BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Project no. VI20172020068

Tools and Methods for Video and Image Processing to Improve Effectivity of Rescue and Security Services Operations (VRASSEO)

Technical report 2018

System architecture and ViAn Server

Jaroslav Zendulka, Vladimír Bartík, Tomáš Volf, Radim Kocman

Brno University of Technology
Faculty of Information Technology
Božetěchova 1/2
Brno, 612 66, Czechia

December 2018

Contents

Abstract

This technical report describes overview of a system for managing video-data and metadata and their subsequent use in order to extract context information and detects predefined situation patterns in an online manner. The report mainly presents the ViAn Sensing API implemented interface, support for ViAn Server sensing and data processing modules for event data storage, API performance testing and technical details for sensing module implementation.

1 Basic system overview

1.1 Hardware architecture

The hardware architecture of the system consists of sensing devices, field stations and a main central server. The deployment of the software components to this hardware nodes is shown in Fig. 1.

1. *Sensing device* — The task of the sensing device, typically a camera, is to acquire video data for further processing. Input video data will be captured by one or more cameras, will be streamed by the system's field station. At the same time, however, these video data on the input side may be pre-processed and streamed together with extracted metadata to the field station.

2. *Field stations* — Here, images/video and metadata extracted from external or internal modules are stored in the system. Field station system allows moving data to a central server and query data stored on the central server.

3. *Central server* — The server stores the previously captured data, namely images/videos and their metadata.

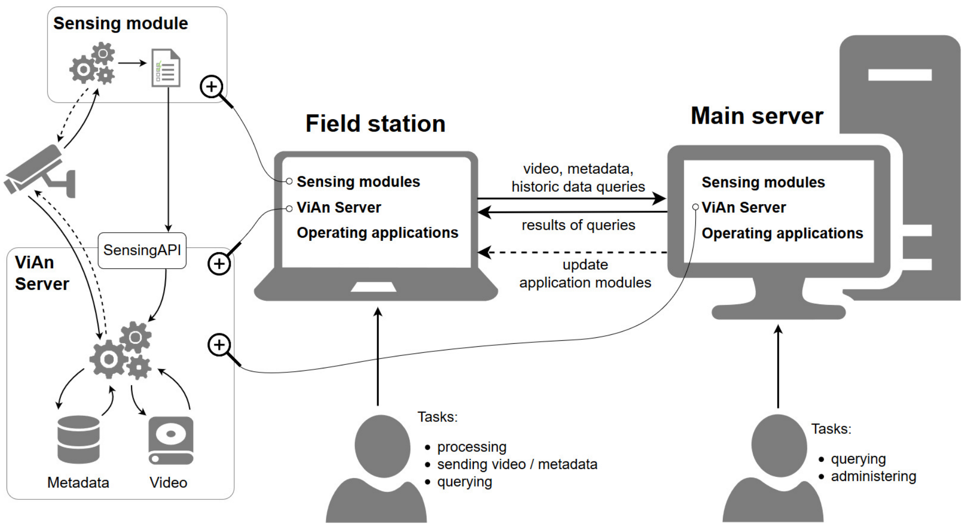

Figure 1: Basic concept of data storage in VRASSEO

1.2 Logical architecture

The logical architecture of the system consists of ViAn Server (VideoAnalysis Server), sensing modules and operational applications.

1. *ViAn Server* provides management of video data and metadata extracted from them. It provides database and basic analytical services. Its task is to support the management of video data and extracted metadata, including the support of some analytic tasks over these metadata ViAn Server provides the following set of APIs:

   (a) *SensingAPI* — the structure of data extracted from video by sensing modules definition and storing the data. Sensing modules communicate with ViAn Server via this API only.

   (b) *ViAnAPI* —- provides services to end users and to applications they use.

2. *Sensing modules* primarily serve to extract metadata from data captured by sensing devices. An example of such a module may be one extracting information about moving objects as people, cars etc. in the observed area. Several sensing modules can be located at one computing node.

3. *Operational applications* define the required task, manage its execution and visualize the results.

