# DEVELOPMENT OF AGENT LOGIC PROGRAMMING MEANS FOR MULTICHANNEL INTELLIGENT VIDEO SURVEILLANCE

**Aleksey A. Morozov, Olga S. Sushkova, Nadezhda G. Petrova**

Kotelnikov Institute of Radioengineering and Electronics of RAS, http://www.cplire.ru
Moscow 125009, Russian Federation

**Margarita 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.* **This paper proposes the experimental means developed in the Actor Prolog parallel object-oriented logic language for implementation of heterogeneous multichannel intelligent visual surveillance systems. These means are examined by the instance of a logic program for permanent monitoring of the peoples' 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 the human body are measured using a time-of-flight (ToF) camera, 3D coordinates of the human body skeleton are computed on the base of these 3D coordinates of the body; (2) 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 of the logic program with the ToF camera of the Kinect 2 device; the *BufferedScene* class implements storing and transfer of 3D images; the Canvas3D class implements 3D graphics based on the Java3D open source library. In the example under consideration, 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 that is 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 the 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 for supporting the agent logic programming paradigm. The logic programming means under consideration are developed for the purpose of the implementation of logical analysis of the semantics of the video scenes in the intelligent visual surveillance systems.**

*Keywords:* **intelligent video surveillance; thermal imaging, Kinect, three-dimensional vision; object-oriented logic programming; the Actor Prolog language; skeletons; recognition of complex events; machine vision; technical vision**

**UDC 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**

**Contents**

## 1. INTRODUCTION

This paper describes the development of a software platform for distributed/decentralized logical analysis of heterogeneous multichannel data in the intelligent video surveillance systems. The platform is based on the Actor Prolog object-oriented logic language. This platform is developed for the purpose of the implementation of logical analysis of the semantics of the video scenes. The idea of the semantic analysis of the video scenes suggests that an intelligent video surveillance system is not only able to recognize, but also to estimate semantics of certain objects, actions, and events in the context of a given video scene. Examples of semantic analysis of video are the analysis of people behavior and, in particular, the analysis of social interactions. People behavior analysis implies, in contrast to the people action analysis, the analysis of context, that is, supplementary circumstances linked with the place, the time, preceding and following events, actions of other people, presence and absence of objects, etc.

The analysis of video scenes semantics has obvious application areas in medicine, education, and art, but in the authors' opinion, the issue of the day is recognition of suspicious behavior of people by the intelligent visual surveillance systems. For instance, the analysis of social communications between people (handshakes, aggressive behavior, etc.) is necessary to reconstruct the complete picture of what happens in a certain building or territory which can help to recognize unauthorized persons and/or offenders. Recognition of the social status and roles of people by the appearance (for instance, detection of the officials, the police, the military, etc.) is necessary to check the adequacy of the people appearance and their actions (for instance, to recognize criminals or terrorists masked by the police uniform).

Mathematical logic and logic programming are convenient means for description and analysis of the context of observed events. The problem of analysis of the video scenes is addressed in various areas of mathematics and computer science, such as temporal and spatial logic, real-time automatic theorem proving, distributed and parallel logical inference, object-oriented logic programming, etc. With all these research areas, the possibility of the practical use of these means is determined by the presence of source information that is enough for inferring useful logical conclusions. For instance, the appearance of sensors for 3D data acquisition (such as time-of-flight cameras, stereo cameras, flash LIDARS, etc.) ensures a possibility for the detailed analysis of the people behavior including the complex social interaction and the manipulation with objects. It is obvious that further progress in the area will be linked with the use of neural networks for the recognition of human poses and objects. Nevertheless, the authors consider the mathematical logic as the most appropriate mathematical language for the declarative description of complex people behavior and other events on the base of combining the descriptions of elementary events and actions. Also, we consider the logic programming as the most useful, adaptable, and practice-relevant tool for the implementation of the real-time logical inference.

