РАЗРАБОТКА СРЕДСТВ АГЕНТНОГО ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ ДЛЯ МНОГОКАНАЛЬНОГО ИНТЕЛЛЕКТУАЛЬНОГО ВИДЕОНАБЛЮДЕНИЯ

[1]Морозов А.А., [1]Сушкова О.С., [2]Хохлова М.Н., [2]Миньо К., [1]Петрова Н.Г.

[1]Институт радиотехники и электроники им. В.А. Котельникова РАН, http://www.fullvision.ru
Москва 125009, Российская Федерация,
[2]Университет Бургундии Франш-Конте, http://www.ubfc.fr/
Дижон 21000, Франция

Поступила 03.06.2018
Представлена членом-корреспондентом РАН и РАЕН Петровым И.Б.

Рассмотрены экспериментальные средства параллельного объектно-ориентированного логического языка программирования Акторный Пролог, разработанные для создания систем многоканального интеллектуального видеонаблюдения. Средства логического языка рассмотрены на примере задачи непрерывного мониторинга одного из физиологических параметров - температуры тела людей, находящихся в зоне видеонаблюдения. Для реализации непрерывного мониторинга температуры тела человека логическая программа осуществляет слияние информации, полученной с помощью двух устройств: (1) времяпролётной видеокамеры, фиксирующей трёхмерные координаты поверхности тела, на основе которых вычисляются трёхмерные координаты скелетона человека - графического изображения его фигуры с акцентом на его опорно-двигательный аппарат, (2) тепловизионной камеры, создающей тепловизионное видеоизображение поверхности тела. Для получения и анализа перечисленных видов трёхмерной и двумерной видеоинформации в Акторном Прологе разработаны встроенные классы, реализующие взаимодействие с времяпролётной камерой устройства Кинект 2, хранение и передачу трёхмерных изображений, трёхмерную графику и др. Рассмотрен пример, в котором тепловизионное видеоизображение поверхности человеческого тела проецируется на узлы и дуги графа, описывающего скелетон человека. Перечисленные операции осуществляются в режиме реального времени специальным логическим агентом, то есть отдельной логической программой, написанной на Акторном Прологе, и передаются другому логическому агенту. Второй логический агент осуществляет усреднение значений температуры скелетона человека во времени и отображает скелетон в виде цветного трёхмерного изображения; при этом цветом обозначается средняя температура узлов и дуг скелетона. Для взаимодействия логических агентов используется механизм удалённых вызовов предикатов, разработанный и реализованный в Акторном Прологе для поддержки агентного логического программирования. Целью разработки перечисленных средств логического программирования является реализация семантического анализа в системах интеллектуального видеонаблюдения.

Ключевые слова: **интеллектуальное видеонаблюдение; тепловидение, устройство Кинект 2, трёхмерное зрение; объектно-ориентированное логическое программирование; язык Акторный Пролог; скелетоны; распознавание сложных событий; машинное зрение; техническое зрение**

УДК 510.663; 519.68:007.5; 519.68:681.513.7; 681.3.06

Содержание

1. ВВЕДЕНИЕ

В настоящей статье рассмотрен проект разработки программной платформы для распределённого логического анализа разнородной многоканальной информации в системах интеллектуального видеонаблюдения на основе объектно-ориентированного логического языка программирования Акторный Пролог. Целью разработки этой программной платформы является реализация семантического анализа в системах интеллектуального видеонаблюдения. Семантический (смысловой) анализ видеосцен означает, что система интеллектуального видеонаблюдения не только распознаёт заданные предметы, действия людей и события, но и способна оценить их смысл в текущем контексте. Примером семантического анализа видеосцены является анализ поведения наблюдаемых людей, и в частности, анализ социального взаимодействия людей. Задача анализа поведения человека, в отличие от задачи распознавания действий человека, предполагает учёт контекста наблюдаемых действий, а именно, дополнительных обстоятельств, связанных с местом, временем, предшествующими и последующими событиями, действиями других людей, наличием или отсутствием каких-либо предметов и др.

Задача смыслового анализа видеосцен имеет очевидные области применения в сферах медицины, образования и искусства, однако наиболее актуальной областью применения семантического анализа видеосцен, по мнению авторов, является распознавание подозрительного поведения людей в интеллектуальных системах безопасности. Например, анализ социального взаимодействия и распознавание социальных связей между людьми (рукопожатие, драка и т.п.) в видеопотоке необходим для воссоздания полной картины социального взаимодействия людей на определённой территории (например, в здании) и выявления посторонних лиц или злоумышленников. Распознавание социального статуса и роли людей по их внешнему виду (например, распознавание должностных лиц, полицейских, военных, обслуживающего персонала) необходимо для проверки соответствия внешнего вида людей и выполняемых ими действий (например, для распознавания преступника или террориста в одежде полицейского).

В качестве математического аппарата для описания и анализа контекста наблюдаемых событий традиционно используются математическая логика и логическое программирование. Задача применения математической логики для анализа видеосцен тесно связана с такими областями математики и информатики как пространственные и временные логики, автоматическое доказательство теорем в реальном времени, распределённый и параллельный логический вывод, объектно-ориентированное логическое программирование и др. При этом возможность практического применения логического анализа видеосцен определяется, прежде всего, наличием исходной информации, достаточной для вывода практически полезных логических следствий. Например, появление современных средств ввода трёхмерной видеоинформации (таких как времяпролётные камеры, стереокамеры, импульсные лидары и др.) обеспечило возможность анализа поведения людей, включающего сложное социальное взаимодействие и манипулирование предметами. Дальнейший прогресс в области логического анализа видеосцен, очевидно, будет связан с развитием нейросетевых средств распознавания поз людей и объектов. Авторы рассматривают математическую логику как математический язык, наиболее подходящий для декларативного описания сложных форм поведения и событий на основе комбинирования описаний простых событий и действий, а логическое программирование как наиболее удобный, гибкий и практически полезный инструмент для реализации логического вывода в реальном времени.

Заметим, что логическое программирование является лишь одним из возможных подходов к использованию математической логики для описания и анализа смысла видеоизображений.

Близкими направлениями являются системы интеллектуального видеонаблюдения, основанные на правилах, нечётком логическом выводе, марковских логических сетях и др. Отличительной особенностью подхода к разработке систем интеллектуального видеонаблюдения, основанного на логическом программировании, является то, что логические программы являются не просто набором логических правил или формул, но одновременно инструкцией по обработке данных. Другими словами, логическая программа обладает не только декларативной, но и операционной семантикой. Это означает, что программист, составляющий логическое описание искомого поведения или события, имеет возможность управлять ходом логического вывода, в том числе, учитывать скорость выполнения отдельных участков программы и объём обрабатываемой информации. Таким образом, применение логического языка (а не просто набора правил) открывает возможность для эффективной обработки мультимедийных (многоканальных) данных и практического применения логического вывода в системах интеллектуального видеонаблюдения.

