Rule-based Engine
Design for video surveillance applications

Tran Hoang Dai, Tran Hoang Nguyen, Choong Seon Hong
{dai.tran, nguyenth, cshong}@khu.ac.kr
Department of Computer Engineering, Kyung Hee University

Abstract

In this paper, we present a new software module that allows flexible reconfiguration of video surveillance system behaviors, which was built based on Vinotion's software engine. Vinotion has developed a set of object detection and motion tracking software libraries, which are used in many of their video surveillance systems. The result helped Vinotion's engineering team reduces time and effort to deploy new surveillance systems.

1. Introduction

Intelligent video surveillance is a challenging problem for researchers and software firms. ViNotion [1], a young company with a talented team, wants to face the challenge. With a strong focus on video and image analysis, the team has been successful in developing applications for video surveillance system, thanks to its advanced software for object detection and motion tracking. In order to extend their software framework, a new module allowing flexible configuration of surveillance system behaviors is required. This new module is called a Rule-based engine. The engine helps users to easily create and modify contextual information and custom behaviors (rules) for controlling the surveillance application. It could be seen as an interface between surveillance system and ViNotion software framework.

2. Vinotion's video surveillance engine

At ViNotion, the team had been successful in delivering several applications for video surveillance system. They developed their own software engine, which provides object-detect1)ion and tracking when processing video images. By incorporating this framework into a video surveillance system, the camera (controlled by the system) can interpret the captured scene, and let the system distinguishes different situations that are happening. Thus, the team can integrate intelligent behaviors into the system, enabling it to act smartly for a variety of scenarios. For instance, the system can send an alarm or record the video. Here are few examples of such a system.

Example: In Figure 1, a virtual fence (green area, defined by the surveillance system) is placed close to the physical fence. Whenever a person crosses that virtual fence, the surveillance system will trigger an alarm. So for this scenario, no alarm should be triggered if a bird or a rabbit (animals) would cross the fence.

Similarly, in Figure 2, a customer wants to protect his fish pond from fish-eating animals (cat, heron, etc). In the picture below, the thermal camera captures a video where a heron just landed next to the customer's fish pond area (green and blue areas, defined by the surveillance system). In this situation, the surveillance system is configured to raise an alarm when animal like a bird is detected, but not if a person approaches the pond to feed the fish.

Figure 1: Human crossing the virtual fence

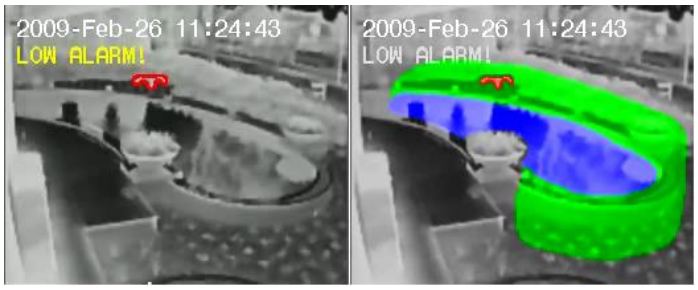

Figure 2: Fish pond protection

3. The problem

Although the team gained some successful results for current applications, in the long run, some problems still remain. Let us have a look at Figure 1. In this scenario, the team has to provide an algorithm that allows the surveillance system to only recognize humans. While in the scenario depicted in Figure 2, they have to modify the algorithm to apply detection only to fish-eater animals like a heron but not humans. This is not very convenient since for each case, they have to rewrite some pieces of the software to adapt to the requirements, and also define new areas of interest in the code. Moreover, if customers want to have more functionality from the system, the team also has to reconfigure the current working software, adding new modules along with old ones, and then deliver again. These are the problems that ViNotion would like to tackle.

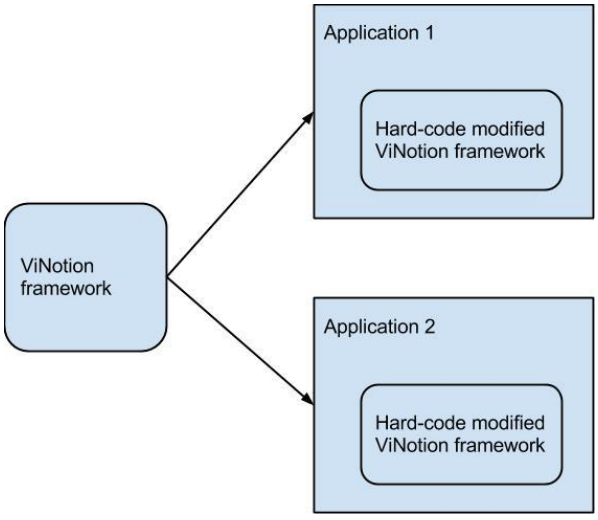

Figure 3: Deployment problem

4. Proposed solution

To avoid hard-coded solution, we build a new rule-based engine module and a software application on top that follow the scheme of Figure 4.

The rule-based engine module applies the Business rule engine model [2] with forward-chaining type [3]. Its logic behaviors are defined by a set of data represented in xml format:

- primitive types such as a point, multi-points, line, color.
- Contextual type such as an area (formed by multiple points and color); or a tripwire, which is formed by multiple lines.
- Behavioral types such as event, action and behavioral rule to connect events with actions for intelligent behavior.

