15

Real-Time Multimedia Stream Data Processing in a Supercomputer Environment

Henryk Krawczyk and Jerzy Proficz
Gdansk University of Technology,
Poland

1. Introduction

The recent development of surveillance systems due to the threat of many attack sources, (including terrorists, organized and ordinary crime) has forced the municipal and state authorities to provide a wide range of security measures. The appropriate sensors are installed alongside the city streets and other public utilities providing a huge amount of incoming data, which needs to be processed and analyzed, either by a human or automatically by computer software.

The computer centers with high-performance computers seem to be a natural solution for automatic mass multimedia processing, for which the computational power is a crucial factor. Moreover, they are usually located near metropolises, which in conjunction with their usual high-speed network connection, or even their direct placement in the network hubs makes them well prepared to receive huge data streams gathered by all surveillance sensors.

A prototype of such an approach being a real proof-of-concept was built and deployed in the Academic Computational Center and ETI faculty of Gdansk University of Technology in Poland. The proposed solution is realized as a hardware-software platform: KASKADA (Polish abbreviation: Context Analysis of Camera Data Streams for Alert Defining Applications). Its development was performed as a part of the MAYDAY EURO 2012 project. Apart from the platform, three pilot applications were developed: suspicious object and dangerous events recognition; endoscopy examination and disease identification, and intellectual property analysis and protection.

Though the proposed platform is a new idea, it can be compare to a the typical distributed/parallel programming frameworks. In comparison to the typical grid computing solutions: Globus (Foster & Kesselman, 1997) or Unicore (Breuer et al., 2004), it provides built in mechanisms for quality of service in the developed services and integrated approach to data stream computation management. For the more classical high performance computation architectures, like MPI with some task queue (eg. PBS), the platform provides service-oriented and real-time features, enabling easy development of the live multimedia streams services and user applications.

The proposed platform can be also compared to the typical distributed/parallel programming frameworks, like J2EE or .NET, however they are general purpose solutions

and have only limited support for massive multimedia stream processing. An interesting platform was proposed in (Yu et al., 2009), in general it is also dedicated for multimedia real-time processing, however the mechanisms used are different, where the computations are concentrated in the database layer, while the KASKADA platform is focused in the middleware.

2. Challenges of the distributed multimedia processing

The typical supercomputer system is designed for the scientific-based simulation-like computations. It usually works over a distributed operating system under supervision of a batch-based queuing system, collecting requests as a list of computational tasks. Their processing is executed off-line without direct communication with the user. The multimedia processing, related to the video and audio surveillance systems, requires a real-time environment, supporting the continuous processing of live data streams, which brings new challenges for system and middleware software.

The first recognized requirement for the computational platform is to provide a proper concept of service management. We assume a service is a piece of functionality performing computations on an input multimedia stream, exposed by a well-defined interface to a user application. Thus, the platform needs to provide support for development, tests, and execution of such services. The proper tools for stream algorithm implementations, service creation and composition, are needed. Moreover, the platform needs to provide a standard service repository for conversions, decoding, encoding, and distribution of video and audio streams. Another important aspect is runtime management of services, where the proper mechanisms of resource allocation and monitoring need to be considered, we proposed the set of algorithms enabling the proper resource management with the optimization of fragmentation of the cluster (Krawczyk & Proficz, 2010).

Apart from a proper computational model, the proposed solution needs to guarantee an appropriate level of usability. The applications using the designed platform should provide a simple and clear user interface; thus, the platform itself should have a set of features enabling easy service development and composition. The solution should unify the processing model and be properly tuned for video and audio streams analysis.

Another problem needing to be solved is event management. We assume any working service performing stream analysis can reach a set of conditions, when the special actions need to be performed by other services or even outside the platform. Such an event can be asynchronously delivered using the message-passing mechanisms provided by the platform. It's worth noticing that the number of recognized incidents can burst drastically, causing a high computational and communicational load, which should be properly handled preserving all required constraints related to event filtering, message distributing and delivering.

The above requirements demand the usage of high-performance systems, in both computational and communicational areas. With respect to other aspects, like security or safety, we decided to build the platform based on the computational cluster located in the academic computer center. It consists of 1132 computation nodes, each containing two multicore processors, with 10896 of available cores. The nodes are interconnected by a high

speed 20-40GB/s Infiniband network. The above configuration gives solid capabilities for parallel processed services, based on the parallel computations, analyzing hundreds of multimedia streams.

The specific characteristics of the performed analysis – real-time, security-related services rely directly on the dependability of the underlying platform – the dedicated platform needs to provide the ability to replicate the same computations, thus more accurate results can be obtained. Moreover, the resource allocation algorithms must provide the minimal, guaranteed level of resources so that the executed analysis is performed smoothly without delays and data traffic jams. We assumed (the typical) three types of resources to be managed directly by the platform: CPU load, indicating the complexity of the performed computations, memory depending mainly on the size of the problem context, which needs to be maintained during the analysis, and network bandwidth used to provide the flowing multimedia stream.