ViAn Server with processing and analytic services and operational applications can be deployed on both the field station and the main server. The field station deployments can provide limited functionality compared to the main server one, for example only some processing and analytic modules. Some modules can be physically deployed to devices other than a field station or a central server, such as a computer dedicated to more computationally intensive metadata extraction.

Field stations receive multimedia data and metadata from the sensing device and allow the user to quickly respond to the processing of this data. In addition, the user should be able to decide which data and metadata from his station should be synchronized with the main server, or query the server for relevant information (e.g., previous occurrences of the object in previous analyses).

The communication of the three basic system components in a case of starting a sensing module and receiving data produced by it is depicted in Fig. 2. First, the user logs on the application, which authenticates him to ViAn Server (1). Then he selects a sensing module he wants to use. The application stores the information about the selected module in ViAn Server, obtains an authentication token for the module (2) and sends the request to run the module to a corresponding node of modules (3). The module first sends a header specifying the structure of data that will be sent (4) and sends data repeatedly.

The communication is designed such a way that each of these components can be located on a separate physical device.

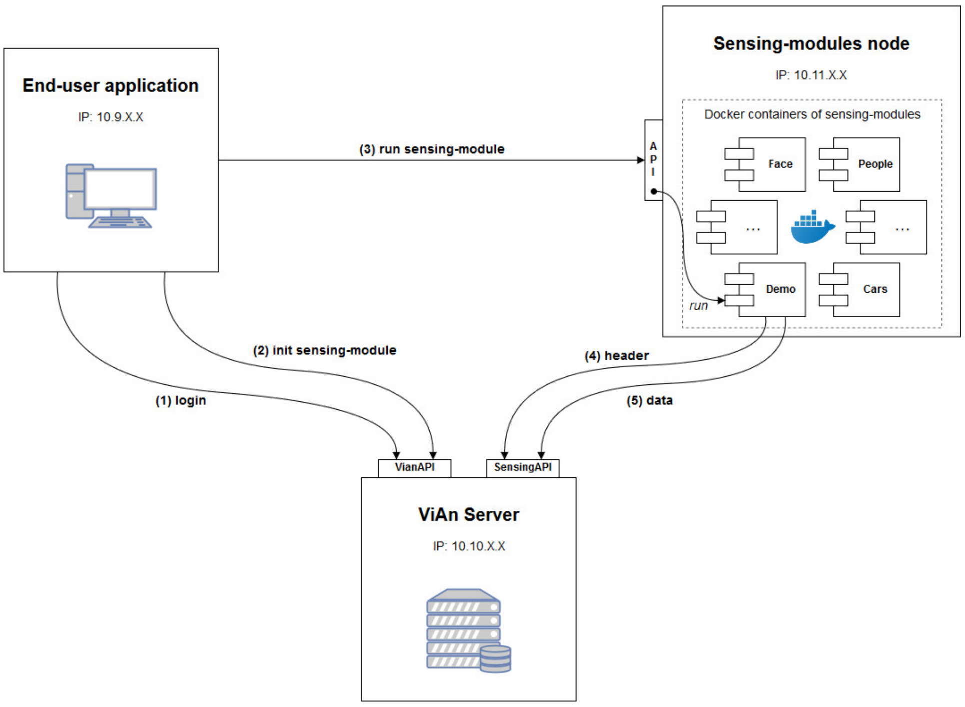

Figure 2: Communication of basic system components

2 ViAn SensingAPI

ViAn SensingAPI is a REST API interface provided by ViAn Server, which allows a sensing module to sent data to store in ViAn Server. This API uses general mechanism to store data at runtime, which means that it is not needed to predefine data on the server-side in advance - storage is created ad-hoc. Another benefit is simplification for developers in cases, when the data type ouput is changed; then ViAn server creates new storage for this data.

For each request it is needed at least following parts of request:

- `dataset` – passed in URI query string
- `access-token` – passed in request header

ViAn SensingAPI has a very limited number of endpoints, more precisely it has only 2 endpoints:

- `/data/header` – for definition data variables and their types
- `/data/store` – for sending (storing) data itself

Each endpoint return HTTP-code `200` if request has succeeded. At the same time is returned new timestamp of token validity and may be returned new access token also. *At the moment both these properties are not used, they are designed for future use.*

First, each module uses `/data/header` endpoint, only afterwards it can use `/data/store` endpoint as much as it needed.