Данная статья посвящена развитию средств объектно-ориентированного и агентного логического программирования в контексте задачи семантического анализа видеосцен. Опыт применения логического программирования для анализа видеоинформации показал, что подобные исследования не только дают практический выход в области анализа видеоинформации, но при этом порождают новые идеи и методы в области логического программирования. Дело в том, что логическое программирование (и наиболее известный логический язык Пролог) традиционно развивались как средства обработки символьной информации в задачах искусственного интеллекта. Обработка данных в реальном времени, а также обработка больших массивов данных (изображений, видеоданных, мультимедийных данных и т.п.) традиционно считались неудачными областями применения логических языков (за исключением нескольких проектов), и это наложило отпечаток на синтаксис логических языков, реализацию компиляторов, библиотек встроенных процедур и пр. Опыт применения логического программирования в области интеллектуального видеонаблюдения позволил выявить «узкие места» в существующих логических языках и компиляторах, а также стимулировал развитие новых направлений в этой области, прежде всего, объектно-ориентированного логического программирования, агентного логического программирования и трансляции языка Пролог в императивные языки программирования.

Объектно-ориентированное логическое программирование – это область исследований, направленная на разработку языков программирования, сочетающих выразительные возможности логического и объектно-ориентированного программирования. Объектно-ориентированное логическое программирование, как самостоятельное направление, развивается уже несколько десятилетий, но при этом продолжает оставаться на периферии исследований в области логического программирования. Достаточно сказать, что до сих пор не существует общепринятого понимания того, что является «объектом», «классом» и «наследованием» в логическом языке; разные исследователи трактуют эти понятия по-разному. Такое положение дел связано, по-видимому, с тем, что в области логического программирования работают как математики, так и специалисты по информатике; при этом математики традиционно считают объектно-ориентированное программирование направлением информатики, несущественным с точки зрения математической логики, а информатики не всегда уделяют должное внимание исследованиям математической (теоретико-модельной) семантики логических языков. В настоящее время опыт применения логического программирования

для анализа мультимедийных данных позволил по-новому расставить акценты и актуализировать исследования в области объектно-ориентированного логического программирования.

Практическая необходимость разработки и применения средств объектно-ориентированного логического программирования связана с необходимостью эффективной обработки больших массивов разнородных данных. В языке Пролог для описания сложных структур данных (в том числе, векторов, матриц и других видов массивов) традиционно использовались структуры и списки, то есть средства символьной обработки информации. Это связано с тем, что структуры и другие элементы данных языка Пролог, в отличие от массивов, имеют взаимно-однозначное соответствие в логике предикатов первого порядка (сколемовские функции, числа, переменные и др.). Это не является проблемой с точки зрения математических свойств и описательных возможностей логического языка, однако делает его практически непригодным для работы с мультимедийной информацией. Выразительные средства объектно-ориентированного логического программирования решают эту проблему, потому что позволяют, не нарушая математическую семантику языка, инкапсулировать большие массивы данных в экземплярах специальных встроенных классов, разработанных для анализа мультимедийной информации.

В данной работе описаны принципы анализа многоканальной (мультимедийной) информации в объектно-ориентированном логическом языке Акторный Пролог на примере работы с трёхмерными видеоданными, получаемыми с помощью устройства Кинект 2 (Microsoft, Inc.), и тепловизионными данными, получаемыми с тепловизионной камеры Thermal Expert V1 (i3system, Inc.). Обсуждается применение средств распределённого логического программирования языка Акторный Пролог для программирования агентов, реализующих анализ трёхмерной видеоинформации и её слияние с тепловизионным видеоизображением.

Слияние тепловизионного видео с трёхмерными видеоданными и данными других видов является перспективной, интенсивно развивающейся областью исследований [1-4]. Основными областями применения этой идеи являются обнаружение, распознавание и реидентификация людей в системах интеллектуального видеонаблюдения [5-11], медицинские приложения, в том числе, дистанционное выявление опасных заболеваний системой безопасности аэропорта [12-15], поддержка работы спасателей [16], взаимодействие людей с роботами [17], сельское хозяйство [18, 19], автоматическое видеонаблюдение за исправностью и безопасностью функционирования оборудования, тепловизионная одометрия [20-22]. В данной статье в качестве примера будет рассмотрена задача дистанционной оценки температуры частей тела людей, перемещающихся в зоне видеонаблюдения.

Во втором разделе статьи рассмотрены архитектура и общие принципы, заложенные в основу системы логического программирования Акторный Пролог. В третьем разделе рассмотрены встроенные классы языка, разработанные авторами для сбора и анализа трёхмерных видеоданных. В четвертом разделе рассмотрен пример логического агента, осуществляющего слияние трёхмерной видеоинформации о форме тела наблюдаемых людей и тепловизионной видеоинформации. В пятом разделе рассмотрены принципы и механизмы взаимодействия логических агентов в языке Акторный Пролог.

2. АРХИТЕКТУРА СИСТЕМЫ ПРОГРАММИРОВАНИЯ АКТОРНЫЙ ПРОЛОГ

Акторный Пролог – это логический язык программирования, разработанный в Институте радиотехники и электроники им. В.А. Котельникова Российской академии наук [23-27]. Акторный Пролог

изначально проектировался как объектно-ориентированный и логический язык одновременно, то есть, в язык введены понятия «класс», «экземпляр класса», «наследование» и др., и при этом объектно-ориентированные логические программы имеют теоретико-модельную семантику. Одновременно с классами и наследованием в Акторном Прологе реализованы синтаксические средства для описания типов данных (доменов), детерминированности и направления передачи аргументов в подпрограммах [28, 29]. Эти средства являются критически важными для промышленного логического программирования, потому что, как показывает опыт, без них практически невозможно отлаживать и поддерживать большие сложные программы. Акторный Пролог является параллельным языком; на уровне синтаксиса языка поддерживается создание и организация взаимодействия параллельных процессов. Эти синтаксические средства языка также обеспечивают наличие теоретико-модельной семантики логических программ, но лишь при наложении определённых ограничений на их структуру [25, 26].

Отличительной особенностью системы логического программирования Акторный Пролог является то, что логические программы транслируются в язык Джава и затем исполняются средствами стандартной виртуальной Джава-машины [30, 31]. Такая схема исполнения логических программ была разработана, прежде всего, для того чтобы в максимальной степени обеспечить устойчивость работы логических программ, включая систему управления памятью. Другим важным достоинством этого проектного решения является лёгкость расширения языка и добавления в него специализированных встроенных классов.

