Post-print of: H. Krawczyk and J. Proficz, "KASKADA - Multimedia processing platform architecture," 2010 International Conference on Signal Processing and Multimedia Applications (SIGMAP), Athens, 2010, pp. 26-31.

KASKADA – MULTIMEDIA PROCESSING PLATFORM ARCHITECTURE[1]

Henryk Krawczyk
Electronics, Telecommunications and Informatics Faculty, Gdansk University of Technology
Narutowicza 11/12, Gdansk, Poland
hkrawk@pg.gda.pl

Jerzy Proficz
Academic Computer Center – TASK, Gdansk University of Technology
Narutowicza 11/12, Gdansk, Poland
jerp@task.gda.pl

Keywords: Multimedia systems, multimedia processing , cluster computing, distributed architecture.

Abstract: The architecture of Context Analysis of the Camera Data Streams for Alert Defining Applications platform (Polish abbreviation: KASKADA, i.e. waterfall), a part of MAYDAY EURO 2012 project, is provided. A new multilayer processing model for multimedia streams is proposed. The model layers: services, computational tasks and processes are described. The composition of complex services with simple service scenario descriptions is presented. An example scenario and its realization in the environment is provided. The object-oriented domain analysis, component and deployment diagrams with their relations to the model are proposed.

1 INTRODUCTION

Context Analysis of the Camera Data Streams for Alert Defining Applications platform (Polish abbreviation: KASKADA, i.e. waterfall), a part of MAYDAY EURO 2012 project, is designed for implementation and evaluation of multimedia streams analysis algorithms. Its main goal is to support development of the multimedia based applications, currently represented by three pilot projects: detection of dangerous situations in public places, illness recognition in endoscopy and detection of plagiarism.

Because of the high computation expectation, deployment of the platform is placed in the cluster environment of the Academic Computer Center in Gdansk – TASK. The key requirements of the whole platform include efficiency – especially regarding the number of the processed streams, reliability – when usage of a single algorithm on the stream is

[1] The work was realized as a part of MAYDAY EURO 2012 project, Operational Program Innovative Economy 2007-2013, Priority 2 „Infrastructure area B+R”.

not enough, security – the natural requirement for all systems with sensitive data, and fault-tolerance – in case some hardware/system part is damaged.

The usage of the centralized computation site has also the following disadvantages: network bandwidth – many multimedia streams, especially video HD, require fast connection, assurance of proper quality of service – especially latency in client notifications, long delay when starting the tasks due to use of the typical queue system, stream recording – the mass storage capacity.

In the next section, we present the processing model supporting the solution of the above problems. The section three describes the platform architecture based on the proposed model, including UML [9] diagrams, and the last section provides the conclusions.

2 THE PROCESSING MODEL

Figure 1 presents the processing model used during the KASKADA platform design. It consists of four layers, including two layers related to webservices: simple and complex, computational tasks analyzing streams and processes.

The top-level layer represents complex services. They are responsible for the functionality directly provided to the user applications (at client or application server). Their execution is performed according to a defined service scenario. Such a scenario enables composition, cooperation and data exchange between simple services.

The sample scenario below is a part of a video-surveillance system supporting the monitoring of entrances with automatic comparison of the amount of people passing the gates, generating an alert when any gate is overcrowded, version for 2 gates:

1. *call service: task startup (#1) – decoding video stream from the gate 1*
2. *call service: task startup (#2) – background exclusion on the stream received from task #1*
3. *call service: task startup (#3) – man detection on the stream received from task #2*
4. *call service: task startup (#4) – decoding video stream from the gate 1*
5. *call service: task startup (#5) – background exclusion on the stream received from task #4*
6. *call service: task startup (#6) – man detection on the stream received from task #5*
7. *call service: task startup (#7) – counting and comparison of events from tasks: 4 and 6,with parameters indicating alert (event) if the number of passing people on any gate is 20% greater than average*