2.1 Header endpoint: `/data/header`

Header endpoint is designed for data storage creation / preparation.

cURL request:

```
curl -X POST
"https://localhost/vian_sensingapi/data/header?dataset=test"
-H  "accept: application/json"
-H  "access-token: ***"
-H  "Content-Type: application/json"
-d "{\"foo\":\"int\", \"bar\":\"string\"}"
```

Response of request:

HTTP-code of succeeded response: `200`

```
{
  "status": "success",
  "data": {
    "access_token": "***",
    "token_validity": "2020-01-10T22:35:27+01:00"
  }
}
```

- `access_token` – new token for further request
- `token_validity` – timestamp of token validity otherwise token expires

At the moment both these properties are not used, they are designed for future use.

2.2 Data store endpoints: `/data/store`

Data store endpoint is designed for data sending (storing) itself.

cURL request:

```
curl -X POST
"https://localhost/vian_sensingapi/data/store?dataset=test"
-H  "accept: application/json"
-H  "access-token: ***"
-H  "Content-Type: application/json"
-d "{\"foo\":1234, \"bar\":\"ViAn module test\"}"
```

Response of request:

HTTP-code of succeeded response: `200`

```
{
  "status": "success",
  "data": {
    "access_token": "***",
    "token_validity": "2020-01-10T22:36:19+01:00"
  }
}
```

- `access_token` – new token for further request

- `token_validity` – timestamp of token validity otherwise token expires

At the moment both these properties are not used, they are designed for future use.

2.3 ViAn SensingAPI testing

To get a general idea how well the ViAn Sensing API performs and what is its possible throughput, we have prepared a suite of speed tests that simulates the usual communication between ViAn server and sensing modules. In the current state, the suite generates dummy detection messages that are being broadcasted and stored on the development server. The suite is designed as a collection of scripts written in Python that can be run repeatedly with various configurations. The tests are in their core based on standard performance measuring tools for the used technologies: ApacheBench for Apache HTTP server and pgbench for PostgreSQL database. In the current development environment, we have tasted the communication throughput of 1 up to 20 simultaneously connected sensing modules and the general throughput of the database on the development server. In the future, we expect to revisit and repeat the tests in greater detail on the production server.

3 Base for C++ sensing module

This C++ base for ViAn sensing module simplifies implementation of a specific ViAn sensing module. It shades a programmer of the specific module from the knowledge about the communication between module itself and the ViAn server. The programmer should define the names and data types of information for storage and then these data insert into the processing queue. The output queue processing itself and the sending data to the ViAn server runs autonomously on the background. This base also unifies the processing of input mandatory arguments, which are mandatory for each sensing module.

Dependencies

- C++ compiler (recommended version at least: GNU C++ 5.5.0)
- CMake 3.1
- pkg-config (recommended version at least: 0.29.1)
- threads
- OpenCV 3++ (recomme version at least: 3.4)
- cURLpp (http://www.curlpp.org/)

Required standard

- at least C++14

3.1 Input arguments defined by this base

Mandatory arguments

- `storage` – storage URL of ViAn server
- `access_token` – security token to access the ViAn Server by the sensing module
- `dataset` – dataset where data should be stored

Optional arguments

- `-c, --camera` – address of camera (not used right now)

3.2 List of data types, which can be inserted to processing queue (which can be processed by ViAn Server)

Logical data types

- `bool`

Integer-like data types

- `int`
- `unsigned int`
- `long`
- `size_t`

Floating point-like data types

- `float`
- `unsigned float`
- `double`
- `unsigned double`

Character-based data types

- `char`
- `char *`
- `std::string`
- `unsigned char`
- `unsigned char *`

OpenCV data types

- `cv::Point`
- `cv::Rect`

Composite data types

- `std::vector<>` of previous data types

3.3 Data definition of simple C++ demo module data definition

Firstly copy file `vian_data.prefilled_base.h` as `vian_data.h` to your project and add your own data types and names of variable to it in the manner outlined below:

vian_data.h

```
#pragma once
#include "vian_module/core/reflection.h"
#define VIAN_DATA REFLECTABLE

struct VianData {
        VIAN_DATA
        (
                (int) frame_from,

        // Here add your own fields in same way:
        // (<type>) <field name>[, (<type>) <field name>]*
        // For example:
        //      (int) foo,
        //      (std::string) bar
        )
};
```

