ViFramework: A framework for networked video streaming components

B. Kersten[1], **K. van Rens**[2], **and R. Mak**[1]
[1]Security and Embedded Networked Systems, Eindhoven University of Technology, The Netherlands
[2]ViNotion B.V. The Netherlands

Abstract— *Real-time video content analysis applications for surveillance become more and more demanding. The need for load distribution, remote management and reusability calls for a component framework specialized in networked video streaming applications. Whereas lots of component frameworks exist nowadays, frameworks targeted at networked video streaming are scarce. Added requirements imposed by video surveillance applications include real-time computing and quick failover. The framework proposed in this paper meets these demands by enabling the distributed execution of video streaming applications in an efficient and resource-aware fashion. In this paper, we present the design of the proposed framework and evaluate a prototype implementation. The results of this evaluation show that this implementation is efficient and can successfully perform failover handling, making it suitable for distributing surveillance applications.*

Keywords: Software component framework, video streaming, video surveillance applications, distributed video analysis

1. Introduction

Developing applications using the Component-Based Software Engineering (CBSE) paradigm [1] has many advantages such as high reusability, low time to market and decreased development costs. Many software component frameworks exist nowadays, but frameworks supporting video streaming are rare, especially when network functionality is required.

Applications that need streaming video are Video Content Analysis applications, such as the ones studied in the recent ITEA2 research projects CANTATA [2] and ViCoMo [3]. These applications are becoming increasingly more demanding. Applications that process video streams originating from multiple cameras with computationally-intensive algorithms like object detection and tracking are becoming more common. Due to the high-volume nature of video data, processing components often have high resource demands. The need for distributed applications is motivated by the need for geographical distribution and load distribution in order to make the applications more scalable.

A framework for networked video streaming components is needed, in order to enable component distribution over hosts connected by a network. The framework must enable components to be configured and composed remotely in order to form an application. By supporting *dynamic reconfiguration* the framework must allow for run-time modifications of the application's component graph.

At design-time, the framework must provide component developers with abstraction of tasks, such as setting up network connections, video compression, timing and multithreading. To make the framework suitable for rapid prototyping, the framework must be flexible and component descriptions easy to adapt. At run-time, the framework should provide network-transparent means to compose an application. In order to allow applications to span both LAN's and WAN's, the framework must support NAT-router and firewall traversal.

Targeting at video surveillance applications, the framework is subject to real-time requirements. In general, a trade-off must be made between timeliness and guaranteed delivery. By adjusting QoS parameters the framework must be able to meet the real-time requirements of the applications. In host-failure situations the framework must be able to perform quick failover, thus increasing robustness and minimizing the amount of lost data.

This paper proposes a framework for networked video streaming components aimed at surveillance applications. An implementation of the proposed framework is presented in [4] and is evaluated in this paper. Evaluation is done by porting an existing surveillance application to the framework after which overhead and failover time is measured.

In Section 2 the general architecture of the proposed framework is presented. An application scenario is sketched in Section 3. Section 4 elaborates on framework details. Framework evaluation is presented in Section 5. Section 6 describes related work and Section 7 concludes the paper.

2. Framework Architecture

The proposed framework exists of a design-time and a run-time part. The design-time part of the proposed framework consists of means that help the component programmer to create components that comply with the framework. This includes an interface definition language, automatic code generation and programming guidelines.

Before existing video content analysis algorithms can be used as components in the framework, they are supple-

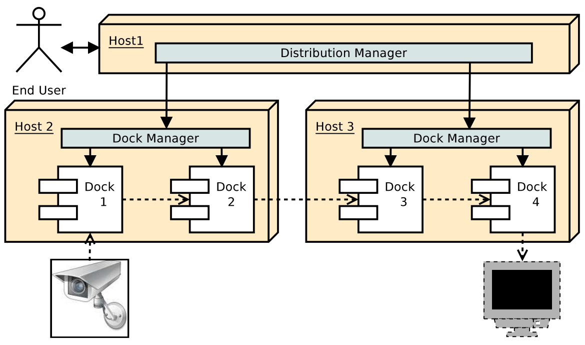

Fig. 1: High-level ViFramework architecture. A video content analysis application consisting of four components spanning two hosts is set up. Each dock manager manages all docks on the host it resides on and one distribution manager controls the whole application.

