An architecture for distributed video applications based on declarative networking

Xiping Wang, Cesar Gonzales, Jorge Lobo, Seraphin Calo and Dinesh Verma
IBM T. J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY, USA 10532

Abstract—Video surveillance applications are examples of complex distributed coalition tasks. Real-time capture and analysis of image sensor data is one of the most important tasks in a number of military critical decision making scenarios. In complex battlefield situations, there is a need to coordinate the operation of distributed image sensors and the analysis of their data as transmitted over a heterogeneous wireless network where bandwidth, power, and computational capabilities are constrained. There is also a need to automate decision making based on the results of the analysis of video data. Declarative Networking is a promising technology for controlling complex video surveillance applications in this sort of environment. This paper presents a flexible and extensible architecture for deploying distributed video surveillance applications using the declarative networking paradigm, which allows us to dynamically connect and manage distributed image sensors and deploy various modules for the analysis of video data to satisfy a variety of video surveillance requirements. With declarative computing, it becomes possible for us not only to express the program control structure in a declarative fashion, but also to simplify the management of distributed video surveillance applications.

Keywords—video surveillance; video analytics; declarative networking; dynamic network; policy technology; distributed computing

I. INTRODUCTION

Video surveillance applications are examples of complex distributed coalition tasks where real-time capture and analysis of image sensor data are required for critical decision making. Deploying video surveillance applications in a dynamic network environment is a challenging task in many defense and battlefield operations. Such tasks often use a diversity of vision sensors connected through a heterogeneous wireless communication infrastructure where data acquisition, communication, and analysis often take place in a distributed computing environment. In complex battlefield situations there is a need to coordinate the operation of a variety of mobile image sensors with different capabilities and power constraints: from satellite-based sensors, to UAV High Definition cameras, to personal video and infrared video cameras. For example, some of these sensors may need to be turned on only when needed so that power consumption is minimized. Further, sensor data may have to be transmitted through a wireless communication infrastructure where available bandwidth is constrained and variable. Finally, to make decisions based on the sensor data it may be required to move the data to the right places in the network where the appropriate analysis and computational capabilities exist. Simplifying and facilitating deployment, management and coordination of all of these devices and activities is the goal of the techniques described in this paper.

We have taken the first steps towards designing and implementing a flexible and extensible architecture for developing, deploying and managing complex distributed video surveillance applications, with a focus on video surveillance. Our approach is based on the use of a declarative networking paradigm to program all aspects of the problem, from controlling the operation of the sensors (their on/off state for example), to the bandwidth-aware communications between sensors and analysis nodes, to the management of the distributed video analytics necessary to help make informed decisions. In addition, the paradigm is managed using declarative rules so that the efforts of deployment and management can be significantly minimized. In this paper, we assume a fixed network infrastructure for simplicity; however, the architecture we propose can be generalized to dynamic networks as well.

In section 2 we discuss the issues we face in deploying and managing complex distributed video surveillance applications in more detail. In section 3 we introduce Declarative Networking (DN) technology and explain how this technology can be used to tackle simultaneously all dimensions of the problem using a single programmatic declarative approach. In section 4 we explain how we applied DN technology to program a simple surveillance example. Finally in section 5 we summarize our work and briefly discuss future extensions.

II. DISTRIBUTED VIDEO SURVEILLANCE APPLICATIONS

Figure 1 shows an example of the sort of distributed video surveillance application environment we envision; it is one with sensors and processing nodes overlaid in a heterogeneous, possibly dynamic network. Through this network messages and data are exchanged among the sub-applications running in various processing nodes to cooperatively control the operation of the sensors (turn on/off video cameras, select the type of video analytics, set up video rate, resolution, etc), and to route the sensor data to the appropriate processing nodes in order to
complete a coalition task. Sensor control and data processing also include tasks that require human intervention, as shown in the picture. In addition, we assume that computer resources may also be available through the Internet.

To significantly simplify and facilitate the programming and deployment of video surveillance applications in this sort of environment we need a system that can handle communications among sensing and computing resources in a heterogeneous networking environment, a system that is aware of and can control these sensor and computing resources, and just as important, a system that is easy to use and program using high level constructs that control the various interactions, communications and processing tasks. To be practical, this system must capture the core processing and communications logic of the applications while hiding the complexity of the detailed command and control instructions needed for their deployment and management in the field.

![Figure 1. An example of a surveillance application cloud.](image)

In summary, we believe that such a system should be capable of performing the following sub-tasks using simple programming constructs:

- allow the user to systematically specify system behaviors using simple rules;
- orchestrate the processing of a task with knowledge of the available hardware and software infrastructure and the existing physical and security constraints;
- coordinate the operational control of dispersed image and video sensors in response to application requirements;
- capture the processing logic steps in each processing node using simple rules that respond to the state of the system;
- automatically and dynamically route processing results of image sensor data to appropriate locations for decision making when critical events are detected;
- access securely restricted compute and database resources; and,
- manage when, where and with whom to invoke human-machine interaction.

