7, *б*.


I (Г) = ∅
0 0
id Г1 put (Клиент_в_ГС, Такт)
2 2
O (Клиент_в_ГС, Такт)
I (Г) = ∅
0 0
put (Клиент_в_ГС, Такт) Г2 put (Такт)
2 1
O (Такт)
I (Клиент_в_ГС, Такт)
0 0
put (Такт) ГС3 put (Отказ)
2 1
O (Такт)
O (Клиент_в_ГС, Такт)
2 2
put (Клиент_в_ГС, Такт) ГС1 receive (Клиент_в_ГС, Такт)
2 2
I (Клиент_в_ГС, Такт)
O (Такт)*O (Такт)
2 2
put (Клиент_в_ГС, Такт) ГС1 receive (Клиент_в_ГС, Такт)
2 2
O (Клиент_из_ГС, Клиент_из_СЗ)
I (Клиент_в_ГС, Такт)
2 2
receive (Клиент_в_ГС, Такт) ГС2 put (Клиент_из_ГС, Отказ)
2 2
O (Клиент_из_ГС, Отказ)
O (Клиент_из_ГС, Клиент_из_СЗ)
2 2
put (Клиент_из_ГС, Клиент_из_СЗ) СС receive (Клиент_из_ГС, Клиент_из_СЗ)
0 0
O (СС) = ∅

а



б

Рис. 7. Множество тайлов поведения системы *S* (*а*) и композиция тайлов системы *S* (*б*)

ЛИТЕРАТУРА

1. *Девятков В.В., Алфимцев А.Н.* Необходимые и достаточные формальные свойства мультимодального интерфейса // Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение. Спец. вып. «Информационные технологии и компьютерные системы». 2011. С. 159–166.
2. *Robin M.* Communicating and Mobile Systems: The π-calculus. Cambridge: University Press, 2003. 159 p.
3. *Medvidovic N., Taylor R.M.* A Classification and comparison framework for software architecture description languages // IEEE Transactions on Software Engineering. 2000. Vol. 26. No 1. P. 70–93.
4. *Braga C., Sztajnberg A.* Towards a rewriting semantics for a software architecture description language // Proceedings of WMF 2003, 6th Workshop on Formal Methods, Campina Grande, Brazil, E.N.T.C.S. 95. 2003. P. 148–168.
5. *Bruni R., Fiadeiro J.L., Lanese I., Lopes A., Montanari U.* New insight into the algebraic properties of architectural connectors // International Federation for Information Processing. 2004. Vol. 155. P. 367–380.
6. *Choutri A., Belala F., Barkaoui K.* A Tile logic based approach for software architecture description analysis // J. Software Engineering & Applications. 2010. Vol. 3. P. 1067–1079.
7. *Bouanaka C., Choutri A., Belala F.* On Generating tile system for a software architecture: case of a collaborative application session // ICSOFT2007 (Second Conference on Software and Data Technologies), July 22–25. 2007. P. 123–128.
8. *Bruni R*. Tile logic for synchronized rewriting of concurrent systems. Phd Thesis. University of Pisa. TD-1/99. March. 1999.
9. *Адаменко А.Н., Кучков А*. Логическое программирование и Visual Prolog. СПб.: БХВ-Петербург, 2003.
10. *Bouanaka C., Belala F., Barkaoui K.* A Tile logic based semantics for mobile software architectures // International Journal of Critical Computer-Based Systems. 2011. Vol. 2(3). P. 288–308.

REFERENCES

[1] Devyatkov V.V., Alfimtsev A.N. Necessary and sufficient formal properties of multimodal interface. *Vestn. Mosk. Gos. Tekh. Univ. im. N.E. Baumana, Priborostr., Spetsvyp.* "*Informatsionnye tekhnologii i komp'yuternye sistemy*" [Herald of the Bauman Moscow State Tech. Univ., Instrum. Eng., Spec. Issue "Information technology and computer system"], 2011, pp. 159–166 (in Russ.).

[2] Robin M. Communicating and Mobile Systems: The π-calculus. Cambridge: University Press, 2003. 159 p.

[3] Medvidovic N., Taylor R.M. A Classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering,* 2000, vol. 26, no. 1, pp. 70–93.

[4] Braga C., Sztajnberg A. Towards a rewriting semantics for a software architecture description language. *Proceedings of WMF 2003, 6th Workshop on Formal Methods.* Campina Grande, Brazil, E.N.T.C.S. 95, 2003, PP. 148–168.

[5] Bruni R., Fiadeiro J.L., Lanese I., Lopes A., Montanari U. New insight into the algebraic properties of architectural connectors. *International Federation for Information Processing*, 2004, vol. 155, pp. 367–380.

[6] Choutri A., Belala F., Barkaoui K. A tile logic based approach for software architecture description analysis. *J. Software Engineering & Applications*, 2010, vol. 3, pp. 1067–1079.

[7] Bouanaka C., Choutri A., Belala F. On generating tile system for a software architecture: case of a collaborative application session. *ICSOFT2007* (*Second Conference on Software and Data Technologies*), 2007, July 22–25, pp. 123–128.
[8] Bruni R. Tile logic for synchronized rewriting of concurrent systems. *Phd Thesis. University of Pisa*. TD-1/99, March, 1999.
[9] Adamenko A.N., Kuchkov A. Logicheskoe programmirovanie i Visual Prolog [Logic Programming and Visual Prolog]. St. Petersburg, BKhV-Petersburg Publ., 2003.
[10] Bouanaka C., Belala F., Barkaoui K. A tile logic based semantics for mobile software architectures. *International Journal of Critical Computer-Based Systems*, 2011, vol. 2(3), no. 1, pp. 288–308.

Статья поступила в редакцию 18.01.2016

Девятков Владимир Валентинович — д-р техн. наук, профессор, заведующий кафедрой «Информационные системы и телекоммуникации» МГТУ им. Н.Э. Баумана (Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5).

Devyatkov V.V. — Dr. Sci. (Eng.), Professor, Head of Information Systems and Telecommunication Department, Bauman Moscow State Technical University (2-ya Baumanskaya ul. 5, Moscow, 105005 Russian Federation).

Просьба ссылаться на эту статью следующим образом:

Девятков В.В. Верификация свойств интеллектуальных интерфейсов в логике тайлов // Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение. 2016. № 3. С. 65–87. DOI: 10.18698/0236-3933-2016-3-65-87

Please cite this article in English as:

Devyatkov V.V. Verification of Intelligent Interface Properties in the Tiles Logic. *Vestn. Mosk. Gos. Tekh. Univ. im. N.E. Baumana, Priborostr.* [Herald of the Bauman Moscow State Tech. Univ., Instrum. Eng.], 2016, no. 3, pp. 65–87. DOI: 10.18698/0236-3933-2016-3-65-87