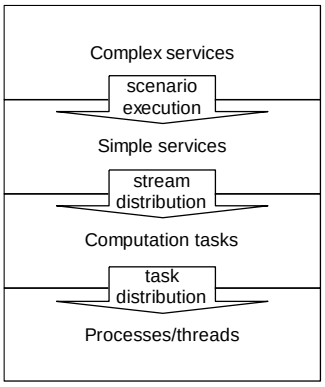
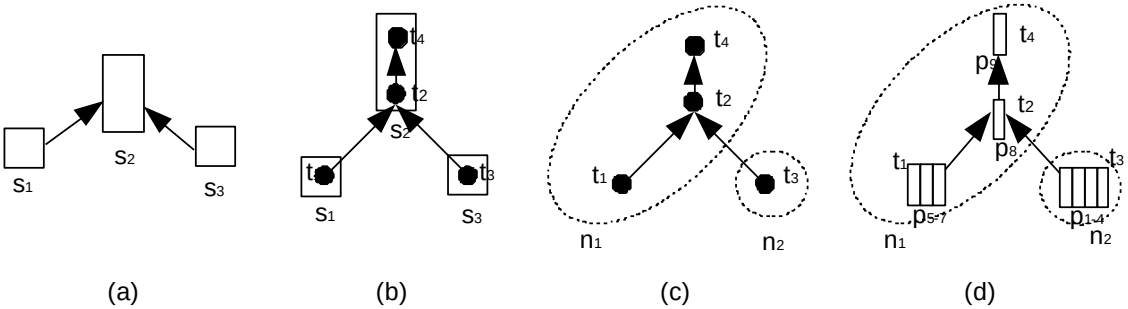


Figure 1 Layered processing model of KASKADA platform

The typical execution of a scenario by a complex service consists of the following steps:

1. *Creation and validation of a service graph.* In the preliminary phase of service execution, the platform creates a graph of simple services used



s_i – simple service, t_i – computation task, n_i – computation node, p_i – process/thread

Figure 2 Phases of preparation to service scenario execution: (a) simple services, (b) task graph, (c) graph assignment to the computation nodes, (d) running processes/thread.

by the particular steps of the scenario. It consists of the vertices representing the services and directed edges indicating data flow. We assume the graph is acyclic – no feedback is allowed. During this step the service descriptions are retrieved from the repository and the types of their input-output data types are validated, see figure 2 (a).

2. *Algorithms' selection and required resource estimation.* In this step, the service graph is converted into a new data flow graph including the computational tasks as vertices and directed edges representing data streams, see figure 2 (b). This transformation is dependent on the requested quality parameters, which can have influence on the algorithm selection as well as on the input data, e.g. camera resolution.
3. *Task assignment to the cluster nodes.* In this step, the vertices of the data flow graph, i.e. computational tasks (derived from the simple services) are assigned to the concrete cluster nodes, see figure 2 (c). We would like to emphasize, this is not a typical scheduling problem (see [4]): the tasks need to be executed concurrently and none of them can be delayed – this is a usual requirement for on-line processing and is more similar to variable sized bin packing problem [5]. The above node assignment can be optimized according to the different criteria, e.g. minimizing the number of partially used nodes (defragmentation), minimizing network load or the delay of the scenario processing.
4. *Scenario startup.* In this step, the computational tasks of the respective simple services are started up on the cluster nodes according to the given assignment. The task identifiers are generated and distributed. The proper data streams are connected and the communication is initialized. Each task consists of one or more processes/threads, whose execution is managed directly by the operating system of the related node, see figure 2 (d).
5. *Scenario monitoring.* During the scenario execution, the platform will monitor the running tasks: processor load, memory usage, multimedia, event and plain data streams' flow. The above procedures are used for continuous collecting and verification of quality related meta-data related to the particular services.
6. *Scenario shutdown.* In this last step, the platform is responsible for the correct finalization of all computational tasks executed with the scenario. During this time, all related processes and threads are finished, the associated resources are freed, the multimedia streams are closed, and the proper information messages are sent to the client.

The next layer of the proposed model is involved in execution of the simple services, which are responsible for selection of the proper algorithm depending on the requested quality parameters. Afterwards, the multimedia stream distribution to the computational tasks is established. For the sake of minimizing network load, the RTSP [12] protocol with the multicast will be used.

The next layer contains the computational tasks, which are the implementation of the concrete stream analysis algorithms. They use the libraries provided by the platform, being embedded into the framework supporting the cooperation with other components of the platform, such as storage or an event server. We can perceive the framework as a template, which already includes common elements used by the algorithm implementation, e.g. an image frame iterator for a video stream, see figure 3. This layer is responsible for task distribution and requested resource acquisition: nodes and processors. We use a typical launcher for these purposes, however it needs to consider additional qualities of service policies, e.g. delays to start of the task.

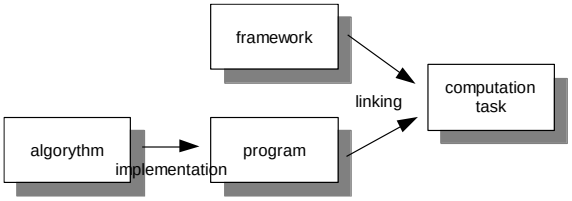


Figure 3 Development of a computation task.