Declarative Networking (DN) is a promising technology which allows us to simplify the programming and management of complex tasks such as the ones described above. This technology is particularly efficient in describing the control of sensors and the overlay asynchronous distributed processing and communications which is driven by state machines operating in each of the nodes of the network.

### III. A DECLARATIVE NETWORKING APPROACH

Declarative languages are used in computer programming to focus on compactly expressing the logic of computations rather than the details of the control flow of those computations, i.e., on what needs to be done rather than how it should be done. By analogy, Declarative Networking (DN) languages have been developed recently to compactly specify networking protocols. Using NDlog, a DN language derived from the database query language SQL, Loo et al. [1] have programmed popular routing protocols in a few tens of lines of code, as compared to literally thousands of lines needed using more traditional procedural languages like Java or C. The DN paradigm specifies network behavior that corresponds to a network protocol by specifying the operations that need to be performed at each network link and node, and the messages that are exchanged among them to guide what are fundamentally asynchronous operations. DN has been shown in this way to be capable of effectively and succinctly representing a variety of networking protocols.

By extending the DN model with the concept of time, the use of DN languages can be expanded to become a very powerful and general programming model for asynchronous distributed computations [2]. Thus, DN can also be used as a unifying approach for the development and deployment of a wide range of distributed applications by overlaying application logic, security constructs, and network management and control [3]. Recent applications of DN include specification of distributed applications not unlike the video surveillance applications we have in mind in this paper [2].

In summary, DN can provide an extensible and flexible infrastructure to efficiently express the behavior of systems as a whole, incorporating both network and application layers simultaneously.

#### A. An architecture for declarative computing

A distributed application is comprised of elementary applications executing in multiple processing nodes of a network, i.e., the application nodes. An application node can interact with other application nodes by exchanging messages, and each application node manages the transitions of an internal state machine programmed to respond to external triggers generated by sensors or other application node messages. In turn, the state machine transitions in an application node generate control and data messages for sensors and other application nodes. The distributed application can then be thought of as managing the operation of a distributed (or meta) state machine (DSM). Figure 2 is a schematic of a distributed application where, for clarity, we have excluded the sensor nodes.

We are developing a platform to support programming and deployment of video surveillance applications using the DN paradigm. This means that programming a distributed
application, including the interactions among application processing nodes and sensors, will be done using a high-level DN language. The idea is to have a platform that supports the execution of DN programs in each application node, and thus a platform that includes a runtime environment as well as the required communications and security middleware. This platform can be quite general, but because we focus here on image and video surveillance applications, we are specializing it to include runtime libraries that support popular image and video capture and compression modules, as well as standard APIs to pluggable image and vision analysis libraries. While using a DN language holds the promise of simplifying the programming task, compiling and running a distributed DN-based application is quite a complex task. Our goal, however, is to provide a platform that spares the application programmer from dealing with most of this complexity.

The specific logic executed at each individual application node may obviously vary from node to node. For example, in a node controlling a video camera, the primary function may be very simple, e.g., to control the on/off operation of the camera, the video image resolution, the video compression data rate, etc... Similarly, in a node containing the computing resources to perform vision analysis, the function of the application may be to respond to external processing alerts and then execute the appropriate vision analysis function based on the state of the system, the type of alert, and the availability of computing resources. Because we use high-level DN programming constructs that run in a common runtime environment which is independent of network topology and communication protocols, we hope that programming, deploying, running and modifying distributed video surveillance applications will be greatly facilitated.

Figure 3 is a pictorial representation of the platform we propose to deploy in each application node. The platform consists of the following four elements:

- **Application Container**: a piece of software running at each node to provide a runtime environment for node applications. It receives node applications from the DN application manager and executes them.
- **State Machine (SM)**: a representation of rules and state transition models which determine the application processing logic. Each SM is initialized with an initial state. The state consists logically of a set of variables contained within a state machine. State transitions happen when external triggers from sensors and other application nodes are processed by the application. A trigger is an event or data message.
- **Node Application**: an elementary application of a distributed application that runs on a node to complete a sub-task. A coalition task is completed by coordinating all node applications as a whole.
- **Node System**: a piece of software that provides communication and other services binding the application to the platform on which it is running. It also maintains a logical topology imposed by a distributed application.

Using this platform, a distributed application is implemented as a collection of node applications, each with its own SM, running in a network of computational nodes (e.g., sensor cameras, back-end servers, mobile devices) exchanging messages between them throughout the network as illustrated in Figure 4.