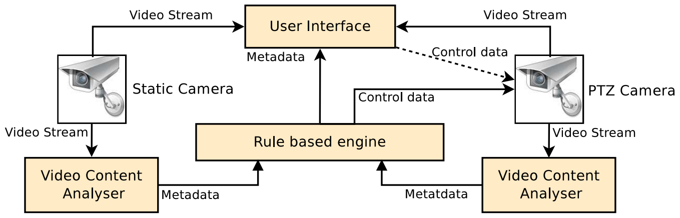

Fig. 2: Data flow diagram of the object detection and tracking application. The control-data flow from the user interface to the PTZ-camera is optional because the end-user may chose to manually control the PTZ-camera or not.

mented with platform-specific software that provides additional functionality for network usage. This is called the instrumentation procedure and results in a so-called *dock*. The term dock is borrowed from the SOFA framework [5] and denotes a container for multiple components that together provide functionality to the environment. Docks include a control part which controls the included components, handles configuration and facilitates network usage, and a functional part which is the component code.

The run-time part of the framework consists of two active entities that enable the distribution of a video streaming application. Each host that takes part in the framework runs one process that manages all docks on that host. On each host, this *dock manager* is the only process that can instantiate docks. After instantiation the dock manager can configure, start, stop and destroy a dock. Configuration includes binding of the component's interfaces in order to connect them to other components.

All dock managers are connected to a central service named the *distribution manager* that is used to gather information about available hosts from their dock managers. Since the distribution manager has control over all available dock managers, it is capable of composing a networked video streaming application, instantiating and connecting available components. A user-interface to the distribution manager enables end-users of the framework to manually setup an application, although the distribution manager can also be configured to automatically setup and manage predefined applications. An overview of the high-level framework architecture is depicted in Fig. 1.

When considering video streaming applications as done in [6], three component types can be distinguished:

- *Input:* Components that capture video data from an input source (e.g. a camera, a file or an Internet stream), convert it to a common internal format, after which it can be offered to an interface.
- *Output:* Components that accept the common internal format from an interface and convert it to an output format which can, for example, be a display or a video file.
- *Processing:* Components that can be used to read video data in the common internal format from an incoming interface, process the video stream before forwarding it to an outgoing interface in the same format.

Typically, processing components can reside on any host, whereas in- and output-components need additional hardware in order to fulfill their task and are therefore located in the proximity of these devices (i.e. a camera or video display). If not composed manually, it is the distribution manager's responsibility to setup a pre-defined application taking into account what resources are available on the connected hosts.

The distribution manager is capable of performing failover by re-instantiating failed docks on other hosts and re-routing the data through the re-instantiated docks. In the same way, the distribution manager is capable of performing load distribution. Because video streaming uses a lot of network bandwidth the framework takes network capacity into account when setting up and managing applications. It does so by adjusting stream routes and choosing appropriate Quality of Service (QoS) levels and compression techniques. By using NAT-router and firewall traversal it is possible to deploy applications that cross the borders of a LAN.

3. Application Scenario

As a proof of concept the proposed framework implementation is used to distribute an existing surveillance application over multiple hosts. The application chosen for this is an object-tracking application using a static and a Pan-Tilt-Zoom (PTZ)-camera as depicted in Fig. 2. The video stream from the static camera is used for object detection and tracking. When an object is detected, the PTZ-camera is used to zoom in on the target and to extract more object-specific information. The video stream from the PTZ-camera could, for example, be used for face recognition on the zoomed-in object. The PTZ-camera is controlled automatically using the coordinate information from the “Video Content Analyser” component which analyses the video feed from the static camera. Moreover, the end-user can at any time connect to

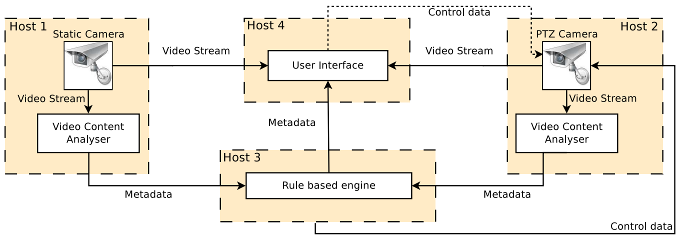

Fig. 3: A possible distribution scenario of the object detection and tracking application spanning four hosts.

the user interface component and watch the incoming video streams and application-generated metadata. Optionally, the user can also take manual control of the PTZ-camera. This application can be divided in up to four docks:

- *Static-camera analysis*: Object detection and tracking algorithms generating PTZ-coordinates based on the video stream provided by the static camera. Metadata describing the objects and their locations is send to a rule-based engine.
- *PTZ-camera analysis*: Controlling the PTZ camera based on incoming PTZ-coordinates and using the video stream from the PTZ-camera as input for a video content analysis algorithm.
- *User interface*: Presentation / interaction component.
- *Rule-based engine*: Gathering metadata from both analysis components and informing the end-user on events by forwarding them to the user interface.