The process/thread layer enables execution of the computational tasks. They can use typical mechanisms of concurrency and parallelism. The platform supports POSIX [16] threads and other similar mechanisms provided by the underlying operating system.

3 ARCHITECTURAL SOFTWARE COMPONENTS

The proposed processing model was implemented as KASKADA platform, below we present the software components. Figure 4 contains the domain model of KASKADA obtained by the requirements' analysis. From the user's point of view the main goal of the platform is to provide the webservices in SOA [7]

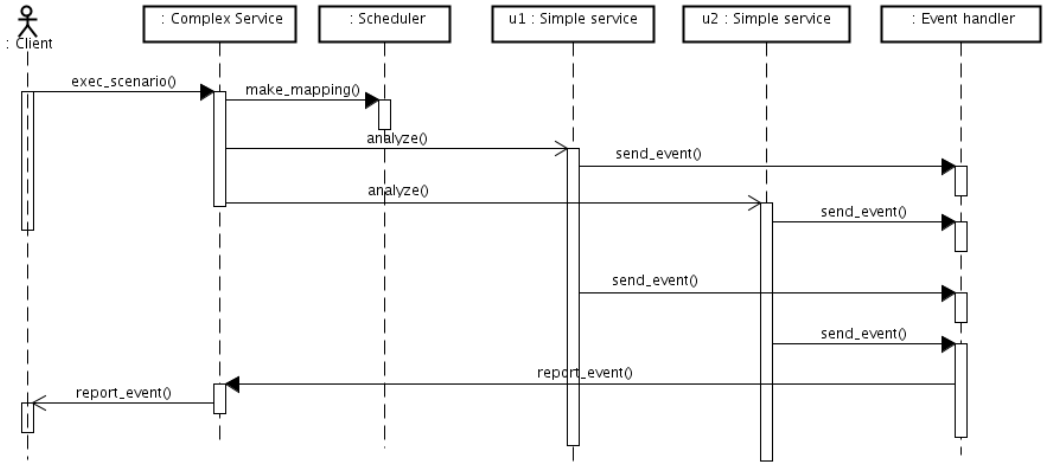


Figure 4 Sequence diagram of the complex service execution.

architecture. They will be responsible for execution of the complex service scenarios using simple services. The example sequential diagram of the scenario execution is presented in figure 5.

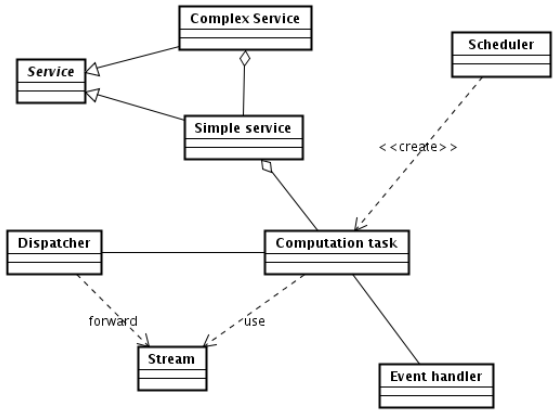


Figure 5 Domain class diagram of the KASKADA platform.

Both service types, i.e. simple and complex ones, are going to be deployed on the same JEE 7 application server, we consider to use a Tomcat web container for this purpose. They will utilize SOAP [18] technologies over HTTP(S) [17] protocol, in case of synchronous remote calls, and a queue system, i.e. ActiveMQ [1] for asynchronous communication within JMS [14] interface. The result return will be performed in separated objects (and components): Event Handler for messages and Dispatcher for multimedia streams.

According to the assumed processing model, simple services manage the distribution of the input and output data streams (see figure 1) for their computational tasks. The object of classes Dispatcher and Scheduler support this functionality. Moreover the responsibility of the Dispatcher object is the stream recording in the storage and sending them back to the client.

Computational tasks – the executable code of the multimedia stream analysis algorithms embedded in the framework accomplish the appropriate computations. They receive the multimedia streams generated by camera, microphone, or other device (e.g. medical equipment) and send an output data stream including discovered event messages or processed multimedia stream, which are delivered to the proper components, respectively Event Handler and Dispatcher, forwarding them through the service layers to the client (see figure 6).

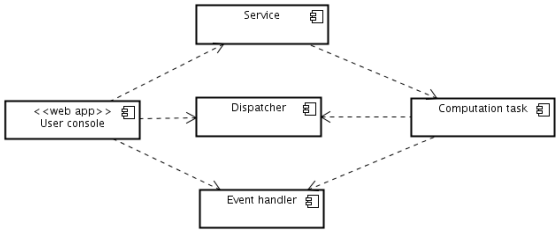


Figure 6 Component diagram of the KASKADA platform.