Расширение языка осуществляется следующим образом. В ходе трансляции логической программы она преобразуется в набор классов Джавы. В языке предусмотрены синтаксические средства, которые позволяют объявлять некоторые автоматически создаваемые классы Джавы потомками других, заранее подготовленных Джава-классов. Таким образом, для добавления в язык нового встроенного класса достаточно реализовать этот класс на языке Джава и подсоединить его к логической программе в ходе трансляции. В настоящее время часть встроенных классов Акторного Пролога реализованы полностью на Джаве, другая часть представляет собой реализованный на Джаве интерфейс с открытыми библиотеками, написанными на Си++. Примерами классов, реализованных полностью на Джаве, являются класс *Database,* представляющий собой простую систему управления базами данных, класс *File*, поддерживающий чтение и запись файлов, класс *WebResource*, поддерживающий получение данных из Интернета, и др. Примерами встроенных классов-интерфейсов с открытыми библиотеками являются класс *FFmpeg,* поддерживающий чтение и запись видеоданных, класс *Java3D*, реализующий трёхмерную графику, *Webcam*, предназначенный для ввода видеоданных, и др. Разработчики рассматривают трансляцию в Джаву как архитектурное решение, позволяющее максимально быстро добавлять и отлаживать новые встроенные классы. Ускорение жизненного цикла разработки встроенных классов происходит вследствие того, что язык Джава, по сравнению с языком Си++, предотвращает появление во встроенных классах наиболее трудных для выявления ошибок, связанных с выходом за границы массивов и некорректным управлением памятью.

3. ВСТРОЕННЫЕ КЛАССЫ ДЛЯ ИНТЕЛЛЕКТУАЛЬНОГО ВИДЕОНАБЛЮДЕНИЯ

В настоящее время в системе программирования Акторный Пролог реализован набор встроенных классов, предназначенных для ввода и анализа двух- и трёхмерной видеоинформации. Разработка этих классов осуществлялась, в основном, для того, чтобы удовлетворить практические

потребности нашей исследовательской группы, возникающие в ходе экспериментов с интеллектуальным видеонаблюдением.

Изначально в Акторном Прологе был реализован класс *ImageSubtractor,* реализующий стандартный набор операций низкоуровневой обработки видео, таких как вычитание фона, гауссова и ранговая фильтрация изображений, выделение блобов, оценка размеров и скорости перемещения блобов, других статистических характеристик, описывающих плавность движения блобов, и др. При этом все матрицы данных, возникающие на различных этапах обработки видео, остаются инкапсулированными в экземпляре класса *ImageSubtractor*, что обеспечивает высокую скорость обработки видеоданных. На уровне логических правил Акторного Пролога обрабатываются лишь результаты низкоуровневого анализа видео, а именно, графы, описывающие траектории движения, скорость и другие свойства блобов, выделенных в видеоизображении. Подробное описание встроенного класса *ImageSubtractor* и принципов его использования для низкоуровневого и высокоуровневого анализа видео можно найти в работах [30, 32-37].

Следующим этапом развития средств анализа видео в Акторном Прологе стала разработка виртуальной машины низкоуровневой обработки видеоизображений [38, 39], которая была реализована в классе *VideoProcessingMachine*. Разработка этой виртуальной машины осуществлялась, в основном, для анализа биомедицинских видеоизображений (анализа поведения лабораторных животных), которые характеризуются наличием в видеоизображении одновременно блобов нескольких типов, таких, что для выделения разных типов блобов требуются разные низкоуровневые операции над изображением [38, 40]. Виртуальная машина низкоуровневой обработки видеоизображений позволяет программисту подготовить последовательность команд (обработка изображений в пиксельном представлении, выделение и обработка пикселей переднего плана, выделение и трассировка блобов и пр.), которая автоматически повторяется для каждого нового кадра видеоизображения. Все матрицы данных, необходимые для обработки видео, так же как и в классе *ImageSubtractor*, остаются инкапсулированными в экземпляре класса *VideoProcessingMachine*.

В данной статье описаны средства Акторного Пролога, разработанные для ввода и анализа трёхмерных видеоданных с устройства Кинект 2. Эти средства и соответствующие встроенные классы Акторного Пролога были созданы для экспериментов с трёхмерным интеллектуальным видеонаблюдением [41, 42]. Разработанные средства основаны на тех же принципах, что и средства анализа двумерной видеоинформации:

1. Разделение низкоуровневой и высокоуровневой обработки видеопотока.
2. Реализация низкоуровневой обработки в специальных встроенных классах, позволяющих инкапсулировать массивы видеоданных.
3. Реализация на уровне правил логического языка высокоуровневой обработки (логического анализа) видеоданных в форме графов, списков и других термов логического языка.

При этом, однако, схема обработки видеоданных была изменена для того, чтобы учесть особенности трёхмерных видеоданных:

1. *Многомодальность видеоданных.* Например, устройство Кинект 2, используемое для ввода трёхмерных данных, поддерживает одновременно несколько потоков данных: кадры, описывающие трёхмерное облако точек, кадры инфракрасного видеоизображения, кадры цветного видеоизображения, кадры, описывающие координаты скелетонов наблюдаемых людей и др.
2. *Очень большой объём видеоданных.* Как правило, производительности современного персонального компьютера не хватает, чтобы в реальном времени сохранять на жёсткий диск полностью все данные, поступающие из устройства Кинект 2,

и предпочтительной схемой обработки является анализ трёхмерных видеоданных в реальном времени и сохранение (передача по сети) промежуточных результатов этой обработки.

В Акторном Прологе разработаны два встроенных класса, реализующие разные аспекты, связанные с вводом и анализом трёхмерной видеоинформации: *Kinect* и *KinectBuffer*. Первый класс предназначен непосредственно для взаимодействия с устройством Кинект 2, второй служит для низкоуровневого анализа, чтения и записи трёхмерных видеоданных. Описания обоих классов вместе с определениями соответствующих доменов (типов данных) помещены в пакет Акторного Пролога "Morozov/Kinect".

Класс *KinectBuffer* является наиболее важным в схеме анализа трёхмерных данных. Предполагается, что экземпляр класса *KinectBuffer* может находиться в одном из четырёх состояний: получение данных из устройства Кинект 2, чтение файла трёхмерных данных, проигрывание файла, запись файла. Режимы чтения файла и проигрывания файла отличаются тем, что в режиме проигрывания файла класс *KinectBuffer* считывает и автоматически передаёт в логическую программу данные в темпе реального времени; в режиме чтения файла считывание каждого отдельного кадра видеоизображения происходит под контролем программиста.

Чтобы выбрать требуемое состояние экземпляра класса, необходимо задать значение атрибута *operating_mode* класса *KinectBuffer*: 'LISTENING', 'READING', 'PLAYING' или 'RECORDING' соответственно. Для чтения и записи файлов достаточно класса *KinectBuffer,* однако если этот класс используется в режиме получения данных из устройства Кинект 2, необходимо создать экземпляр класса *Kinect* и передать его в экземпляр класса *KinectBuffer* в качестве значения атрибута *input_device*. В следующем разделе будет рассмотрен пример логической программы, осуществляющей чтение и анализ трёхмерных видеоданных.