The increasing number of the analyzed services, depending directly on the incoming data, requires the platform to provide a flexible way to extend its computational and communicational capabilities. Thus the scalability of the proposed solution is a crucial factor for the final outcome of the executed services. The management procedures need to comply with the constraints and limitations given by the underlying hardware according to capability for the extension of its resources. Moreover, even for a single node the increasing utilization of the resources (cores, memory) used by the executed services needs to conform with the speedup of their computations.

Finally, taking under consideration performance, dependability, and scalability of the proposed solution, the proper means for maintaining the required platform state have to be developed. The monitoring subsystem is responsible for measuring and control of the cluster nodes and executed services. In case of abuse, such a component needs to be properly handled, the service needs to be stopped and the node isolated from the cluster. Moreover the monitor provides the external API for tracing the above measurements by the utility applications, so as to be able to react flexibly to the occurring problems.

3. Multimedia processing model

Fig. 1 presents a layered processing model for multimedia processing. The whole platform is deployed in the cluster environment – system infrastructure including a Linux operating system, computation nodes and the network. On the other side, it serves as a middleware for user applications, which are directly responsible for the interactions with the users (Krawczyk & Proficz, 2010b).

Apart from the applications and the infrastructure, the model consists of the following four layers: (1) complex services, (2) simple services, (3) computational tasks, and (4) processes. The top layer manages the complex services exposed directly to the user applications, which are working according to defined scenarios of simple services included in the underlying layer.

Figure 2 presents an MSP-ML example of a complex service scenario being 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. Two cameras are used capturing video streams and implementations of the following algorithms: decoder – unpacking encoded video frames, background remover – the algorithm detecting moving objects in the video stream and removing the background, human detector – the algorithm detecting a human silhouette in the incoming images, event counter – comparing the number of messages with detected events describing incoming people and signaling large imbalance between them.

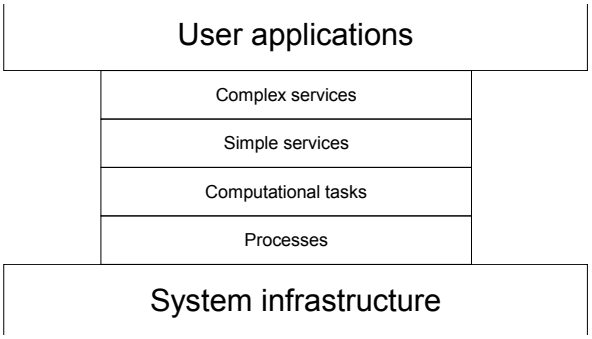


Fig. 1. KASKADA platform processing layers

- simple service (1) – decoding video stream from gate 1
- simple service (2) – background exclusion on the stream received from task #1
- simple service (3) – human detection on the stream received from task #2
- simple service (4) – decoding video stream from gate 2
- simple service (5) – background exclusion on the stream received from task #4
- simple service (6) – human detection on the stream received from task #5
- simple service (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

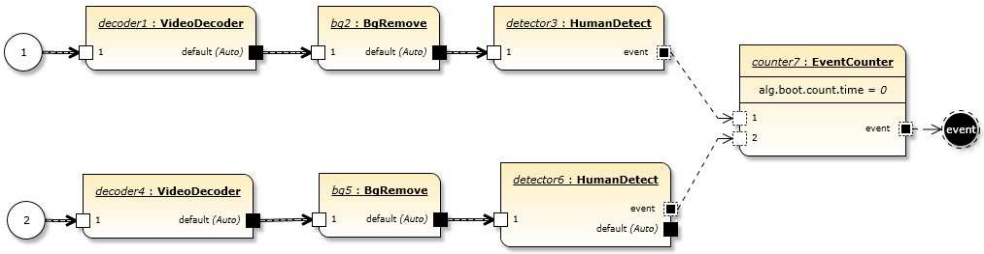


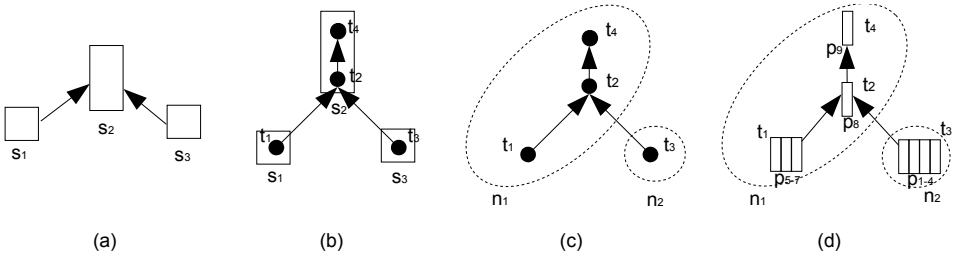
Fig. 2. An example of a complex service scenario in MSP-ML

The **complex services layer** is responsible for execution of scenarios consisting of complex and/or simple services according to the following steps:

1. *Creation and validation of a service scenario.* In the preliminary phase of service execution, the platform creates a sequence of simple services used by the particular steps of the scenario. It consists of the vertices representing the services and directed edges indicating data flow. We assume that such graphs are acyclic – no feedback is allowed.