During the algorithm implementation, the programmer can use software components provided by the computation cluster environment: POSIX threads [16] and openMP [10] library for shared memory processing and object serialization (supported by boost library [3]) for object data exchange between the computational tasks.

Almost all the above domain classes can be straightforwardly converted into the software components of the proposed platform. The only exception is the User Console component which

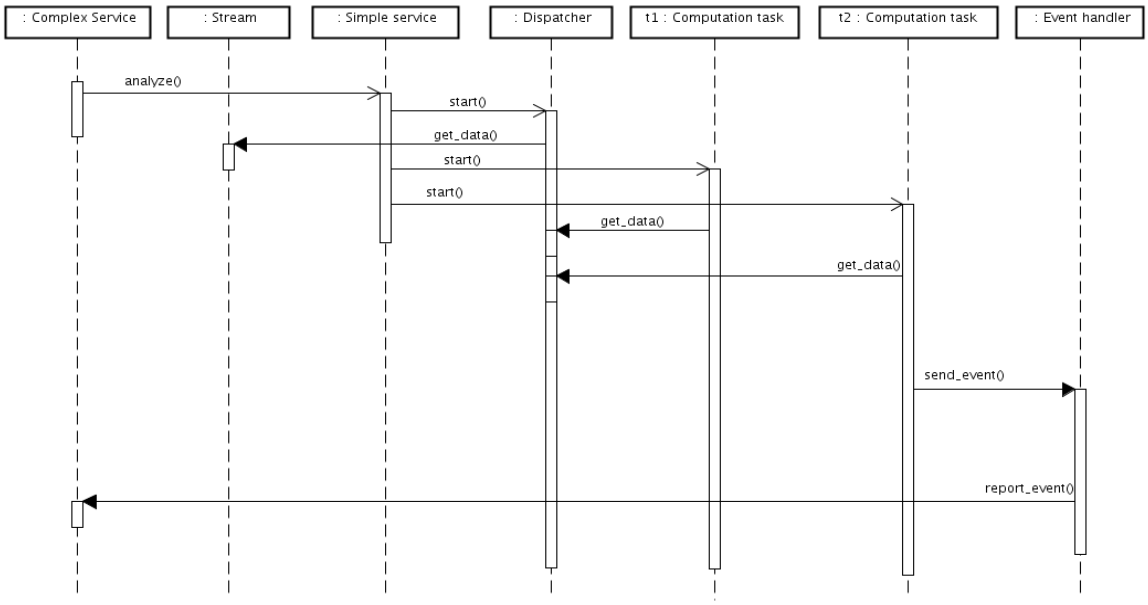


Figure 7 Sequence diagram of the simple service execution.

aggregates Scheduler class as well as manages the other platform components including operations on the multimedia and other data streams (especially in off-line mode – using recorded data), security and service configuration and deployment.

User console functionality is provided through a web interface and can be easily accessed with an Internet browser. For its development, we use JEE [13] standard supported by an application server, i.e. a Tomcat web-container, including technologies: JSP [15] and AJAX [2].

4 HARDWARE AND DEPLOYMENT ARCHITECTURE

To execute computation tasks all software components should be deployed on computer systems. Figure 8 presents the deployment diagram including hardware nodes with the assigned software components. The core of the platform is the cluster executing computational tasks. We use the supercomputer 'Galera' within the Academic Computer Centre in Gdańsk (TASK). It consists of 672 two-processor nodes, each processor has 8 cores, which gives in total 5376 cores with theoretical computational power of 50 Tflops.

The stream managing sever is responsible for multimedia stream format and communication protocol conversion enabling its usage by the computational tasks and receiving by the clients. It is especially important due to the large number of streams and network load minimizing strategy: some cameras or other devices, do not support mutlicast [11] data transmission, so it needs to be provided by the platform. The Dispatcher component is responsible for this functionality, as well as stream recording and archiving.

The process managing server is responsible for direct cooperation with the client software. Here are deployed services and the User console component. It is prepared for serving a large number of webservices, the simple ones – which are easily mapped to the computational tasks, as well as the complex ones – executing the scenarios.

The messaging server supports the Event handler component. It enables receiving, analysis and former processing of the data (but not multimedia) streams containing discovered events. It cooperates with the process managing server where the event related services are deployed.

The data server is used for recorded data storage. We plan to use high performance hard drives with 500TB capacity and the Lustre file system [8], the server is going to be connected to the cluster and other servers by the Infiniband [4] network, for its low delay and high bandwidth.