4. ВВОД И СЛИЯНИЕ ТРЁХМЕРНОЙ И ТЕПЛОВИЗИОННОЙ ВИДЕОИНФОРМАЦИИ

Рассмотрим пример логической программы, осуществляющей ввод и анализ трёхмерных видеоданных, полученных с помощью времяпролётной камеры устройства Кинект 2. Далее будут приведены фрагменты исходного кода на Акторном Прологе с комментариями. Исходный код будет содержать пропуски; целью данного примера является лишь описание схемы обработки данных, реализованной во встроенных классах *Kinect* и *KinectBuffer*.

Определим класс *3DVideoSupplier,* являющийся потомком класса *KinectBuffer*. Значение атрибута *operating_mode* указывает, что данные должны вводиться из файла "My3DVideo" в режиме проигрывания (PLAYING).

```
class '3DVideoSupplier' (specialized
'KinectBuffer'):
name           = "My3DVideo";
operating_mode = 'PLAYING';
```

В ходе создания экземпляра класса *3DVideoSupplier* предикат *goal* автоматически загрузит из файла "MyLookUpTable.txt" таблицу соответствий трёхмерных координат, измеряемых времяпролётной камерой, и двумерных координат на тепловизионном видеоизображении. После этого с помощью предиката *start* будет активировано чтение данных из файла.

```
goal:-!,
     set_lookup_table("MyLookUpTable.
txt"),
     start.
```

Класс *KinectBuffer* поддерживает двумерные и трёхмерные таблицы соответствий. Предполагается, что таблица соответствий вычислена заранее на основе данных, полученных при калибровке используемой видеосистемы, и записана в текстовый файл. Двумерная таблица представляет собой матрицу, размеры которой соответствуют размерам изображения *K*, выдаваемого устройством Кинект 2. В каждой ячейке (*i, j*)

этой матрицы записаны координаты x и y на изображении T (в рассматриваемом примере, тепловизионном), которое должно быть отображено на трёхмерные поверхности, исследуемые с помощью времяпролётной камеры. Трёхмерная таблица соответствий также представляет собой матрицу, однако в ячейках матрицы записаны не сами координаты, а коэффициенты квадратичных полиномов для вычисления координат на изображении T. В каждой ячейке матрицы (i, j) записаны шесть коэффициентов $p1, p2, p3, q1, q2, q3$. Координаты на изображении T вычисляются с помощью квадратичных полиномов, зависящих от обратной величины расстояния $d(i,j)$ в метрах от времяпролётной камеры до точки (i, j) на исследуемой поверхности:

$$x = p_1 (1/d)^2 + p_2 (1/d) + p_3,$$
$$y = q_1 (1/d)^2 + q_2 (1/d) + q_3.$$

В рассматриваемом примере отображение тепловизионного изображения на трёхмерную поверхность осуществляется при считывании из файла каждого нового кадра трёхмерного видеоизображения. При считывании очередного кадра в экземпляре класса *KinectBuffer* автоматически вызывается предикат *frame_obtained*. С помощью вызова встроенного предиката *commit* программист сообщает классу *KinectBuffer* о том, что он собирается обработать текущий кадр видеоизображения. После этого все встроенные предикаты класса будут обращаться к содержимому именно этого кадра, пока предикат *commit* не будет вызван снова. В данном примере программа с помощью встроенного предиката *get_recent_frame_time* извлекает время в миллисекундах *Time1* полученного трёхмерного кадра, и на основе этой информации вычисляет номер соответствующего кадра тепловизионного видеоизображения. Атрибут *texture_time_shift* содержит величину временного сдвига между трёхмерной и тепловизионной видеозаписями. Атрибут *texture_frame_rate* содержит частоту кадров тепловизионного видео.

```
frame_obtained:-
     commit,!,
     get_recent_frame_time(Time1),
     Time2==Time1-texture_time_shift,
     FileNumber==  texture_frame_rate
* Time2 / 1000,
```

Для упрощения примера предполагается, что тепловизионное видео заранее разбито на отдельные кадры. По номеру кадра предикат *frame_obtained* вычисляет имя файла в формате JPEG, содержащего соответствующий кадр. Далее, с помощью предиката *load* кадр загружается в экземпляр *image* встроенного класса *BufferedImage*. Вызывается встроенный предикат *get_recent_scene* класса *KinectBuffer*. Этот предикат на основе данных времяпролётной камеры создаёт трёхмерную поверхность и отображает на неё заданную текстуру. Текстура передаётся в предикат с помощью второго аргумента (атрибут *image*, содержащий экземпляр класса *BufferedImage*). Отображение текстуры на создаваемую поверхность осуществляется с помощью загруженной ранее трёхмерной таблицы соответствий. Созданная трёхмерная поверхность возвращается через первый аргумент предиката, который должен содержать экземпляр встроенного класса *BufferedScene*, разработанного для хранения и передачи трёхмерных данных. Содержимое экземпляра класса *BufferedScene* может быть, в частности, включено в состав трёхмерной сцены, отображаемой средствами библиотеки *Java3D*. В рассматриваемом примере для этого используется встроенный предикат *set_node*, заменяющий указанный узел "MyLabel" в трёхмерном изображении, созданном с помощью экземпляра класса *Java3D*, содержащегося в слоте *graphic_window*.

```
ImageToBeLoaded == text?format(
  "%08d.jpeg",?round(FileNumber)),
image ? load(ImageToBeLoaded),
get_recent_scene(buffer3D,image),
graphic_window ? set_node(
     "MyLabel",
     'BranchGroup'({
          label: "MyLabel",
          allowDetach: 'yes',
```

```
            compile: 'yes',
            branches: [buffer3D]
        })),
```

Предикат *get_recent_mapping* класса *KinectBuffer* работает приблизительно так же, как предикат *get_recent_scene,* однако при этом трёхмерная поверхность не создаётся, и результаты отображения текстуры на трёхмерное изображение возвращаются в виде двумерного изображения. В данном примере предикат записывает созданное изображение в экземпляр *buffer2D* класса *BufferedImage*. Предикат *get_skeletons* класса *KinectBuffer* возвращает набор скелетонов людей, обнаруженных в текущем видеокадре. Скелетоны представляют собой графы, содержащие информацию о координатах туловища, головы, рук и ног людей. В программе эти графы представлены с помощью простых и составных термов логического языка – списков, структур, недоопределённых множеств, символов и чисел [41, 42]. В рассматриваемом примере скелетоны вместе с тепловизионным изображением передаются для дальнейшего анализа другому логическому агенту. Механизм передачи данных между логическими агентами будет рассмотрен в следующем разделе, здесь мы заметим лишь, что для передачи изображения, хранящегося в экземпляре *buffer2D* класса *BufferedImage*, оно преобразуется в терм типа BINARY. Для этого используется встроенная функция *get_binary* класса *BufferedImage*.

```
get_recent_mapping(buffer2D,image),
get_skeletons(Skeletons),
communicator?notify_all_consumers(
        Skeletons,buffer2D?get_
binary()).
```

Результаты слияния трёхмерной и тепловизионной видеоинформации, осуществляемые логическим агентом, представлены на **рис. 1.**