Note that the logic programming is not a single approach for the logical description and analysis of the semantics of the video scenes; there are research intelligent visual surveillance systems based on rules, fuzzy logical inference, Markov

logic networks, etc. A distinctive feature of the logic-programming-based approach to the intelligent video surveillance system development is that the logic program is not just a set of logical rules/formulae, but also data processing instructions. In other words, the logic program has not only declarative but also operational semantics. It means that the logic programmer has a possibility to control the logical inference during the analysis of given people behavior and/or complex event to account the speed of data processing and the size of the data to be processed. Thus the application of a logic language instead of a plain set of logical rules opens a possibility to use logical inference on the multimedia/multichannel data efficiently in the framework of intelligent video surveillance systems.

This paper addresses the development of the means of the object-oriented and agent logic programming in the context of the problem of the semantic analysis of the video scenes. The experience of the application of the logic programming for the video data analysis indicates that this research activity yields perspective results in the area of video data analysis and, at the same time, gives new ideas and means in the area of logic programming. The point is that the logic programming and the first logic language Prolog were designed and developed as the means for symbolic processing in the framework of artificial intelligence. Thus, the real-time data processing and the processing of big data arrays (including images, videos, and multimedia data) were traditionally considered as unsuitable areas for the application of the logic languages. This fact left an imprint on the syntax of logic languages, the implementation of compilers and built-in libraries, etc. The experience of the application of the logic programming in the area of the intelligent video surveillance ensured a possibility to discover bottlenecks in the logic languages and compilers that gives new prospects for the development of the object-oriented/agent logic programming and translators of Prolog to imperative languages.

The object-oriented logic programming is a research area that addresses the development of programming languages that are object-oriented and logic simultaneously. This research direction is continuously developed during recent decades, but unfortunately is still an exotic direction in the framework of logic programming; it suffices to say that various researchers use the object, class, and inheritance terms in different ways. This practice is probably implied by the fact that both the pure mathematicians and computer scientists are involved in the area of logic programming; the mathematicians often consider the object-oriented features as a non-sufficient aspect of the logic languages and the programmers often ignore the research in the area of the model-theory semantics of the logic languages. Today the experience of the application of the logic programming for the multimedia data analysis stimulates the research in the area of the object-oriented logic programming.

The urgent need for the development and application of the object-oriented logical instruments was caused by the problem of efficient processing of big heterogeneous/multimedia data arrays in declarative languages. The Prolog language uses lists and structures for this purpose because these data items have a one-to-one correspondence with the elements of the first order logic (Skolem constants, numbers, variables, etc.). This fact does not decrease the descriptive power of the Prolog language, but it makes it

an inconvenient tool for big data processing. The object-oriented means give a solution to this problem because one can encapsulate the big arrays of data in the instances of specialized built-in classes without violation of the declarative semantics of the logic programs.

In this paper, the basic ideas of using the Actor Prolog object-oriented logic language for the multichannel/multimedia data analysis are described by the example of processing of 3D video data acquired using the Kinect 2 device (Microsoft, Inc.) and 2D thermal imaging video acquired using the Thermal Expert V1 camera (i3system, Inc.). The distributed logic programming means of the Actor Prolog language are discussed by the example of the communicating logical agents that analyze 3D video data and implement a fusion of these data with the thermal video.

The fusion of the thermal imaging video with data of different modalities (including the 3D video data) is a rapidly developing research area [1-4]. The main applications of this concept are detection, recognition, and re-identification of people in the intelligent video surveillance systems [5-11], medical applications including non-invasive detection of dangerous diseases in the airport [12-15], rescue party support [16], human-computer interaction [17], agriculture [18, 19], and automatic check of equipment and odometry [20-22]. This paper considers the problem of remote measurement of human body parts temperature dynamics in the video surveillance area.

In the next section, the architecture and basic principles of the Actor Prolog logic programming system are described. In the third section, a set of classes developed by the authors for the Actor Prolog language are proposed for the acquisition and analysis of 3D video data. The fourth section gives an example of a logical agent that inputs 3D data of the body surface of the people under the video surveillance and implements a fusion of this data with the thermal imaging video. In the fifth section, the basic principles and means for the communication of the logical agents in the Actor Prolog language are considered.