To form one behavioral rule, a combination of different xml data is grouped into one large xml container element. The rule-based engine reads and analyzes the conditions that are happening on the captured video and perform actions that are defined in the xml container.

5. Application interface

To put the rule-based engine into practical usage, we built an easy to use application that allow user to define rules and test the result directly. The interface is illustrated in Figure 5.

6. Conclusion

This rule-based engine module not only helps address the deployment issues of Vinotion's framework, but also extends their video surveillance framework, make it more powerful and flexible in developing their system.

Acknowledgement

This paper was supported by NIA (National Information Society Agency). Dr. CS Hong is the corresponding author.

References

1. Vinotion, "vinotion.nl"
2. Nagl, Christoph, Florian Rosenberg, and Schahram Dustdar. "VIDRE--A Distributed Service-Oriented Business Rule Engine based on RuleML." In Enterprise Distributed Object Computing Conference, 2006. EDOC'06. 10th IEEE International, pp. 35-44. IEEE, 2006.
3. Gu, Tao, Hung Keng Pung, and Da Qing Zhang. "A service-oriented middleware for building context-aware services." Journal of Network and computer applications 28, no. 1 (2005): 1-18.

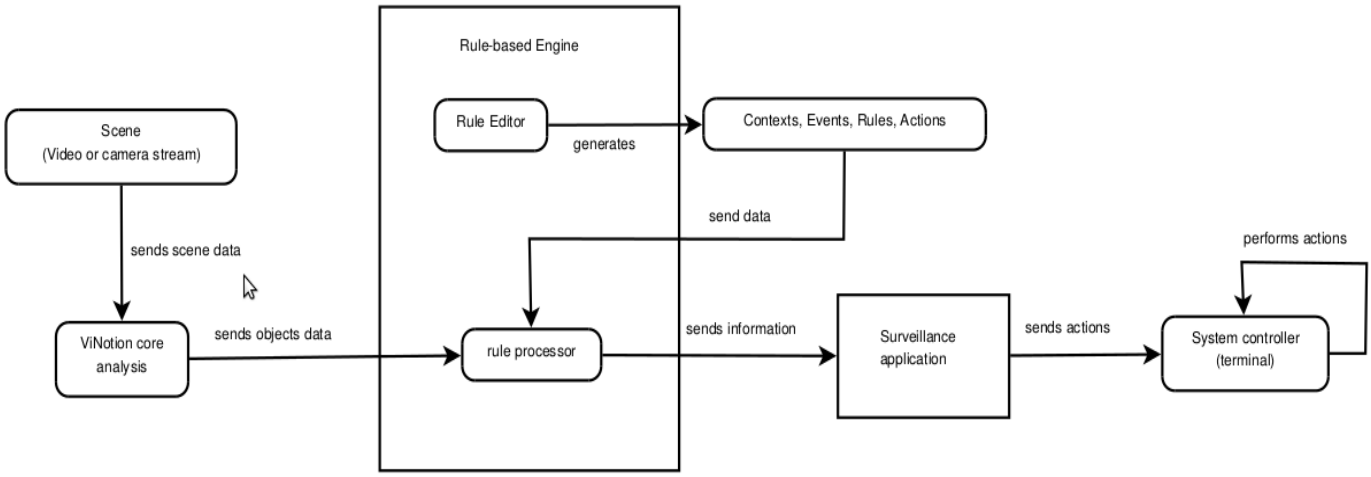

Figure 4: Proposed Rule-based model

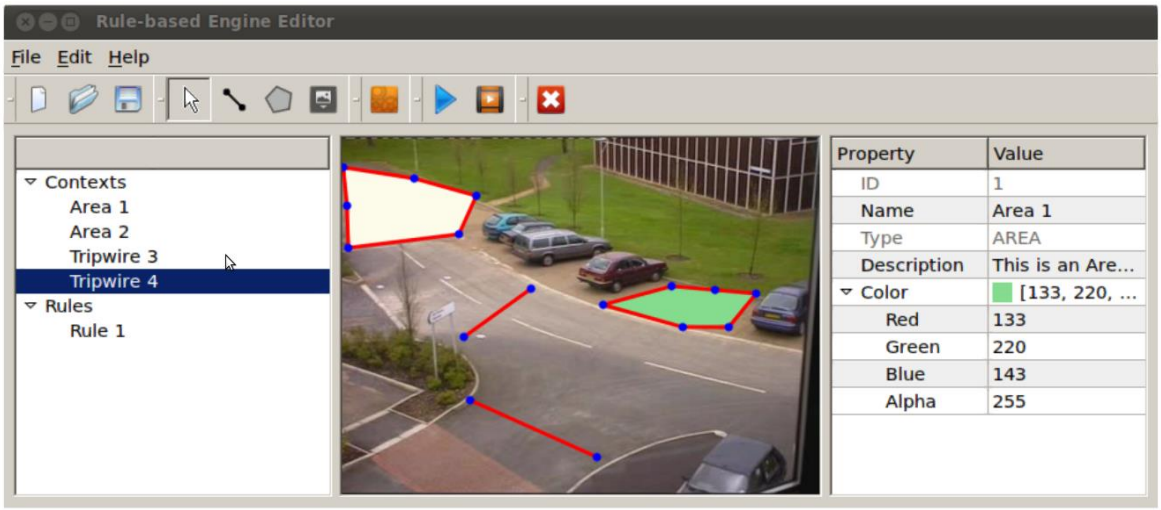

Figure 5: Application interface