Далее будет рассмотрена схема взаимодействия между отдельными логическими программами (логическими агентами) с помощью встроенных классов

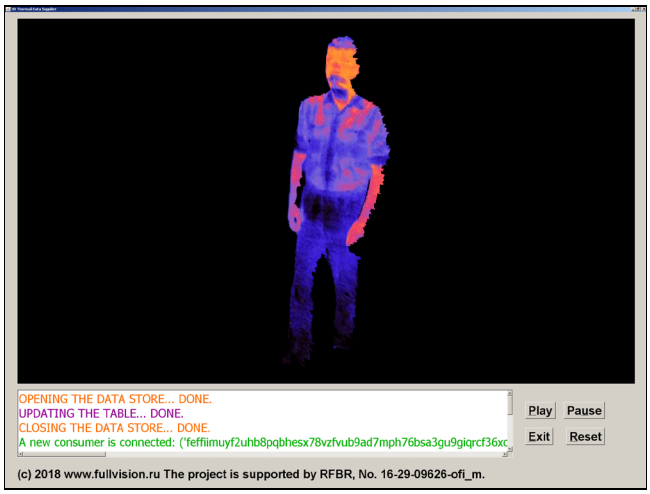

Рис. 1. *Логический агент, собирающий и передающий многоканальную видеоинформацию (времяпролётная камера + тепловизор).*

Database и *DataStore*, а также механизм удалённых вызовов предикатов.

5. УСТАНОВЛЕНИЕ СВЯЗИ И ВЗАИМОДЕЙСТВИЕ МЕЖДУ ЛОГИЧЕСКИМИ АГЕНТАМИ

Отличительной особенностью логического языка Акторный Пролог является механизм удалённых вызовов предикатов, разработанный для поддержки распределённого и децентрализованного логического программирования. Основная идея этого механизма заключается в том, что любой объект (в языке Акторный Пролог синонимами термина «объект» являются термины «экземпляр класса» и «мир») логической программы (далее, логического агента) может быть передан другому логическому агенту; после этого агент, получивший этот экземпляр класса, может осуществить асинхронный вызов предикатов переданного объекта [28, 29]. Сложность разработки этого механизма взаимодействия логических агентов заключалась в том, что Акторный Пролог является языком со строгой типизацией, и, следовательно, механизм проверки типов языка не предусматривал возможность динамического соединения и обмена информацией между логическими агентами, не располагающими на этапе трансляции абсолютно никакой информацией друг о друге. Решением

этой проблемы стала разработка в распределённой версии Акторного Пролога комбинированной системы типов, идея которой заключается в том, что принцип статической проверки типов в языке был смягчён, а именно, проверка корректности использования объекта, полученного из другой программы (агента), откладывается до тех пор, пока он не понадобится для удалённого вызова предиката. Соответствие типов аргументов предиката, вызываемого во внешнем объекте, осуществляется по структуре, а не по именам типов данных. Во всех остальных случаях в языке осуществляется стандартная статическая проверка типов. Комбинированная система типов позволила совместить в Акторном Прологе преимущества языка со строгой типизацией, а именно, возможность получения быстрого и надёжного исполняемого кода, с возможностями динамической проверки типов при взаимодействии агентов.

Для установления связи между логическими агентами экземпляр класса одного агента должен быть каким-либо образом передан другим агентам. Передача объекта может быть осуществлена разными способами, в том числе, через файл, через разделяемую базу данных или даже просто в текстовой форме по электронной почте. В рассматриваемом примере для этого будет использована встроенная система управления базами данных языка Акторный Пролог, реализованная с помощью встроенных классов *Database* и *DataStore*.

Определим класс *Main*, который в рассматриваемом примере будет являться одновременно основным классом логического агента (это означает, что исполнение логической программы начинается с создания экземпляра этого класса) и классом, экземпляр которого будет передан другому логическому агенту для установления связи и взаимодействия агентов.

В состав класса *Main* входят несколько слотов (переменных экземпляра класса). В частности, слот *datastore* содержит экземпляр встроенного класса *DataStore*. Значением слота *database* является экземпляр класса *3DDataSources*, потомка встроенного класса *Database*. Значением слота *video_supplier* является экземпляр класса *3DVideoSupplier*, рассмотренного в предыдущем разделе. Слот *consumers* содержит экземпляр класса *ConsumersList*, потомка класса *Database*, и служит для хранения списка агентов, запрашивающих информацию от рассматриваемого агента.

```
class 'Main' (specialized 'Alpha'):
     datastore = ('DataStore',
          name="AgentBlackboard.db",
          sharing_mode='shared_
access',
          access_mode='modifying');
     database  = ('3DDataSources',
          place= shared(
               datastore,
               "3DDataSources"));
     video_supplier=('3DVideoSupplier',
          communicator=self);
     consumers = ('ConsumersList');
```

В классе *Database* реализована встроенная система управления базами данных Акторного Пролога, поддерживающая традиционные для логических языков операции накопления и поиска элементов данных произвольной структуры. Можно сказать, что традиционные реляционные базы данных являются частным случаем базы данных Пролога, когда база данных используется только для хранения данных типа структура (состоящих из имени и набора аргументов). Заметим, что класс *Database* предназначен для хранения данных в оперативной памяти компьютера, то есть, для управления врЕменными данными, хотя операции записи и чтения содержимого базы данных в файл также поддерживаются.

Для управления структурами данных, разделяемых между несколькими программами, необходимо задействовать механизм управления данными более высокого уровня, реализованный во встроенном классе *DataStore*. Класс *DataStore* позволяет управлять работой одного или

нескольких экземпляров класса *Database*, например, записывать в файл и считывать из файла содержимое сразу нескольких экземпляров класса *Database*. Другой функцией класса *DataStore,* которой мы воспользуемся в рассматриваемом примере, является поддержка доступа к разделяемым данным, то есть связь экземпляров класса *Database* с файлами операционной системы и автоматическая передача изменений данных, осуществлённых в одной логической программе, в память других логических программ. Целостность разделяемых данных при этом обеспечивается с помощью общепринятого механизма транзакций.

В рассматриваемом примере в конструктор экземпляра класса *DataStore* переданы следующие атрибуты: атрибут *name*, содержащий имя файла операционной системы "*AgentBlackboard.db*", в котором будут храниться разделяемые данные, атрибут *sharing_mode*, задающий режим совместного доступа к данным *shared_access,* и атрибут *access_mode*, указывающий, что логическая программа требует прав на изменение разделяемых данных (режим доступа к данным *modifying*).

В конструктор экземпляра класса *3DDataSources* передан атрибут *place*, содержащий структуру с двумя аргументами *shared*(*datastore*,"*3DDataSources*"). Этот атрибут указывает, что экземпляр базы данных *3DDataSources* будет находиться под управлением класса *DataStore,* причём содержимое базы данных в контексте экземпляра класса *DataStore* будет иметь уникальный идентификатор "*3DDataSources*". Если продолжить аналогию с реляционными базами данных, то "*3DDataSources*" – это имя (обобщённой) реляционной таблицы в разделяемой базе данных "*AgentBlackboard.db*".