## 2. THE ARCHITECTURE OF THE ACTOR PROLOG LOGIC PROGRAMMING SYSTEM

Actor Prolog is a logic programming language developed in the Kotel'nikov Institute of Radio Engineering and Electronics of Russian Academy of Sciences [23-27]. Actor Prolog was designed initially as an object-oriented and logic language simultaneously, that is, the language implements classes, instances, and inheritance; at the same time, the object-oriented logic programs have model-theory semantics. The Actor Prolog language supports means for definition of data types (so-called domains), the determinacy of predicates, and the direction of data transfer in the subroutine arguments [28, 29]. These means are vital for the industrial applications of the logic programming because the experience demonstrated that it is very difficult to support and debug big and complex logic programs without these means. Actor Prolog is a parallel language; there are syntax means that support creation and control of communicating parallel processes. These syntax means of the language provide the model-theory semantics of the logic programs as well, but only when certain restrictions are imposed on the syntax and structure of the programs [25, 26].

A distinctive feature of the Actor Prolog logic programming system is the fact that the

logic programs are translated in Java code and executed by the standard Java virtual machine [30, 31]. This scheme of logic program execution was developed, mainly, to ensure the stability of the programs and prevent possible problems with the memory management. Another important feature of this scheme is the fact that it ensures high extensibility of the logic language: one can easily add necessary built-in classes to the language.

One can add a new built-in class to the Actor Prolog language in the following way. During the translation of the logic program, it is converted to the set of Java classes. There are special syntax means to declare some automatically generated Java classes as descendants of external Java classes that were created manually by the programmer. Thus, it is enough to implement a new built-in class in Java and link it with the logic program in the course of translation to make this class the built-in class of Actor Prolog. Currently, a set of built-in classes of Actor Prolog are implemented totally in pure Java; other built-in classes are Java interfaces with open source libraries implemented in C++. The examples of the former classes are: the *Database* class that implements a simple data management system; the *File* class that supports reading and writing files; the *WebResource* class that implements data acquisition from the Web. The examples of the latter classes are: the *FFmpeg* class that links Actor Prolog with the FFmpeg open source library for video reading and writing; the *Java3D* class that implements 3D graphics; the *Webcam* class that supports video data acquisition. The authors consider the translation to Java as an architectural solution that helps to develop and debug rapidly new built-in classes. The speeding-up of the software life cycle is caused by the fact that the Java language (in comparison with the C++ language) prevents the appearance of the bugs linked with the incorrect memory access and out-of-range array access that can be very difficult to detect.

## 3. BUILT-IN CLASSES SUPPORTING INTELLIGENT VISUAL SURVEILLANCE

A set of built-in classes for 2D and 3D video acquisition and analysis is implemented in the Actor Prolog logic programming system. These built-in classes were developed mainly in the course of experimenting with the methods of intelligent video surveillance.

First of all, the *ImageSubtractor* built-in class was developed in Actor Prolog that implements a set of standard low-level image processing operations: background subtraction, Gaussian and rank filtration, extraction of blobs, estimation of sizes and velocities of the blobs, computation of additional statistical features describing the evenness of the movements, etc. All data matrices used on various stages of the image processing are encapsulated in the instances of the ImageSubtractor class, that ensures high performance of video processing. The logical rules written in Actor Prolog manipulate only with the results of the low-level video processing, that is, graphs describing the trajectories, velocities, and other features of the blobs. One can see a detailed description of the *ImageSubtractor* built-in class and examples of its application for the low-level and high-level image processing in [30, 32-37].

The next stage of the development of the means for the low-level video analysis in Actor Prolog was the creation of the video processing virtual machine [38, 39] that was implemented in the *VideoProcessingMachine*

built-in class. This virtual machine was developed in the course of experimenting with the analysis of the biomedical videos, namely, the analysis of the behavior of the laboratory rodents. The distinctive feature of these videos is the presence of the blobs of various types in the image; different blobs require different algorithms of low-level image processing [38, 40]. The programmer can define a sequence of low-level video processing operations and download it to the instance of the virtual machine; this sequence will be repeated automatically for every incoming frame of the video. The typical operations are: processing the image in a pixel format; selection and processing foreground pixels; extraction and tracking blobs in the video; etc. All the intermediate data matrices are encapsulated inside the instance of the *VideoProcessingMachine* class in the same way as in the *ImageSubtractor* class.