A possible distribution of these docks is depicted in Fig. 3.

4. Framework Details

4.1 Location transparency

The ViFramework provides generic means that allow the end-users to deploy and connect docks on available host irrespective of the underlying network topology. In order to create this *location transparency* the framework is built on top of the XMPP protocol [7] originally designed for messaging purposes. Because of its modularity and ease of extensibility it has become a communication protocol used by all kind of applications such as a the Peer-to-Peer desktop grid computing substrate [8]. XMPP has very attractive features for this framework such as presence information of clients, possibility of NAT-router and firewall traversal and extensive security measures like TLS ans SASL. Because of its modularity, a light-weight framework can be created by including only the XMPP modules that are necessary. All this makes the XMPP protocol an excellent network substrate for an easy to extend component-framework that satisfies the needs of demanding video content analysis applications.

A major drawback of the XMPP protocol when used for the ViFramework is the lack of efficient video streaming support. XMPP does have an extension that facilitates stream initiation (XMPP extension number XEP-0095) that is used for our own video streaming algorithms.

The main types of data that are communicated between components in a networked video streaming application are video-data, metadata and control-data. The framework supports these data types. The metadata and control data are assumed to be event-based and are communicated using the XMPP protocol itself. This protocol is XML-based and can therefore be used to send any data type that can be represented by structured text. Streaming video is done outside the protocol. The ability to communicate these data types is sufficient for the application example in Fig. 3. In general, these data types are sufficient for almost any network video streaming application.

For video streaming three types of communication are used dependent on the relative location of the components that are connected to each other:

- *Local:* For docks that are instantiated on the same host, shared memory is used for communication. The dock manager manages this shared memory.
- *RTP:* For connections between docks that reside within the same LAN the RTP protocol is used. Using RTP upon UDP makes it makes possible to meet real-time requirements because the protocol will not wait for lost packages.
- *SOCKS 5:* For connection between docks that reside on distinct LAN's (and therefore needs to traverse a NAT-router or firewall) no RTP connection can be setup because this protocol is IP-address based and hosts behind a NAT-router do not have an unique IP-address. Furthermore, firewalls could block the ports used by the protocol. A SOCKS 5 [9] proxy is used to setup a SOCKS 5 byte-stream between the two components. Such a byte-stream is based on a TCP connection and is therefore not very suitable for applications subject to QoS.

Typically, the real-time part of video content analysis applications resides on a LAN, whereas WAN connections are, due to their higher delays, mostly used for monitoring, control and notifications. For the latter tasks guaranteed delivery is more important, which makes a SOCKS 5 byte-stream a suitable candidate for inter-LAN connections.

For metadata communication, the XMPP protocol is used, except for intra-host communication, for which we use method invocation. The message passing XMPP protocol needs an XMPP server to relay messages between hosts. End-to-End connections can be used for intra-LAN communication of metadata but this requires an extension of the XMPP protocol (XEP-0246). Because the vast majority of the data communicated within a typical video content analysis application is video data, there is little to gain and therefore, this extension is not implemented.

Fig. 4 depicts a possible network structure supported by the framework. End-to-End RTP sessions are used for intra-

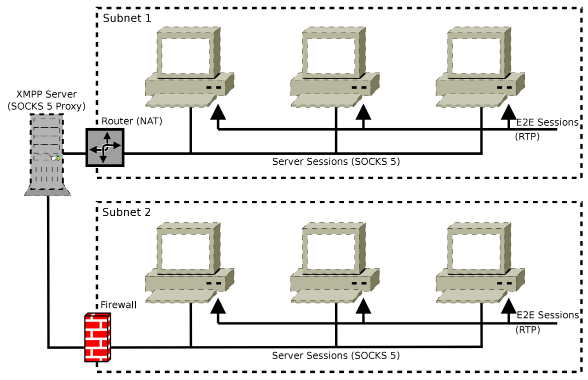
Subnet 1
XMPP Server (SOCKS 5 Proxy)
Router (NAT)
E2E Sessions (RTP)
Server Sessions (SOCKS 5)
Subnet 2
Firewall
E2E Sessions (RTP)
Server Sessions (SOCKS 5)

Fig. 4: Possible network structure supported by the ViFramework. End-to-End RTP sessions are used for intra-LAN video streaming. For inter-LAN video streaming the XMPP server is used as SOCKS 5 proxy. Only the XMPP server requires a public IP-address. Metadata communication is not depicted in this figure.