При создании экземпляра класса *Main* будет автоматически осуществлена серия операций с разделяемыми данными. С помощью встроенного предиката *open* будет открыт доступ к разделяемым данным "*AgentBlackboard.db*". После этого будет открыта транзакция с возможностью изменения данных, все существующие записи в таблице "*3DDataSources*" будут удалены, экземпляр класса *Main* запишет в базу данных сам себя и закроет транзакцию. После успешного изменения разделяемых данных "*AgentBlackboard.db*" доступ к данным будет закрыт с помощью предиката *close*.

```
goal:-
     datastore ? open,
     database  ?  begin_transaction
('modifying'),!,
     database ? retract_all(),
     database ? insert(self),
     database ? end_transaction,
     datastore ? close.
goal:-!.
```

После занесения в разделяемую базу данных, экземпляр класса *Main* становится доступным для других логических агентов. В частности, логический агент, желающий получать информацию о температуре поверхности наблюдаемых людей, должен прочитать этот экземпляр класса из базы данных и вызвать в нём предикат *register_consumer*. В качестве аргумента он должен передать в предикат самого себя. Предикат *register_consumer* занесёт его во внутреннюю базу данных *consumers* (потребители рассматриваемого логического агента).

```
register_consumer(ExternalAgent):-
     consumers?insert(ExternalAgent).
```

При каждом вызове предиката *frame_obtained* класса *3DVideoSupplier*, рассмотренного в предыдущем разделе, вызывается предикат *notify_all_consumers*, который с помощью поиска с откатом извлекает из базы данных *consumers* по очереди всех потребителей и пересылает им очередную порцию данных.

```
notify_all_consumers(Skeletons,Image):-
     consumers?find(ExternalAgent),
          notify_consumer(
               ExternalAgent,
Skeletons,Image),
          fail.
notify_all_consumers(_,_).
```

Предикат *notify_consumer* осуществляет удалённый вызов предиката *new_frame* во внешнем мире *ExternalAgent*.

```
notify_consumer(ExternalAgent,Skeletons,Image):-
    [ExternalAgent] [<<] new_frame(Skeletons, Image).
```

Синтаксические обозначения, использованные в данном примере, имеют следующий смысл. Разделитель << означает, что для передачи данных использовано так называемое «информационное» прямое сообщение [25, 26], которое является одной из разновидностей асинхронных сообщений, поддерживаемых Акторным Прологом. Квадратные скобки вокруг разделителя [<<] означают, что данное сообщение не подлежит буферизации, то есть если агент-получатель не успеет обработать это сообщение до того, как получит новое, сообщение будет потеряно. Квадратные скобки вокруг переменной [*ExternalAgent*], содержащей принимающего агента, означают, что удалённый вызов предиката *new_frame* должен быть осуществлён немедленно при исполнении рассматриваемой команды (в противном случае удалённый вызов был бы отменён при откате предиката *notify_all_consumers*).

Принимающий логический агент получит и обработает сообщение *new_frame(Skeletons,Image)*. В рассматриваемом примере принимающий агент должен проанализировать граф *Skeletons*, описывающий скелетоны наблюдаемых людей. Узлы и дуги графа, имеющие статус «непосредственно наблюдаемый» (*TRACKED*), будут сопоставлены с тепловизионным изображением *Image*. Величина яркости точек на тепловизионном изображении, координаты которых соответствуют двумерным координатам дуги графа, усредняется. Величины яркости узлов и средние значения яркости дуг заносятся в специальную таблицу. По мере поступления новых кадров (новых вызовов предиката *new_frame*), осуществляется усреднение значений таблицы, соответствующих каждому отдельному узлу и каждой отдельной дуге графа. Смысл такой обработки заключается в том, что в ходе произвольных движений и перемещений человека в кадре собирается информация о среднем значении температуры частей его тела (см. **рис. 2**).

В рамках рассмотренного примера были описаны принципы анализа и слияния многоканальной видеоинформации средствами объектно-ориентированного логического программирования, а также встроенные классы логического языка Акторный Пролог, разработанные для управления базами данных, сбора и распределённого анализа трёх- и двумерной видеоинформации. Полный текст примера входит в состав установочного пакета Акторного Пролога, опубликованного в открытом доступе на сайте [43]. С исходными текстами встроенных классов Акторного Пролога, использованных в примере, можно ознакомиться в репозитории GitHub [44].

6. ЗАКЛЮЧЕНИЕ

В рамках проекта «Акторный Пролог» разработаны и реализованы средства ввода и анализа двух- и трёхмерных видеоданных, в том числе средства слияния тепловизионных данных с трёхмерными данными, получаемыми с помощью времяпролётной камеры. Основным достижением данного проекта авторы считают то, что обеспечена

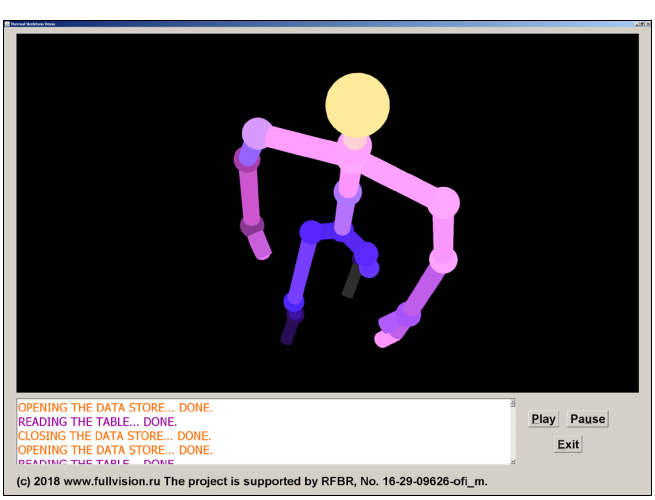

Рис. 2. *Логический агент, осуществляющий мониторинг температуры тела людей в зоне видеонаблюдения.*

эффективная обработка больших массивов разнородных данных средствами логического языка. Это достигается за счёт объектно-ориентированной архитектуры системы логического программирования Акторный Пролог, позволяющей инкапсулировать большие массивы данных в экземплярах специализированных встроенных классов так, чтобы большой объём данных не влиял на скорость логического вывода. Разработанные средства логического программирования открывают новые перспективы в области интеллектуального видеонаблюдения, а именно, позволяют проводить смысловой анализ видеосцен, то есть делать логические выводы на основе разнородной информации, описывающей контекст наблюдаемых предметов, действий людей и событий.

БЛАГОДАРНОСТИ

Исследование выполнено при поддержке РФФИ (проект № 16-29-09626-офи-м).