In this paper, new means of the Actor Prolog language are described that were developed for 3D data acquisition and analysis using the Kinect 2 device. These means and specialized built-in classes of Actor Prolog were created for the experimenting with the 3D intelligent visual surveillance [41, 42]. The developed means are based on the same ideas as the means for the 2D video analysis:

1. The low-level and high-level video processing stages are separated.
2. The low-level video processing stage is implemented in special built-in classes that encapsulate all intermediate data matrices.
3. The high-level video processing stage is implemented by the logical rules; the data is processed in the form of graphs, lists, and other terms of the logic language.

The data processing scheme was adapted to the following properties of 3D video data:

1. The data are heterogeneous (multimodal). For instance, the Kinect 2 device provides several data streams simultaneously. They are the frames describing 3D point clouds, the infrared imaging frames, the conventional colored (RGB) frames, the frames that describe coordinates of human skeletons, etc.
2. 2The size of 3D video data is usually huge. A typical personal computer is not powerful enough to store in real time all raw data of Kinect 2 to the hard disk. Thus, a preferable scheme of 3D data processing includes preliminary real-time analysis of the data and storing/networking the intermediate results of the analysis.

There are two built-in classes in Actor Prolog that support 3D video acquisition and analysis: *Kinect and KinectBuffer*. The former class implements communication between the logic program and the Kinect 2 device. The latter class implements low-level analysis as well as reading and writing 3D data files. The definitions of these classes including the definitions of data types (domains) and predicates are placed in the "Morozov/Kinect" package of Actor Prolog.

The *KinectBuffer* class is the most important element in the 3D data processing scheme. The instance of the *KinectBuffer* class can be used in the following modes: data acquisition from Kinect 2; reading data from the file; playing 3D video file; writing data to the file. The playing-3D-video-file and reading-from-the-file modes differ in that in the former mode the *KinectBuffer* class reads data and transfers them to the logic program in real-time; in the reading-from-the-file mode, the programmer has to control reading of every frame of the video.

The *operating_mode* attribute of the *KinectBuffer* class is to be used to select the operating mode of the instance of the class: '*LISTENING', 'READING', 'PLAYING*', or '*RECORDING*' correspondingly. The *KinectBuffer* class can be used alone to read/write 3D videos; however one has to use it in connection with the *Kinect* class to acquire data from the Kinect 2 device. In this mode, one has to create an instance of the *Kinect* class and transmit it to the constructor of the *KinectBuffer* class instance; the *input_device* attribute is to be used as the argument of the constructor. An example of the logic program that reads and processes 3D video data from the file is considered in the next section.

## 4. ACQUISITION AND FUSION OF 3D AND THERMAL IMAGING VIDEO DATA

Let us consider an example of the logic program that reads and implements a simple analysis of 3D video data that were acquired using ToF camera of the Kinect 2 device. Fragments of source code written in Actor Prolog will be demonstrated below with comments; naturally, the source code is reduced, because the purpose of the example is just to demonstrate the scheme of the data processing using the Kinect and *KinectBuffer* classes.

Let us define the 3DVideoSupplier class that is a descendant of the *KinectBuffer* class. The operating_mode attribute has the *PLAYING* value that indicates that the data are to be read from the "My3DVideo" file in the playing-3D-video-file mode.

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

In the course of the creation of the *3DVideoSupplier* class instance, it downloads a lookup table from the "MyLookUpTable.txt" file. This lookup table establishes the correspondence between the 3D coordinates measured by the ToF camera and 2D coordinates on the 2D thermal imaging video. After that, the reading from the file is activated using the *start* predicate of the *KinectBuffer* class.