Figure 4 shows another component of the architecture we are developing: the application manager. It is used to distribute
and update the DN-based applications and SM rules running in the various application nodes. While a distributed application assumes a logical topology to specify the routing of messages among different application nodes and sensor instances, the application manager may provide the services that determine the physical routing. Alternatively, it is the application manager that may distribute the DN-based network protocol specifications needed for routing.

B. A DN programming language

The state transition function in each SM is defined by a collection of declarative rules, and states are represented by a collection of tuples, i.e., an expression of the form $p(a_1,\ldots,a_n)$, where $p$ is the name of the tuple and the $a_i$ the arguments of the tuple, are constant values like integers and strings of characters. The output messages corresponding to an SM transition can also be expressed as tuples that specify a destination where the tuple should be delivered. A transition function thus defines the tuples that should be added to or deleted from the set of tuples defining the current state to generate the set of tuples in the resulting state, as well as the tuples to be sent out to other SMs.

In this section we will present an informal description of our language using simple examples. A full description of the language will be described elsewhere [4]. First, one needs to declare the schema of the tuples that are going to be used in a program. The schema describes the name of the tuple, the number and types of arguments, e.g., integer, string, etc., and the type of the tuple itself: input, transport (i.e., representing a message), or state. For example, we can have the following schema:

**input**

- new_im_processor(string,string,string);

**transport**

- req_process(string,string);
- process(string,string,string);

**state**

- img_processor(string,string,string,string);

There are three types of rules in our DN programming language. The first type of rule is one to add new tuples to the state. An example of this type of tuple is:

```
```

This rule takes an input tuple indicating that a new image processing module $P$ with capabilities $C$ has been installed in server $S$ and adds a tuple to the state indicating that the module is currently inactive (assuming that the module can be activated only once). The second type of rule is one to remove tuples from states. For example, if there is a request $R$ for processing an image $I$ by a particular image processing module “mod1”, then we need to check that the module is inactive, and make it active. This is done using two rules:

```
img_processor(P,C,S,"inactive") -= req_process(I,P)@R,
img_processor(P,C,S,"inactive") += req_process(I,P)@R,
```

The first rule removes the inactive tuple from the current state, and the second adds the corresponding active tuple. In these two rules the tuple $req_process(I,P)$ is a message tuple sent from $R$. Here, $P$, $C$, $S$ and $R$ are variables and when the tuple arrives from $R$ the value of $P$ is “mod1”. The last type of rule is a rule that generates messages. In our example we will like to send the image $I$ to the appropriate server. This is described by the rule:

```
process(I,P,R)@S += req_process(I,P)@R,
img_processor(P,C,S,"inactive")
```

In this rule $S$ represents the server to which the tuple $process(I,P,R)$ should be sent. We pass $R$ in the tuple so that $S$ can directly send the results of the processing to $R$. In addition to variables, one can use expressions that operate over variables and constants. For example, if the type of two variables, $X$ and $Y$ appearing on the right hand side of a rule are integers, then the expression $X + Y$ can be one of the arguments of the tuple in the left hand side. If the variables are strings they can be concatenated. Aggregation operations such as sum or average of values can also be done. For example, the following rule counts how many processors are active:

```
active_processors(count(P)) += mg_processor(P,C,S,"active")
```

And the following rules group the count per server:

```
active_processor_per_server(S,count(P)) +=
    img_process(P,C,S,"active")
```

Since one can use any tuple in the left hand side of a rule, rules can depend on other rules, so that one can construct complex combinations and conditions. For example, if one wants to limit the number of active processors to, let’s say 20, the rules of activation above can be replaced with:

```
img_processor(P,C,S,"inactive") -= req_process(I,P)@R,
img_processor(P,C,S,"inactive") += active_processors(X), X > 21
```

We did not defined schemas for active_processors and active_processors_per_server but they need to be defined. These are temporary tuples that are not part of the state but are just used to hold temporary values during computation and are called transient. Their declaration will be:

**transient**

```
active_processors(int);
active_processors_per_server(string,int);
```

C. An extended platform for distributed surveillance applications

By including video specific middleware in the common platform described in Section 3.1, we can specialize the platform to support distributed video surveillance applications. Figure 5 shows the diagram of a DN application in more detail. Note that while we focus here on image and video sensors, it is
easy to see that we can easily extend the idea to support other media types such as sound. In our implementation of state machines, a rule engine manages the state machine transitions based on rules (distributed by the application manager). A state transition will be driven by the values of the current state, the values of external messages (which are tuples generated by other external application nodes), and metadata extracted from sensor input data using the video middleware software. One of the functions of the video middleware is to process the sensor video streams to extract metadata that is used by the rules engine. At the same time, the type of video processing itself may be controlled by the rules engine.