ЛИТЕРАТУРА

1. Rikke G, Moeslund TB. Thermal cameras and applications: a survey. *Machine Vision and Applications*, 2014, 25:245-262.
2. Nakagawa W, Matsumoto K, de Sorbier F, Sugimoto M, Saito H, Senda S, Shibata T, Iketani A. Visualization of temperature change using RGB-D camera and thermal camera. *European Conference on Computer Vision-ECCV 2014 Workshops,* 2015, LNCS 8925:386-400.
3. Rangel J, Soldan S, Kroll A. 3D thermal imaging: fusion of thermography and depth cameras. *Proc. Intern. Conf. on Quantitative InfraRed Thermography, 2014*; DOI: 10.21611/qirt.2014.035.
4. Han S, Gu X, Gu X. An accurate calibration method of a multi camera system. *Proc. Intern. Conf. on Life System Modeling and Simulation,* Singapore, Springer, 2017:491-501.
5. Dickens JS, van Wyk MA, Green JJ. Pedestrian detection for underground mine vehicles using thermal images. *AFRICON, 2011*:1-6; DOI: 10.1109/AFRCON.2011.6072167.
6. Møgelmose A, Bahnsen C, Moeslund TB, Clapés A, Escalera S. Tri-modal person re-identification with RGB, depth and thermal features. *Proc. IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2013*:301-307.
7. Susperregi L, Arruti A, Jauregi E, Sierra B, Martínez-Otzeta JM, Lazkano E, Ansuategui A. Fusing multiple image transformations and a thermal sensor with Kinect to improve person detection ability. *Engineering Applications of Artificial Intelligence*, 2013, 26(8):1980-1991.
8. Kristoffersen MS, Dueholm JV, Gade R, Moeslund TB. Pedestrian counting with occlusion handling using stereo thermal cameras. *Sensors,* 2016, 16(1):62.
9. Spremolla IR, Antunes M, Aouada D, Ottersten B. RGB-D and thermal sensor fusion: application in person tracking. *Intern. Conf. on Computer Vision Theory and Applications*: 610-617; doi:10.5220/0005717706100617.
10. Corneanu CA, Oliu M, Cohn JF, Escalera S. Survey on RGB, 3D, thermal, and multimodal approaches for facial expression recognition: history, trends, and affect-related applications. *IEEE transactions on pattern analysis and machine intelligence,* 2016, 38(8):1548-1568.
11. Berretti S, Daoudi M, Turaga P, Basu A. Representation, analysis and recognition of 3D humans: a survey. *ACM Transactions on Multimedia Computing, Communications and Applications,* 2018, 14(1s):1-35.
12. Skala K, Lipić T, Sović I, Gjenero L, Grubišić I. 4D thermal imaging system for medical applications. *Periodicum Biologorum*, 2011, 113(4):407-416.
13. Procházka A, Charvátová H, Vyšata O, Kopal J, Chambers J. Breathing analysis using thermal and depth imaging camera video records. *Sensors,* 2017, 17(6):1408.
14. Sun G, Matsui T, Kirimoto T, Yao Y, Abe S. Applications of infrared thermography for noncontact and noninvasive mass screening of febrile international travelers at airport quarantine stations. *Application of Infrared to Biomedical Sciences,* 2017:347-358.

15. Chernov G, Chernov V, Flores MB. 3D dynamic thermography system for biomedical applications. *Application of Infrared to Biomedical Sciences,* 2017:517-545.
16. Schönauer C, Vonach E, Gerstweiler G, Kaufmann H. 3D building reconstruction and thermal mapping in fire brigade operations. *Proc. 4th Augmented Human International Conference, ACM,* 2013:202-205.
17. Worch J-H, Bálint-Benczédi F, Beetz M. Perception for everyday human robot interaction. *KI – Künstliche Intelligenz,* 2016, 30(1):21-27.
18. Narváez FJY, del Pedregal JS, Prieto PA, Torres-Torriti M, Cheein FAA LiDAR and thermal images fusion for ground-based 3D characterisation of fruit trees. *Biosystems Engineering,* 2016, 151:479-494.
19. Kawasue K, Win KD, Yoshida K, Tokunaga T. Black cattle body shape and temperature measurement using thermography and Kinect sensor. *Artificial Life and Robotics,* 2017, 22(4):464-470.
20. Borrmann D, Nüchter A, Đakulović M, Maurović I, Petrović I, Osmanković D, Velagić J. A mobile robot based system for fully automated thermal 3D mapping. *Advanced Engineering Informatics,* 2014, 28(4):425-440.
21. Vidas S, Moghadam P, Bosse M. 3D thermal mapping of building interiors using an RGB-D and thermal camera. *IEEE International Conference on Robotics and Automation (ICRA*), 2013:2311-2318.
22. Yamaguchi M, Truong TP, Mori S, Nozick V, Saito H, Yachida S, Sato H. Superimposing thermal-infrared data on 3D structure reconstructed by RGB visual odometry. *IEICE Transactions on Information and Systems,* 2018, E101D(5):1296-1307.
23. Морозов АА. Акторный Пролог. *Программирование*, 1994, 5:66-78.
24. Morozov A.A. Actor Prolog: an object-oriented language with the classical declarative semantics. *Proc. Intern. Workshop on Implementation of Declarative Languages (IDL 1999*), Paris, France, 1999:39-53.
25. Морозов АА. Об одном подходе к логическому программированию интеллектуальных агентов для поиска и распознавания информации в Интернет. *Журнал радиоэлектроники*, 2003; http://jre.cplire.ru/jre/nov03/1/text.html.
26. Morozov AA. Logic object-oriented model of asynchronous concurrent computations. *Pattern Recognition and Image Analysis*, 2003, 13(4):640-649.
27. Morozov AA. Operational approach to the modified reasoning, based on the concept of repeated proving and logical actors. *CICLOPS,* 2007, 7:1-15.
28. Морозов АА, Сушкова ОС, Полупанов АФ. О проблеме введения средств распределённого многоагентного программирования в логический язык со строгой типизацией. *Журнал Радиоэлектроники*, 2016, 7; http://jre.cplire.ru/jre/jul16/9/text.pdf.
29. Morozov AA, Sushkova OS, Polupanov AF. Towards the distributed logic programming of intelligent visual surveillance applications. *Advances in Soft Computing: Proc. 15th Mexican Intern. Conf. on Artificial Intelligence, MICAI 2016,* Cancun, Mexico, 2017:42-53.
30. Morozov AA, Polupanov AF. Intelligent visual surveillance logic programming: implementation issues. *CICLOPS-WLPE*, 2014, AIB:31-45.
31. Morozov AA, Sushkova OS, Polupanov AF. A translator of Actor Prolog to Java. *RuleML,* 2015, 8.
32. Morozov AA, Vaish A, Polupanov AF, Antciperov VE, Lychkov II, Alfimtsev AN, Deviatkov VV. Development of concurrent object-oriented logic programming system to intelligent monitoring of anomalous human activities. *Proc. 7th Intern. conf. biomedical electronics and device*s, Spain, 2014:53-62.
33. Morozov AA. Development of a method for intelligent video monitoring of abnormal behavior of people based on parallel object-oriented logic programming. *Pattern Recognition and Image Analysis*, 2015, 25(3):481-492.