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

The *KinectBuffer* class supports 2D and 3D lookup tables. The lookup table is supposed to be computed and stored in the text file in advance during the calibration of the video acquisition system. The 2D lookup table is a matrix K of the same size as the Kinect 2 infrared video frame. Each cell $(i, j)$ of the matrix contains coordinates $x$ and $y$ on an image $T$; in the example under consideration, $T$ is the thermal image. Thus, the $T$ image is to be projected to 3D surfaces investigated using the ToF camera. The 3D lookup table is also a matrix, but the cell of this matrix contains quadratic polynomial coefficients that are necessary for the computation of the coordinates on the $T$ image, but not the $(x, y)$ coordinates themselves. Each cell $(i, j)$ of the matrix contains six coefficients $p_1$, $p_2$, $p_3$, $q_1$, $q_2$, and $q_3$. The coordinates on the $T$ image are computed using the quadratic polynomial depending on the inverse value of the distance $d(i,j)$ in meters between the ToF camera and the surface of the object to be investigated:

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

In the example under consideration, the thermal image is projected to the 3D surface

during the processing of every frame of 3D video. The *frame_obtained* predicate is invoked automatically in the instance of the *KinectBuffer* class when a new frame of 3D video is read from the file. The programmer utilizes the *commit* predicate to inform the *KinectBuffer* class that s/he is going to process this frame. After that, all the predicates of the *KinectBuffer* class operate with the content of this particular frame until the *commit* predicate is called again. The logic program gets the *Time1* time of the frame in milliseconds using the *get_recent_frame_time predicate*. Then the number of corresponding thermal imaging frame is calculated using this information. The *texture_time_shift* attribute contains a value of the temporal shift between the 3D and thermal videos. The *texture_frame_rate* attribute contains the frame rate of the thermal imaging video.

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

Suppose the thermal imaging video is converted to the separate frames. The *frame_obtained* predicate computes the name of the corresponding JPEG file using the number of the frame. Then it uses the load predicate to read the frame and stores it to the *image* instance of the *BufferedImage* built-in class. The *get_recent_scene* predicate of the *KinectBuffer* class is called; this predicate creates a 3D surface on the base of the ToF camera data and projects given texture to this surface. The texture is transferred to the predicate by the second argument *image* that contains an instance of the *BufferedImage* class. The lookup table loaded above is used for the implementation of the texture projection. The created 3D surface is returned from the *get_recent_scene* predicate via the first argument. This argument has to contain an instance of the *BufferedScene* built-in class. The *BufferedScene* built-in class implements storing and transferring of the 3D data. In particular, the content of the *BufferedScene* class instance can be inserted into the 3D scene displayed by the means of the Java3D graphics library. In the considered example, the *set_node* predicate is used for this purpose. It replaces the "MyLabel" node of the 3D image created using the *graphics_window* instance of the *Java3D* built-in class.

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

The *get_recent_mapping* predicate of the *KinectBuffer* built-in class operates approximately in the same way as the *get_recent_scene* predicate. The difference is the following. It does not create a 3D surface and the results of the projection of the texture to the 3D surface are returned in a form of a convenient 2D image. In this example, the *get_recent_mapping* predicate stores the created image to the *buffer2D* instance of the *BufferedImage* built-in class. The *get_skeletons* predicate of the *KinectBuffer* class returns a list of the skeletons detected in the current frame. The skeletons are graphs that contain information about the coordinates of the human body, head, arms, and legs. In the logic program, the graphs are described using the standard simple and compound

terms: lists, structures, underdetermined sets, symbols, and numbers [41, 42]. In the example, the skeletons and thermal images are transferred to another logical agent that implements further analysis and fusion of the data. The routine of the data transfer between the logical agents will be considered in the next section. Note that the image to be transferred from the *buffer2D* instance of the *BufferedImage* class is converted to the term of the *BINARY* type. The *get_binary* predicate of the *BufferedImage* class is used for this purpose.

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

The results of the fusion of 3D and thermal imaging data implemented by the logical agent under consideration are demonstrated in **Figure 1.**

In the next section, a scheme of communication between the logic programs (logical agents) based on the *Database* and *DataStore* built-in classes and the mechanism of the remote predicate calls are discussed.