The service descriptions are retrieved from the repository of scenarios and their input-output data types correctness id validated, see figure 3 (a).

2. *Algorithms' selection and required resource estimation.* In this step, the service scenario is converted into a new data flow graph including the computational tasks as vertices and directed edges representing data streams' flow, see figure 3 (b). This transformation is dependent on the requested quality parameters, which can have influence on the tasks algorithm selection as well as on the input data choice, e.g. camera resolution.



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

Fig. 3. Phases of preparation to service scenario execution: (a) simple services; (b) task graph; (c) tasks' assignment to the computation nodes; (d) tasks running as processes and threads.

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 3 (c). We would like to emphasize that the tasks need to be executed parallel or concurrently, satisfying requirement for on-line processing and is more similar to a variable sized bin packing problem. The optimized criteria can be as follow: minimizing the number of partially used nodes (defragmentation), minimizing total network load, or the total delay of the scenario processing (Krawczyk & Proficz, 2010a).
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 assigned to the tasks and the communication is initialized. Each task after initiation consists of one or more processes/threads, whose execution is managed directly by the operating system installed on the cluster nodes, see figure 3 (d).
5. *Scenario monitoring.* During the scenario execution, the platform will monitor the running tasks and evaluate the following parameters: processor load, memory usage, multimedia, event and plain data streams' flow. The evaluation procedures are used for continuous collecting and verification of quality related meta-data describing the particular services.
6. *Scenario termination.* In the last step, the platform is responsible for the correct termination of all computational tasks executed with the scenario. This means that, 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, presented in figure 1 is involved in execution of the **simple services**, which are responsible for selection of the proper algorithm form alternative proposition according to the requested quality parameters. Moreover, the multimedia

stream distribution to the computational tasks is established. For the sake of minimizing network load, the RTSP protocol with the multicast is used.

The next layer corresponds to the **computational tasks**, which are the implementation of the concrete stream analysis algorithms. They use the libraries of special functions such as cooperation with other components of the platform, including storage or an event server. It perceives the framework as a template, which already includes supporting objects used by the algorithm implementation, e.g. an image frame iterator for a video stream. This layer is responsible for task distribution, and requested resource allocation: nodes and processors. It also uses a typical launcher for these purposes, besides additional qualities of service policies, e.g. delays to the start of each task are considered.

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

The model described above was implemented in the KASKADA platform, all its layers are realized and their cooperation is managed by specialized servers. The description of the platform architecture is shown in section 6.

To create user application we build, we use a suitable application server cooperating with the KASKADA platform. The main part of the application is a specialized interface which allows to perform one or more of the possible user scenarios consisting of simple or complex services running on the KASKADA platform.

4. Development environment for multimedia stream processing

Fig. 4 presents the typical flow of activities leading to building of a complex service scenario. We start with an idea of multimedia stream processing, for instance identification of well-known objects or events. Then we can focus on the algorithm described in pseudo-code or other high level language, eg. flowchart. When that transition appears, we translate them to the real source code. It is compiled and linked into a computation task executable code. Having the executable algorithm program we enrich it with the quality and parameters metadata to obtain a simple service. The platform provides automatic generation of the WSLD description of this service, which in turn can be used to build more complex structures. The special language MSP-ML was designed to express the service composition keeping characteristic stream-processing constraints.

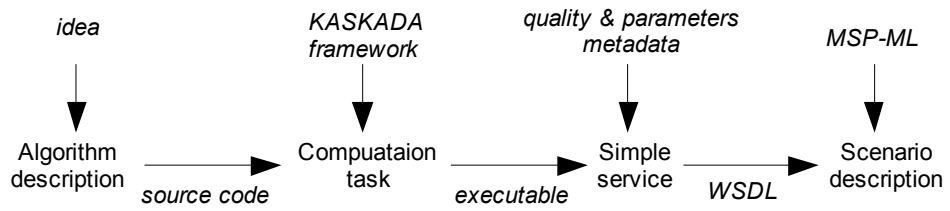


Fig. 4. A typical flow of activities of complex service creation

4.1 Algorithms to computation tasks

The KASKADA platform supports execution of multimedia stream analysis algorithms as computation tasks. They can be able to receive an input stream to detect objects and events, and finally to generate the output stream. Moreover, the tasks can be composed in pipelines or even more complex structures, so that the results of one task can be used as an input of the others. For instance we can distinguish a task detecting faces in the monitored video stream, and then one recognizing specific properties of the face, and a final task checking the database of persons wanted by the police.