During the initial phase of the project, three pilot applications are to be developed. The first one is supposed to provide automatic detection, recognition and alerting, for dangerous events and objects in the

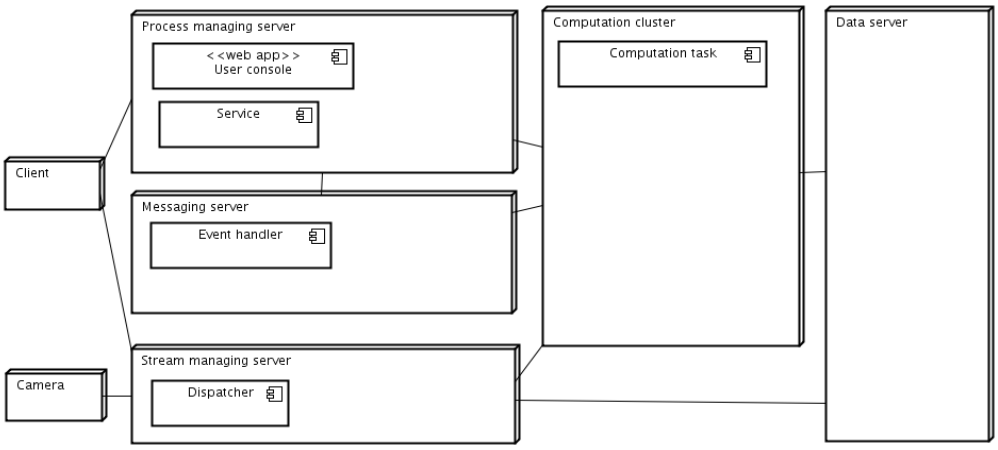


Figure 8 Deployment diagram of KASKADA platform.

audio-video streams received from the security monitoring cameras. The next application is responsible for detection of abnormal characteristics during endoscopy. The third application enables detection of copyright violation of the electronic productions, compositions and documents.

5 CONCLUSIONS

The proposed architecture is supposed to process the great amount of data generated by the multimedia stream sources. The performed requirements analysis indicated the software components to be implemented for the proper functionality and quality. The proof-of-concept prototype is already developed providing an exemplary web-service and the web application for managing and monitoring its behavior.

The current development of the platform is focused on the software components and framework libraries. The future work is going to cover further component development, deployment, and tests. The quality analysis is still to be performed, especially for such factors as: effectiveness, performance, reliability, security and safety. The additional work needs to be done for supporting algorithm implementation and assessment.

REFERENCES

[1] ActiveMQ homepage, http://activemq.apache.org/
[2] AJAX wikipedia page, http://en.wikipedia.org/wiki/Ajax_(programming)
[3] boost C++ homepage, http://www.boost.org/
[4] El-Rewini H., Lewis T. G., Ali H. H., 1994. *Task Scheduling in Parallel and Distributed Systems*, Prentice-Hall Series In Innovative Technology
[5] Haouari M., Serairi M., 2009. *Heuristics for the variable sized bin-packing problem*, Computers & Operational Research 36, 2877-2884
[6] InifniBand Trade Association homepage, http://www.infinibandta.org/
[7] Krafzig D., Banke K., Slama D., 2004. *Enterprise SOA: Service-Oriented Architecture Best Practices*, Prentice Hall PTR
[8] Lustre homepage, http://wiki.lustre.org/
[9] Object Management Group, 2009. *Unified Modeling Language (UML)*, v. 2.2 http://www.omg.org/technology/documents/formal/uml.htm
[10] OpenMP homepage, http://openmp.org/
[11] Savola P., Overview of the Internet Multicast Routing Architecture, RFC 5110, 2008, http://www.faqs.org/rfcs/rfc5110.html
[12] Schulzrinne H., Rao A., Lanphier R., 1998. *Real Time Streaming Protocol (RTSP), RFC 2326*, http://www.ietf.org/rfc/rfc2326.txt
[13] Sun Inc., Java Enterprise Edition (JEE, J2EE), http://java.sun.com/javaee/
[14] Sun Inc., *Java Message Service (JMS)*, http://java.sun.com/products/jms/index.jsp
[15] Sun Inc., *Java Server Pages*, http://java.sun.com/products/jsp/
[16] The Open Group, 1997. *The Single UNIX ® Specification, Version 2*, Threads, http://opengroup.org/onlinepubs/007908775/xsh/threads.html
[17] World Wide Web Consortium, 1999. *Hypertext Transfer Protocol – HTTP/1.1 Specification, RFC 2612*, http://www.w3.org/Protocols/rfc2616/rfc2616.html
[18] World Wide Web Consortium, Simple Object Access Protocol Specification, http://www.w3.org/TR/soap/