## 5. A LINK STARTUP AND COMMUNICATION BETWEEN THE LOGICAL AGENTS

The mechanism of remote predicate calls is a special feature of the Actor Prolog language that was developed to support distributed/decentralized logic programming. The idea of this mechanism is that any object of the logic program (logical agent) can be transferred to another logical agent; after that, the new owner of the object can remotely and asynchronously invoke the predicates of the object [28, 29]. Note that in terms of Actor Prolog, the object is a synonym of the world and the

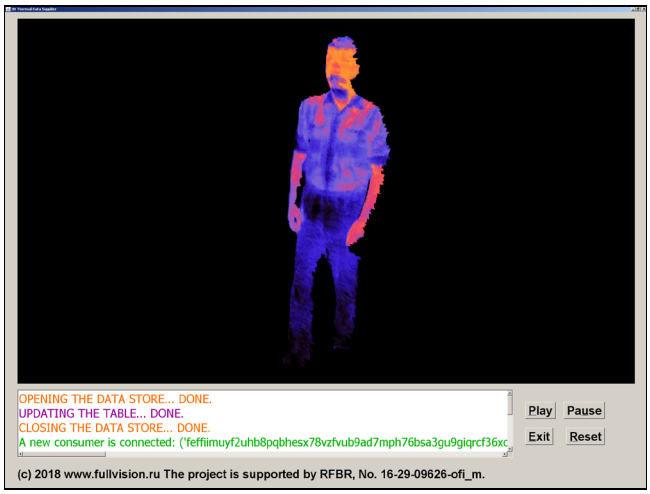


**Fig. 1.** *An example of a logical agent that collects and transfers heterogeneous multichannel data (3D video data and thermal imaging data).*

instance of a class. The complication of the development of this mechanism was that Actor Prolog has a strong type system and, therefore, the type system of the language did not provide a possibility for the agents to link and communicate dynamically without a preliminary exchange of the information about the types of the data to be transferred. Usually, the languages with strong type systems require an exchange of data type definitions on the stage of compilation of the agents, but in the Actor Prolog language, another solution was elaborated. In the distributed version of Actor Prolog, a combined type system was developed, that is, the strong type system was partially softened in the case when the inter-agent data exchange is performed. To be more precise, the types of the arguments in the remote predicate calls are compared by structure, but not by names; moreover, the check of that the external object implements a required predicate is postponed until this predicate call is to be actually performed. In all other cases, the standard static type check is implemented in the language. The combined type system

unites the advantages of the strong type system that is used for generation of fast and reliable executable code and the possibility of the dynamic type check during the inter-agent data exchange.

An instance of a class of the logical agent is to be transferred to other agents somehow to establish a connection between the agents. The instance of the class can be transferred via an operating system file, a shared database, or just in a text form by an E-mail. In the example under consideration, the built-in database management system of Actor Prolog is used for this purpose. This system is implemented by the *Database* and *DataStore* built-in classes of the language.

Let us define the *Main* class that will implement two roles in the logic program: the execution of the logic program begins with the creation of an instance of the *Main* class in accordance with the definition of the language; the instance of the class is to be transferred to another agent to establish a link and for the communication in the example under consideration.

The *Main* class contains several slots (that is, the class instance variables). The *datastore* slot contains an instance of the *DataStore* built-in class. The *database* slot contains an instance of the *3DDataSources* class that is a descendant of the *Database* built-in class. The *video_supplier* slot contains an instance of the 3*DVideoSupplier* class that was considered in the previous section. The *consumers* slot contains an instance of the *ConsumersList* class that is a descendant of the *Database* built-in class; the *consumers* database keeps a list of external logical agents that requested information from the agent under consideration.

```
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');
```

The *Database* built-in class implements a simple database management system that provides storing and searching the data of arbitrary structure; the operations of this kind are typical for the Prolog-like logic languages. One might say that the convenient relational databases are a special case of the Prolog databases when the database is used only for storing the data of the structure type, that is, the records with a name and a list of arguments. Note that the *Database* class is designed for storing data in the main memory of the computer, that is, for the management of the temporary data. The temporary data can be stored to the file or loaded from the file if necessary.