LAN video streaming whereas the XMPP server can be used as a SOCKS 5 proxy in order to setup a SOCKS 5 byte-stream between two hosts in different subnets. The XMPP Server needs to be accessible from both subnets so a public IP-address is required.

4.2 Interface definition

The typical pipe and filter architecture pattern [10] found in video streaming applications consists of an in- and output component with one or more intermediate processing components. The need for dynamic reconfiguration calls for a data-centric composition technique. The proposed framework allows dock builders to specify what data types the dock requires and provides. When deploying an application the required docks are instantiated. An interface-matching algorithm is used to calculate, given an interface, which interfaces can be connected to it. The result of this algorithm can be used to automatically set up an application or can aid the user in manually setting up the application.

The demand for flexible dock definitions and the use of the XMPP protocol makes XML an appropriate language for dock and interface definitions and it is therefore used as the *Interface Definition Language* (IDL) in the proposed framework. At design-time, a configuration XML file is designed for each host. At start-up, this file is read by the dock manager which parses, amongst others, its identifier, the XMPP server address and the available dock definitions from this file. A dock definition contains a dock identifier, a functionality description and a list of interfaces with their respective QoS properties. For each dock instantiated by the dock manager, a copy of the dock definition is made, which can be modified by the dock manager. Changes to these description instances can be made to, for example, bind interfaces by adding target information to the interface element or to set QoS properties.

Each video streaming interface can set a topic, which

```
<providedInterfaces>
 <interface>
  <videoStream topic='PTZ'>
   <qos>
    <fps>
     <min>5</min>
     <max>15</max>
    </fps>
   </qos>
  </videoStream>
 </interface>
</providedInterfaces>
```

```
<requiredInterfaces>
 <interface>
  <videoStream topic='ANY'>
   <qos>
    <fps>
     <min>10</min>
     <max>25</max>
    </fps>
   </qos>
  </videoStream>
 </interface>
</requiredInterfaces>
```

Fig. 5: Interface definition of two matching video streaming interfaces. The provided interface can operate in the range specified by the required interface. The required interface will accept streams with any topic.

can be used for stream identification and a number of QoS parameters. Two examples of interface definitions are listed in Fig.5. The matching algorithm checks whether the provided interface can meet all demands of the required interface which is the case in this figure. Metadata interfaces are defined by the XML representation of the object they communicate. The dock builder is able to construct any data type for communication as long as it is representable in XML. When, for example, the dock builder needs information about detected cars to be communicated, interface definitions as depicted in Fig. 6 can be specified. When a dock with a provided interface having this specification sends data, it fills the `<carInfo>` element with data and sends it to the required interface. This representation allows easy creation of new object types and easy extension of existing ones.

At run-time a dock can be easily replaced by an other dock with compatible interfaces but with potentially different functionality. The user receives an overview from the framework on what connections can be made between instantiated docks. With the proposed framework, creating more complex component graphs is quite straightforward as provided interfaces are able to setup connections to multiple required interfaces. When streaming video, each of these connections can have its own QoS properties. Moreover, this allows run-time extension of existing applications by adding additional processing steps, or by branching the video stream at a certain point, in order to create a separate processing path.

4.3 Host failure recovery

The use of the XMPP protocol as a network substrate provides the proposed framework with information about the presence of dock managers. The XMPP server will notify the distribution manager when a host has gone off-line. The distribution manager will react on such an event by starting a recovery algorithm. This algorithm tries to re-instantiate the docks that were running on the failing host, on other (possibly unused) hosts, tries to reconnect them and upon success, restarts the failed part of the application. An

```
<providedInterfaces>
 <interface>
  <metaData>
   <carInfo>
    <location/>
    <speed/>
    <direction/>
   </carInfo>
  </metaData>
 </interface>
</providedInterfaces>
```

```
<requiredInterfaces>
 <interface>
  <metaData>
   <carInfo>
    <location/>
    <speed/>
    <licensePlate/>
   </carInfo>
  </metaData>
 </interface>
</requiredInterfaces>
```

Fig. 6: Interface definition of two non-matching metadata interfaces. The addition of the `<direction/>` element is allowed because a provided interface might supply more data then needed. Adding the `<licensePlate/>` element in the required interface will make this interface no longer matching because the provided interface can not provide this element.