3.4 Code of simple C++ demo module

vian_demo_module.cpp

```
#include "vian_module/vian_module.h"

class VianDemoModule: public VianModule {
        public:
                VianDemoModule(int argc, const char **argv):
    VianModule(argc, argv) {}
                virtual int run();
        // ... add your own code here ...
};

int VianDemoModule::run() {
        // ... add your own  code here ...
        // For example:
        //      // Define variable for data
        //      VianData testdata = {};
        //      // Fill in the data with values detected
        //      // by your module
        //      testdata.foo = 1234;
        //      testdata.bar = "ViAn module test";
        //      // Insert the data into the processing queue
        //      this->_data.push(testdata);
        //      return EXIT_SUCCESS;
}

int main(int argc, const char** argv)
{
        VianDemoModule vm(argc, argv);
        vm.startup();
}
```

3.5 CMake of simple C++ demo module

CMakeLists.txt

```
# IS SET IN SUBDIR > SET(CMAKE_CXX_STANDARD 14)
# IS SET IN SUBDIR > SET(THREADS_PREFER_PTHREAD_FLAG ON)

INCLUDE(${PROJECT_SOURCE_DIR}/<GIT submodule dir>/
   VianModuleBase.cmake)

INCLUDE_DIRECTORIES(${VianModuleBase_INCLUDE_DIRS})
ADD_EXECUTABLE(VianDemoModule ${VianModuleBase_SRC}
   vian_demo_module.cpp)
TARGET_LINK_LIBRARIES(VianDemoModule ${VianModuleBase_LIBS})
```

- `<GIT submodule dir>` – directory, where base module is included, i.e. as a GIT submodule

3.6 Important CMake variables

- `-DVIAN_DATA_H=<path>` – path, where is `vtapi_data.h` located in developer's project (if not defined, CMake of base module tries to find it in root directory of developer's project and also in its include subdir). *It is highly recommended to define this location explicitly.*

- `-DVIAN_DEBUG=<level>` – debug level of ViAn base module

  – 0 = no debug messages (default)

  – 1 = requests only debug messages

  - 2 = requests and responses only debug messages
  - 3 = all ViAn module debug messages (ie. start and stop of main thread and also communication thread)

- `-DVIAN_DEBUG_BLOCK=<bool>` – format of debug messages
  - 0 = inline debug messages (default)
  - 1 = block debug messages

3.7 How to compile module

```
        mkdir build
        cd build
        cmake -DviAN_DATA_H=<path of vian_data.h file> ..
        make
```

3.8 How to run compiled module

```
./VianDemoModule <storage, ie. https://10.11.12.13/>
                 <sensing module token> <dataset to store>
```

4 Database schema

The schema shown in Figure 3 allows storage of datasets from various sources of video data. It also contains tables designed for storage of information about the sources, e.g. cameras, sensing modules etc. The PostgreSQL database is used for the storage.

The ViAn database consists of two schemas. The main schema (`PUBLIC`) contains general information about datasets stored in the database, users which create them, cameras and sensing modules. Information about sensing modules includes types of sensing modules (`Sensetypes`), sensing modules (`Sensemodules`) and instances of sensing modules (`Sensemodules_instances`). The instances can be grouped in the sensing module nodes (`Sensenodes`).

For each dataset there is a new schema in the PostgreSQL database created. Its name is derived from value of the `code_name` attribute in the table Datasets. The main table of the schema for a dataset is `Configs`, which stores information about a camera and a sensing module, which produce data for this dataset. Information about the attributes and data types of this dataset and time information are stored as well. The `Sequences` table represents the video data, which can be stored together with data from the sensing modules, which is stored into `Sensedata_cars_XXX` or `Sensedata_face_XXX` tables. Their names are derived from the `data_storage` attribute values in the `Configs` table. If the version of a sensing module is changed, information about the older version's storage, attributes and data types is stored into the `Sensemodule_datastorage` table.