34. Morozov AA, Polupanov AF. Development of the logic programming approach to the intelligent monitoring of anomalous human behavior. *Proc. 9th Open German-Russian Workshop on Pattern. Recognition and Image Understanding (OGRW 2014*), Koblenz, University of Koblenz-Landau, 2015, 5:82-85.
35. Morozov AA, Sushkova OS, Polupanov AF. An approach to the intelligent monitoring of anomalous human behaviour based on the Actor Prolog object-oriented logic language. *RuleML*, 2015, DC and Challenge, 2015, 8.
36. Morozov AA, Vaish A, Polupanov AF, Antciperov VE, Lychkov II, Alfimtsev AN, Deviatkov VV. Development of concurrent object-oriented logic programming platform for the intelligent monitoring of anomalous human activities. *Proc. Intern. Joint Conf. on Biomedical Engineering Systems and Technologies*, Heidelberg, Springer, 2015, 511:82-97.
37. Morozov AA, Sushkova OS. Real-time analysis of video by means of the Actor Prolog language. *Computer Optics,* 2016, 40(6):947-957.
38. Morozov AA, Sushkova OS, Vaniya SM. Development of methods and algorithms based on object-oriented logic programming for video monitoring of laboratory rodents. *13th Intern. Conf. on Signal-Image Technology and Internet-Based Systems*, IEEE, 2017:311-318; DOI: 10.1109/SITIS.2017.59.
39. Морозов АА, Сушкова ОС. Виртуальная машина низкоуровневой обработки видеоизображений в Акторном Прологе. *Сб. трудов IV межд. конф. и молодёжной школы «Информационные технологии и нанотехнологии» (ИТНТ-2018), Самара,* Самарский нац. иссл. ун-т, 2018:1275-1285.
40. Морозов АА, Сушкова ОС. О разработке методов и алгоритмов на основе объектно-ориентированного логического программирования для видеомониторинга лабораторных крыс. *Сб. трудов IV межд. конф. и молодёжной школы «Информационные технологии и нанотехнологии» (ИТНТ-2018), Самара,* Самарский нац. иссл. ун-т, 2018:1182-1192.
41. Morozov AA, Sushkova OS, Polupanov AF. Object-oriented logic programming of 3D intelligent video surveillance: The problem statement. *IEEE 26th Intern. Symposium on Industrial Electronics (ISIE)*, IEEE Xplore Digital Library, 2017:1631-1636.
42. Морозов АА, Сушкова ОС, Полупанов АФ. Объектно-ориентированное логическое программирование систем 3D интеллектуального видеонаблюдения: постановка задачи. *Радиоэлектроника. Наносистемы. Информационные технологии (РЭНСИТ*), 2017, 9(2):205-214; DOI: 10.17725/rensit.2017.09.205.
43. Morozov AA, Sushkova OS. The intelligent visual surveillance logic programming Web Site. Retrieved May 20, 2018, from http://www.fullvision.ru.
44. Morozov AA. A GitHub repository containing source codes of Actor Prolog built-in classes. Retrieved May 20, 2018, from https://github.com/Morozov2012/actor-prolog-java-library.

Морозов Алексей Александрович
к.ф.-м.н., с.н.с.
ИРЭ им. В.А. Котельникова РАН
11/7, ул. Моховая, Москва 125090, Россия
morozov@cplire.ru

Сушкова Ольга Сергеевна
к.т.н., с.н.с.
ИРЭ им. В.А. Котельникова РАН
11/7, ул. Моховая, Москва 125090, Россия
http://www.fullvision.ru
o.sushkova@mail.ru

Хохлова Маргарита Николаевна
аспирант
Le2i, FRE CNRS 2005, Университет Бургундии Франш-Конте
9 аллея Ален Савари, Дижон 21000, Франция
margokhokhlova@gmail.com
Миньо Кирилл
доктор наук
Университет Бургундии-Франш-Конте
9 аллея Ален Савари, Дижон 21000, Франция
cyrille.migniot@u-bourgogne.fr
Петрова Надежда Геннадиевна
вед. инженер
ИРЭ им. В.А. Котельникова РАН
11/7, ул. Моховая, Москва 125090, Россия
petrova@cplire.ru.

DEVELOPMENT OF AGENT LOGIC PROGRAMMING MEANS FOR MULTICHANNEL INTELLIGENT VIDEO SURVEILLANCE

Aleksei A. Morozov, Olga S. Sushkova, Nadezhda G. Petrova
Kotelnikov Institute of Radioengineering and Electronics of RAS, http://www.fullvision.ru
Moscow 125009, Russian Federation
Margaret N. Khokhlova, Cyrille Migniot
University Bourgogne Franche-Comte, http://www.ubfc.fr
Dijon 21000, France
morozov@cplire.ru, o.sushkova@mail.ru, margokhokhlova@gmail.com, cyrille.migniot@u-bourgogne.fr, petrova@cplire.ru

Abstract. **Experimental means developed in the Actor Prolog parallel object-oriented logic language for implementation of heterogeneous multichannel intelligent visual surveillance systems are considered. These means are examined by the instance of a logic program for permanent monitoring of people's body parts temperature in the area of visual surveillance. The logic program implements a fusion of heterogeneous data acquired by two devices: (1) 3D coordinates of human body are measured using a time-of-flight (ToF) camera; (2) 3D coordinates of human body skeleton are computed on the base of 3D coordinates of the body; (3) a thermal video is acquired using a thermal imaging camera. Special built-in classes are developed in the Actor Prolog language for the acquisition and analysis of 3D and 2D video data: the Kinect and KinectBuffer classes implement interaction between the logic program and the ToF camera of the Kinect 2 device; the BufferedScene class implements storing and transferring 3D images; the Canvas3D class implements 3D graphics based on the Java3D open source library. In the considered example, the thermal video is projected to the 3D surface of the human body; then the temperature of the human body is projected to the vertices and edges of the skeleton. A special logical agent (i.e., the logic program written in Actor Prolog) implements these operations in real-time and transfers the data to another logical agent. The latter agent implements a time average of the temperature of the human skeletons and displays colored 3D images of the skeletons; the average temperature of the vertices and edges of the skeletons is depicted by colors. A method of remote predicate calls is used for the interaction of the logical agents; this method was developed and implemented in Actor Prolog to support the agent logic programming paradigm. The logic programming means under consideration are developed for the purpose of the implementation of logical analysis of video scene semantics in the intelligent visual surveillance systems.**

Keywords: **intelligent video surveillance; thermal imaging, Kinect 2, three-dimensional vision; object-oriented logic programming; the Actor Prolog language; skeletons; recognition of complex events; machine vision; technical vision**
УДК 510.663; 519.68:007.5; 519.68:681.513.7; 681.3.06
Bibliography – 44 references *Received 03.06.2018*
RENSIT, 2018, 10(1):101-116 **DOI: 10.17725/rensit.2018.10.101**