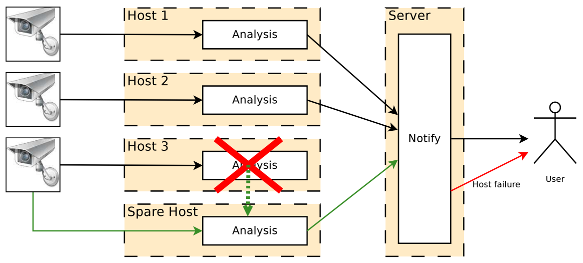

Fig. 7: Crash recovery example. Three camera feeds are processed, each by a dedicated host. All data gathered by the analysis algorithms are forwarded to the server which will notify the end-user on certain events. If *Host 3* fails, the framework will try to find a host to re-instantiate the lost analysis dock on and reroute the video stream that was processed by the crashed host through the new one. The end-user will be notified about this host-failure.

example situation is depicted in Fig. 7.

The framework is designed for real-time systems and therefore no attempt will be made to resend frames that are lost due to host-failure. Because the framework is targeted at surveillance applications, crash recovery should be performed in the least amount of time possible in order accomplish minimal data-loss.

4.4 Resource Management

To enable automatic deployment of new applications, dynamic reconfiguration of existing application and host failure recovery the distribution manager needs information about the available resources (e.g. CPU, memory, network bandwidth) on each connected host. To enable load balancing, also information about the current resource usage is required from each connected host. Resource information is gathered by the dock managers and forwarded to the distribution manager. This enables resource management on two levels; at host-level and at system-level.

Ideally, the dock manager process is the only process running on each host apart from mandatory OS processes. Because of the low resource usage of OS processes it can be assumed the dock manager has all the host's resources at its disposal. As future work, this could be forced by running the dock manager in a virtual machine. The dock manager will spawn a new thread for each dock it instantiates. At this point resource reservations can be made for this new dock. Docks are allowed to spawn new threads themselves. In the current implementation, host-level resource management is left to the operating system.

At system-level, the distribution manager has knowledge about the available and used resources of each host. Therefore, it can make educated decisions when deploying new docks. For example, when the new analysis dock is instantiated in the host failure recovery situation of Fig. 7, the distribution manager will opt for the unused host, rather than a host that is already doing heavy computation.

Resource requirements for video content analysis algorithms are often data-dependent [11]. This requires the framework to respond to a sudden increase in resource requirements. If a host cannot meet the resource requirements of its docks, the distribution manager needs to redistribute the application in a more appropriate way.

So, to make its global deployment decisions, we see that the distribution manager needs resource information that is as accurate and recent as possible. As stated in [12], large applications that constantly send resource information to a central service create an extensive network usage overhead. This makes an implementation in which the dock managers send resource usage updates to the distribution manager at a high fixed rate not well scalable and therefore unsuitable for a video-streaming framework. In [12] a solution for this problem is proposed. This solution divides resource usage in three usage-levels and only sends on level transitions. The ViFramework uses a similar, but more extensive solution to solve this problem.

For each resource (e.g. CPU usage) a new value will only be reported if it exceeds a user-defined threshold with respect to the last reported value and only when this situation persists for a user-definable duration. Fig. 8 shows an example resource graph. This solutions enables a trade-off to be made by the end-user of the framework between network bandwidth usage and information granularity. It is an improvement over [12] because it provides the end-user with more detail when needed and it prevents large data bursts when resource usage oscillates between two usage-levels.

5. Framework Evaluation

In order to evaluate the proposed framework, the computational overhead and the time needed for host failure recovery were measured. Because data compression and streaming, although configurable by the framework are not dependent on the framework, no network usage measurements for applications deployed on multiple hosts are carried out. Furthermore, after application initialization, the only framework related network traffic is resource usage information, which is negligible.