A database management mechanism of a more high level is to be used to control data that are shared between several logic programs. This mechanism is implemented in the *DataStore* built-in class. The *DataStore* class can coordinate and control the operation of one or several instances of the *Database* class. For instance, one can read from or write to the file the content of several instances of the *Database* class at once. Another useful mechanism of the *DataStore* class is the support of the shared data access, that is, it can link the instances of the *Database* class with the operating system files and automatically transfer the updates of the data in the memory of one

logic program to the memory of other logic programs. The data integrity is guaranteed by the standard mechanism of transactions. These instruments of the *DataStore* class are used in the example under consideration.

In the code above, the constructor of the *DataStore* class instance accepts the following input arguments: the name attribute that contains the name of the "AgentBlackboard.db" file that is to be used for the shared data storage; the *sharing_mode* attribute that assigns the shared mode of the data access (the *shared_access* mode); the *access_mode* attribute that indicates that the logic program demands the privileges for shared data modification (the *modifying* mode).

The constructor of the *3DDataSources* class instance accepts the *place* argument that contains the *shared(datastore,"3DDataSources")* structure with two internal arguments. This argument indicates that the instance of the *3DDataSources* database will operate under the control of the *DataStore* class and the content of the database has the "3DDataSources" unique identifier in the namespace of the *DataStore* class instance. Using the analogy with the relational databases, "3DDataSources" is the name of a generalized relational table in the "AgentBlackboard.db" shared database.

In the course of the creation of the *Main* class instance, a sequence of operations on the shared data will be performed. The *open* predicate initiates the access to the "AgentBlackboard.db" shared data file. After that, a transaction will be opened with the data modification privileges. All the records in the "3DDataSources" database will be deleted and the *Main* class instance will store itself in the database. Then the transaction will be completed by the *end_transaction* predicate and the access to the "AgentBlackboard.db" database will be ended by the close predicate.

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

The instance of the *Main* class becomes available for other logical agents after the storing to the shaded database. In particular, an external agent can read this instance from the database and invoke the *register_consumer* predicate in the world to receive the information about the temperature of the people in the scope of the visual surveillance system. The external agent has to send himself as the argument of the *register_consumer* predicate. The *register_consumer* predicate of the former agent will store it to the *consumers* internal database.

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

During each call of the *frame_obtained* predicate of the *3DVideoSupplier* class, the *notify_all_consumers* predicate is called. This predicate uses the search with backtracking to extract one-by-one the receivers from the *consumers* database and send them the data set.

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

The *notify_consumer* predicate implements the remote call of the *new_frame* predicate in the *ExternalAgent* external world.

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

The syntax notation of this clause has the following semantics. The << symbol indicates that the so-called informational direct message is to be used for the data transfer. The informational direct message is a kind of asynchronous messages supported by the Actor Prolog language [25, 26]. The square brackets in the [<<] expression indicate that this message must not be buffered, that is, the message is to be discarded if the receiver did not process it before the receiving of the next message. The square brackets in the [*ExternalAgent*] expression indicate that the remote call of the *new_frame* predicate is to be performed immediately during the execution of the command under consideration; otherwise, the remote call will be suspended and afterward canceled during the backtracking of the *notify_all_consumers* predicate.

The receiving logical agent will accept and process the *new_frame*(*Skeletons,Image*) message. In the example, the receiving agent has to analyze the Skeletons graph describing the skeletons of the people observed by the intelligent video surveillance system. The vertices and edges of the graph with the *TRACKED* status (that means that the ToF camera observes them directly) will be selected and compared with the *Image* thermal imaging frame. The intensities of the pixels at the thermal image that correspond to the coordinates of the graph edges are averaged. The intensities of vertices and average intensities of edges are stored in a special table. In the course of the processing of new frames, a time average of the intensities of vertices and edges stored in the table will be implemented. The purpose of this processing is the estimation of the average temperature of the human body parts during the unrestricted movements of the people in the video scene (see **Figure 2**).

The principles of analysis and fusion of heterogeneous multichannel video data by the means of the object-oriented logic programming were discussed using the concrete example. A set of built-in classes of the Actor Prolog language for the database management as well as acquisition and distributed processing of 2D and 3D video data were considered. The complete code of the example considered in this paper is published in the installation package of the Actor Prolog system that is freely available in the Web Site [43]. The source codes of the built-in classes considered in the paper are available on GitHub [44].