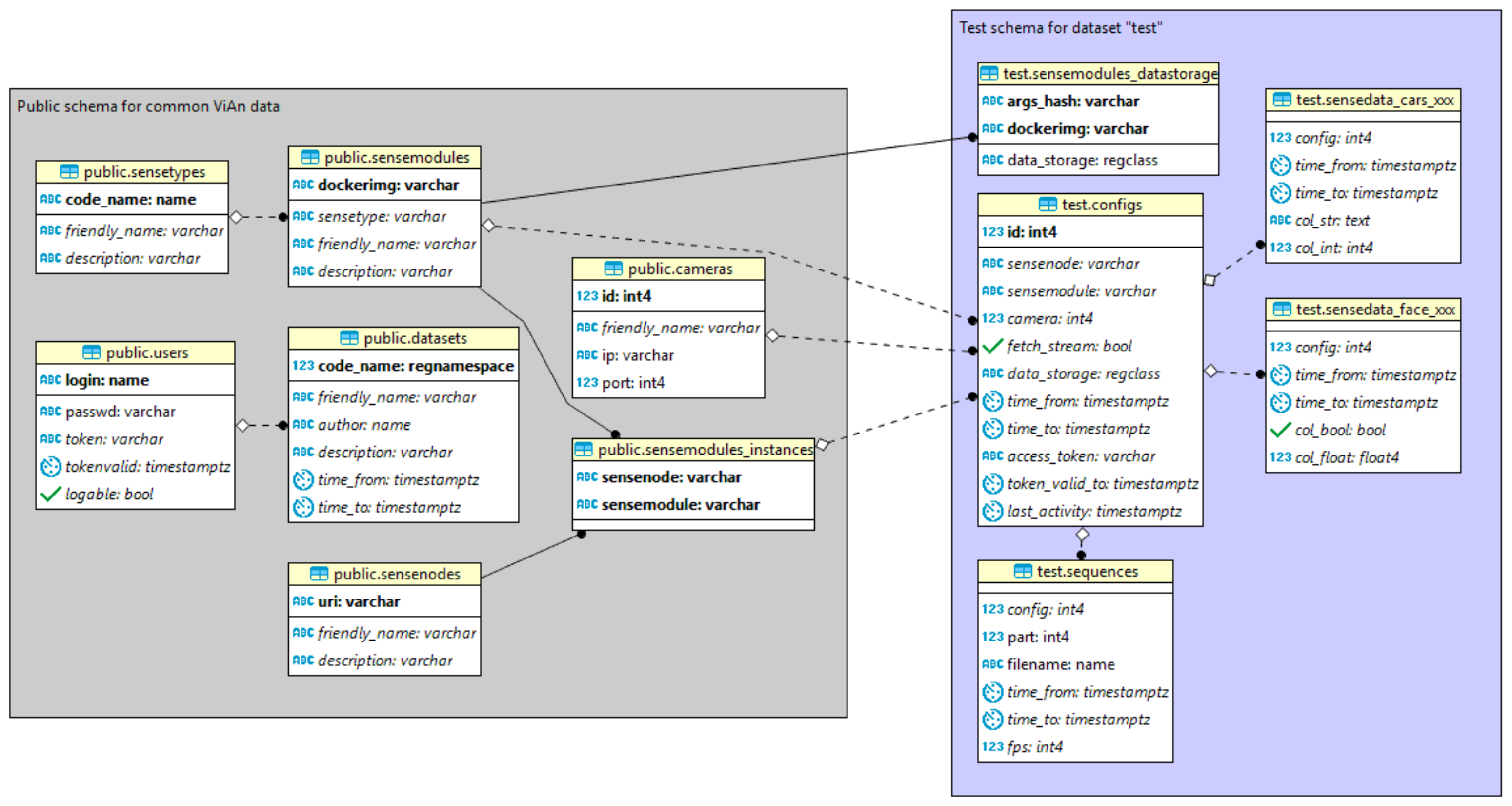

Figure 3: ViAn Database design