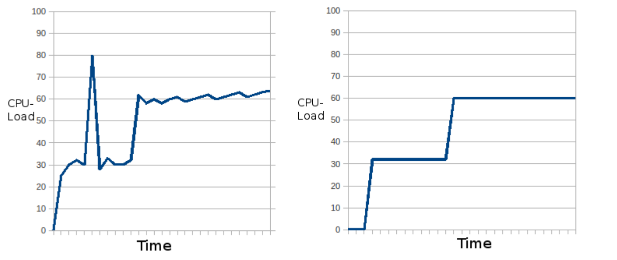

Fig. 8: Left: Host CPU usage registered by the dock manager | Right: Resource graph received from the dock manager at the distribution manager.

Table 1: Overhead Measurements

Cpu usage	**Min**	**Avg**	**Max**
Standalone1	58%	60%	62%
Frameworked1	65%	67%	70%
Standalone2	84%	87%	89%
Frameworked2	95%	97%	99%

5.1 Evaluation method

An application was created that reads a video from file, applies *object detection* on the video and writes the resulting video to a display. The video used has a resolution of 640x480 at 15 frames per second. For framework evaluation the application is divided into three docks (read, process, write) which are all deployed on the same host. The frameworked application and the standalone application were both executed on the same host.

The test host contains a quadcore Intel® Core™ i7 870 processor at 2.93 Ghz with 2 GB of RAM. The operating system used is Linux 2.6.36-26.

5.2 Overhead

Video content analysis algorithms are most often computationally intensive, making it important that the framework overhead in terms of CPU usage is minimized. Furthermore, because of real-time requirements, the processing delay the framework introduces should also be minimal. To measure the overhead the framework imposes, a standalone application is compared to the same application in the proposed framework but deployed on only one host. While the algorithm was running, the CPU usage was measured for three minutes. Two runs were made, the second run executing a more demanding version of the object detection algorithm.

The results of the measurements are presented in Table 1. Both runs indicate a framework overhead of about 12%. For most target applications this overhead is considered acceptable, and can be improved as the current framework implementation is still a rapid prototype.

Table 2: Failover Measurements

Metric:	Value:
# Measurements	20
Min. time	467 ms
Avg. time	584 ms
Max. time	830 ms
Avg. frame-loss	9

5.3 Failover automation

On host failure the framework tries to re-instantiate a failing dock as quickly as possible in order to lose a minimal amount of data. Because of real-time requirements, no data is retransmitted, so the longer it takes to take over the functionality of the failing host, the more data will be lost. In this benchmark the three docks of the evaluation application were deployed on two hosts. The reading and writing docks are deployed on one host and the object detection dock is deployed on the other. The dock performing object detection is deliberately interrupted by killing its dock manager process. The time between this point and the point where the second host has taken over the dock on the failed host is measured in order to calculate data-loss. The results are presented in Table 2 and show that fast recovery is possible using the proposed framework. Losing slightly more that half a second of data on average is acceptable for most surveillance applications.

6. Related Work

In [5] the advanced component system SOFA 2.0 is presented which was created in order to overcome limitations of formerly existing component-based systems. Due to the lack of video streaming support and the service-oriented nature of SOFA 2.0 this component system is considered unsuitable for real-time video streaming applications. Nevertheless, this work inspired some aspects of the proposed framework such as docks being instrumented components and the dynamic re-configuration of deployed applications.

In [13] the OpenDDS component framework is presented which supports complex data flows and dynamic reconfiguration. The drawbacks of OpenDDS are; the lack of video streaming support, the absence of security algorithms and problems with NAT router and firewall traversal. Another problem is the inflexibility of the framework when designing components for rapid prototyping, dynamic data types, for instance, are not supported.

In [14] the GStreamer framework is presented that focuses on audio and video streaming applications. The framework aims at creating single machine multimedia applications by composing existing components called plug-ins. The frameworks lacks presence information which is a main feature of the proposed framework and has no built-in means that support dynamic reconfiguration.

In [8] a network substrate for desktop grid computing named *Orbweb* is presented. This work describes the effort that is made to extend the XMPP protocol in order to meet the substrate needs. This substrate uses XMPP for NAT and firewall traversal and takes advantage of the available security protocols embedded in XMPP. Orbweb is considered unsuitable because no functionality for real-time applications is available.

In [6], Westerink proposes a flexible framework for building multi-media streaming applications. This framework identifies the general architectural structure of streaming applications and using this knowledge to create an easy-to-use framework which is used for some existing applications. The framework proposed by Westerink is targeted at creating single machine applications from existing components, and therefore not suitable to be used as a networked component framework.