## 6. CONCLUSION

In the Actor Prolog project, the means and tools for 2D and 3D video data acquisition and processing are developed and implemented including the means for the fusion of 3D data collected using a ToF camera and a thermal imaging camera. In the authors' opinion, the main result of the project is the logic language adopted for the effective processing of the big arrays of heterogeneous data. This is provided thanks to the object-oriented architecture of the Actor Prolog logic programming system that supplies the encapsulation of the big data arrays in the instances of specialized built-in classes. The developed logical instruments open new prospects in the area of intelligent visual surveillance, namely, they enable to implement semantic analysis of the video scenes, that is, to conduct logical inference on the base of heterogeneous data describing the context of the human actions, objects,

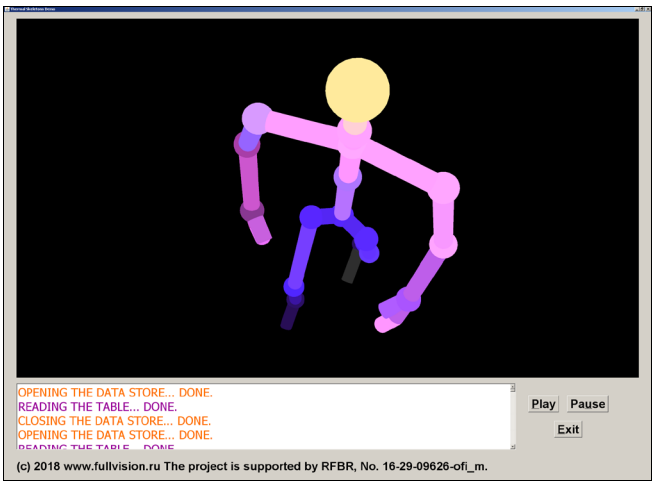


**Fig. 2.** *A logical agent that estimates the average temperature of the human body parts during the unrestricted movements of people in the video scene.*

and events observed by the intelligent video surveillance system.

**ACKNOWLEDGES**

*This research was supported by the Russian Foundation for Basic Research (grant number 16-29-09626-ofi_m).*

## REFERENCES

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. Morozov AA. Aktorny Prolog [Actor Prolog]. *Programmirovanie*, 1994, 5:66-78, (in Russ.).
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. Morozov AA. Ob odnom podkhode k logicheskomu programmirovaniyu intellektualnykh agentov dlya poiska i raspoznavaniya informatsii v Internet [On one approach to the logical programming of intelligent agents for searching and recognizing information on the Internet]. *Zhurnal radioelektroniki*, 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. Morozov AA, Sushkova OS, Polupanov AF. O probleme vvedeniya sredstv raspredelennogo mnogoagentnogo programmirovaniya v logicheskiy yazyk so strogoy tipizatsiey [On the problem of introducing the tools of distributed

multi-agent programming in a logical language with strong typing]. *Zhurnal radioelektroniki*, 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 devices*, 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. Morozov AA, Sushkova OS. Virtualnaya mashina nizkourovnevoy obrabotki videoizobrazheniy v Aktornom Prologe [Virtual machine for low-level video processing in Actor Prolog]. *Sat Works IV Int. conf. and youth school "Information technology and nanotechnology»* (ITNT-2018), Samara, Samara Univercity, 2018:1275-1285.
40. Morozov AA, Sushkova OS. O razrabotke metodov i algoritmov na osnove ob'ektno-orientirovannogo logicheskogo programmirovaniya dlya videomonitoringa laboratornykh krys [On the development of methods and algorithms based on object-oriented logic programming for video monitoring of

laboratory rats]. *Sat Works IV Int. conf. and youth school "Information technology and nanotechnology»* (ITNT-2018), Samara, Samara Univercity, 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. Morozov AA, Sushkova OS, Polupanov AF. Ob'ektno-orientirovannoe logicheskoe programmirovanie sistem 3D intellektualnogo videonablyudeniya: postanovka zadachi [Object-Oriented Logical Programming of 3D Systems of Intellectual Video Surveillance: Problem Statement]. *Radioelectronics. Nanosystems. Information Technologies* (*RENSIT*), 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*.