IV. EXPERIMENTAL SCENARIO

To demonstrate the concepts and to help drive our work we have implemented a simple surveillance experiment with a first version of our platform. The experimental setup, which is shown in figure 7, consists of two surveillance cameras, two application nodes, and two backend analysis computers.

Figure 5. Block diagram of application node.

Figure 6 shows a block diagram of the video middleware shown in Figure 5. Besides providing a number of standard services such as video compression and decompression, network and sensor interfaces, and network monitoring and transport, the middleware has an extensible architecture with a pluggable vision analytics API. The goal is for this API to provide a standard way of interfacing externally provided image and video analysis functions as required by the specifics of the surveillance application. A few examples of these functions are shown in the figure.

Figure 6. Video middleware.

Using this middleware we can, for example, modulate the bandwidth of video streams to meet network traffic and QoE constraints, dynamically control image sensors, apply encryption to video streams when required, coordinate and analyze multiple video streams simultaneously based on DN rules that define what events to look for in the analysis, and determine what to do when events are detected and how to communicate the results to neighboring network nodes.

In our scenario we assume the following operational components and constraints:

- Video camera 1 is “on” all the time and is connected to application node 1 (AN1).
- Video camera 2 has limited battery power and therefore the time spent in its “on” state needs to be minimized. It is connected to and controlled by application node 2 (AN2).
- AN1 and AN2 have limited video analysis capabilities and can only be used to detect object intrusions in their attached camera’s normal field of view, i.e., frame differences against a “normal” frame.
- Backend analysis 1 and 2 are powerful remote computers with greater vision analytics capabilities.

The sequence of events in the scenario we have implemented runs as follows:

1. Camera 1 captures the view of a driveway in front of a secured building and transmits compressed video to AN1.
2. AN1 decompresses and processes the video searching for an intrusion in camera 1’s field of view. When an object (a car) enters the field of view, AN1 sends a message to backend analysis 1 to start analysis of the video. Simultaneously, the compressed video from camera 1 is passed on (possibly cropped and transcoded) to backend analysis 1 for further processing.
3. Backend analysis 1 decompresses and processes video until it detects that a passenger has been
dropped off in front of the building; it then sends an alert to AN1.

4. AN1 sends an alert to AN2 which is connected to camera 2, which is positioned to monitor the entrance lobby of the building.

5. AN2 sends a command to camera 2 to start capturing and sending compressed video.

6. AN2 decompresses and processes video searching for a person (new object) in its field of view. When AN2 detects a person entering the lobby it sends message to backend analysis 2 to start processing. Simultaneously, the compressed video from camera 2 is passed on (possibly cropped and transcoded) to backend analysis 2.

7. Backend analysis 2 identifies the intruder and sends a message back to AN2 to turn off camera 2 as a power savings measure.

As described above, the sample video surveillance application is divided into two node applications each of which is managed by a state machine and the goal of object tracking is completed by coordinating the two node applications. Figure 8 illustrates the state diagrams corresponding to the operation of AN1 and AN2. A circle in Figure 8 represents a state and an edge denotes a transition. An event triggers the rule engine to evaluate conditions and forces the state machine to transition from one state to another state if the conditions hold true. The states and state transitions in Figure 8 can be captured using declarative networking code or rules. Figure 9, as an example, shows the DN rules that represent the state diagram for AN1 using the DN language in its current state of development.

V. CONCLUSIONS AND FUTURE WORK

Video surveillance applications are examples of complex decision making applications where the operation of multimedia sensors and the management of data analysis tasks need to be coordinated over distributed processing nodes. For example, the intermediate analysis results of video data may need to be dynamically routed to appropriate locations for further analysis based on resource and security constraints. Developing, deploying, and modifying this type of application is a very challenging job. To tackle this job, we have proposed an architecture based on a declarative networking paradigm. Such a paradigm makes it easier to program distributed applications, integrating in a single framework various interdependent operational aspects of distributed systems such as networking, security, and management and control of sensors and analysis processes. With this architecture, we hope to greatly facilitate and automate the process of developing and deploying distributed video surveillance applications, i.e., the work that expert systems analysts must perform to translate application requirements and user goals into a set of declarative rules which can be easily deployed and executed in the network.

We have implemented a simple example of a video surveillance application using the proposed declarative computing paradigm. While the work described here is still in progress, this example illustrates the potential of the declarative networking paradigm. Future directions for the evolution of the platform include expanding the work to support mobile ad-hoc networks which will require incorporating network protocol definitions within the same declarative networking framework.

REFERENCES