7. Conclusion

In this paper the ViFramework, a framework for networked video streaming components targeted at surveillance applications, is presented. This framework provides dock component builders with flexible means to specify docks using XML as definition language. The end-users are provided with easy-to-use tools to create complex application architectures from the available docks. The combination with the XMPP protocol enables the framework to deploy an application on a WAN by using firewall and NAT-router traversal. This enables remote monitoring, control and notifications. Basic resource monitoring on host-level is performed. Gathered information is communicated to a distribution manager in a smart and configurable manner in order to enable application wide load balancing.

A failover algorithm enhances the robustness of the frameworked application. Evaluation of this algorithm shows that failover is achieved within acceptable time. Measurements show that framework has an acceptable computation overhead. Overall the measurements of the presented prototype implementation show that it is efficient and suitable for video surveillance applications.

8. Future work

Future functionality of the proposed framework will include resource usage profiling of docks on the available hosts as is done by Korostelev et. al in [15]. This will enable the distribution manager to predict what resources a certain dock will use on a host. Using this information the distribution manager can deploy applications more efficiently. The dock manager will also perform resource allocation instead of the operating system which is responsible for this in the current implementation.

The current implementation only supports end-to-end connections. More interface types are to be developed to enable other communication constructs such as publish-subscribe and multi-cast.

For now it is assumed that all docks are pre-compiled on the hosts used by the framework. A *Dock Repository* will be developed that allows run-time uploading of docks to hosts. This facilitates adding “blank” hosts to the system on which, on demand, appropriate docks can be installed.

Optimizations to the framework can be made in order to reduce CPU usage overhead.

Acknowledgment

The research reported in this paper has been done in the context of the first author’s master’s project. The project has been carried out at ViNotion B.V. and the support received from the company and its staff is gratefully acknowledged. Furthermore, we thank Johan Lukkien and Egbert Jaspers for their comments on an earlier version of this paper.

References

[1] G. T. Heineman and W. T. Councill, *Component-Based Software Engineering: Putting the Pieces Together*. Addison-Wesley Professional, June 2001.

[2] CANTATA, “Content aware networked systems towards advanced and tailored assistance,” URL, 2011, http://www.hitech-projects.com/euprojects/cantata/.

[3] ViCoMo, “Visual context modeling,” URL, 2011, http://www.vicomo.org/.

[4] B. Kersten, “Instrumentation of networked video streaming components (to appear),” Master’s thesis, Eindhoven University of Technology, April 2011.

[5] T. Bures, P. Hnetynka, and F. Plasil, “Sofa 2.0: Balancing advanced features in a hierarchical component model,” *Software Engineering Research, Management and Applications, ACIS International Conference on*, vol. 0, pp. 40–48, 2006.

[6] P. Westerink and F. Schaffa, “A high level flexible framework for building multi-platform multi-media streaming applications,” in *Wireless and Optical Communications Conference (WOCC), 2010 19th Annual*, May 2010, pp. 1 –5.

[7] XMPP, URL, 2011, http://www.xmpp.org/.

[8] S. Schulz, W. Blochinger, and M. Poths, “Orbweb - a network substrate for peer-to-peer desktop grid computing based on open standards,” *J. Grid Comput.*, vol. 8, no. 1, pp. 77–107, 2010.

[9] “Socks protocol version 5,” URL, 2011, http://tools.ietf.org/html/rfc1928.

[10] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, Apr. 1996.

[11] I. David, B. Orlic, R. H. Mak, and J. J. Lukkien, “Towards resource-aware runtime reconfigurable component-based systems,” *Services, IEEE Congress on*, vol. 0, pp. 465–466, 2010.

[12] L. Rizvanovic and G. Fohler, “The matrix - a framework for real-time resource management for video streaming in networks of heterogenous devices,” in *The International Conference on Consumer Electronics 2007*, January 2007. [Online]. Available: http://www.mrtc.mdh.se/index.php?choice=publications&id=1164

[13] OpenDDS, URL, 2010, http://www.opendds.org/.

[14] GStreamer, URL, 2011, http://gstreamer.freedesktop.org/.

[15] A. Korostelev, J. Lukkien, J. Nesvadba, and Y. Qian, “Qos management in distributed service oriented systems,” in *Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference: parallel and distributed computing and networks*, ser. PDCN’07. Anaheim, CA, USA: ACTA Press, 2007, pp. 345–352. [Online]. Available: http://portal.acm.org/citation.cfm?id=1295581.1295637