The KASKADA platform provides a dedicated framework supporting implementation of the analysis algorithms. Every algorithm, embedded in the KASKADA framework, can use a special API providing the platform functionality: multimedia encoding and decoding, intermediate results interchange, signaling the events, processing the input parameters, and stream synchronization. The algorithms are implemented in C++ and together with the framework create computation tasks (see Fig. 5) under the Linux operating system in the cluster environment.

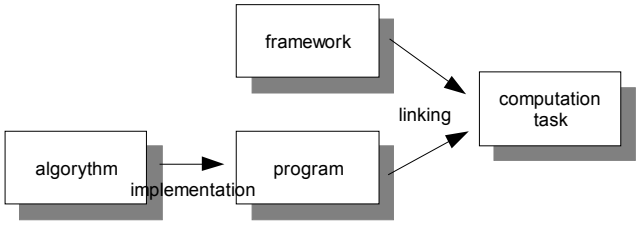


Fig. 5. Development of a computation task

Multimedia streams captured by the cameras or microphones are received under the RTSP protocol, the framework decodes the flowing video frames or audio probes and forwards them as input data to the task. Additionally, it can use the metadata related to the received stream, eg. the currently processed frame timestamp, resolution or video frame rate, or the audio probing frequency.

Another interesting feature provided by the KASKADA framework is a mechanism managing input parameters. They are formatted according to the convention used in GNU GPL license software. The framework recognizes the parameters related to the streams, filters and interprets them, in order to reconfigure the implemented algorithm. The platform provides the methods for parameters' propagation through the platform layers, so they don't need to be distributed manually to the destination services or tasks.

From an input data distribution point of view any task can be executed in three different modes: off-line streaming from a file, off-line streaming from a streaming server, and on-line directly from the stream source (see Fig. 6 a-c). The first mode enables direct usage of archived streams using a network file system, that the processing can be even faster than the recording, eg. for video streams the processing frame rate is limited only by the CPU speed.

The streams transmitted in the off-line streaming server mode are provided in a very similar fashion to the live streaming, with the exception of the data source, which is located in the off-line archive. This approach supports the tests using the archived content, but within the

constraints of the live streaming. The later mode is based on the direct connection to the streaming device, ie. a camera or a microphone enabling their real-time processing.

From the point of view of output data distribution, the algorithm execution and tests can be performed using archiving or streaming modes (see Fig. 6 d-e). In the first case the processed output stream is stored into the archive file, and in the latter the results are transmitted by a streaming server to the client. There is also the possibility to use a hybrid approach where the data are simultaneously stored in the archive and transmitted after some transformations to the users (see Fig. 6 f).

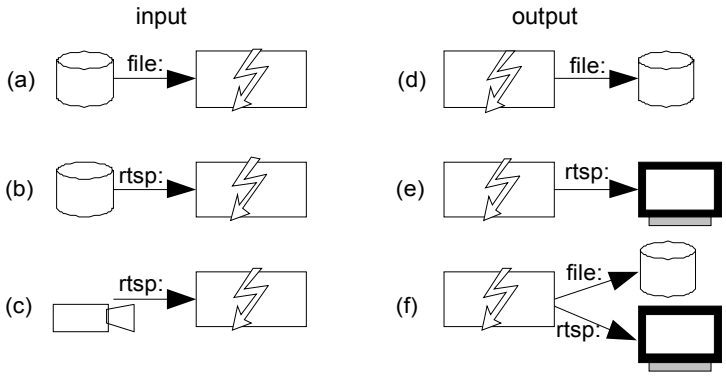


Fig. 6. Input/output algorithm modes, for input: (a) off-line from a file; (b) off-line through a streaming server; (c) on-line from a streaming device; for output: (d) archiving to a file; (e) transmitting through the streaming server; (f) hybrid.

The above algorithm input/output modes directly implies interaction modes with the services or even the whole platform. Thus we can distinguish the following interaction modes:

1. Real-time mode – when the multimedia streams are directly processed and the results are provided to the user.
2. Off-line mode – when the streams are read directly from the archive and the results are stored for the user.
3. Real-time simulator mode – when streams are read from the archive, but they are transmitted to the tasks just like from their original source, and the results are provided to the user – used for testing purposes.

Apart from the multimedia stream, any running algorithm can send the messages enclosing the detected objects or events it found during the processing. The KASKADA framework provides the message passing API, and the platform contains the mechanisms of their routing, delivering, and monitoring. Fig. 7 presents an example source code for sending the message, and the screenshot of the web browser with monitored messages containing the information about reported events.

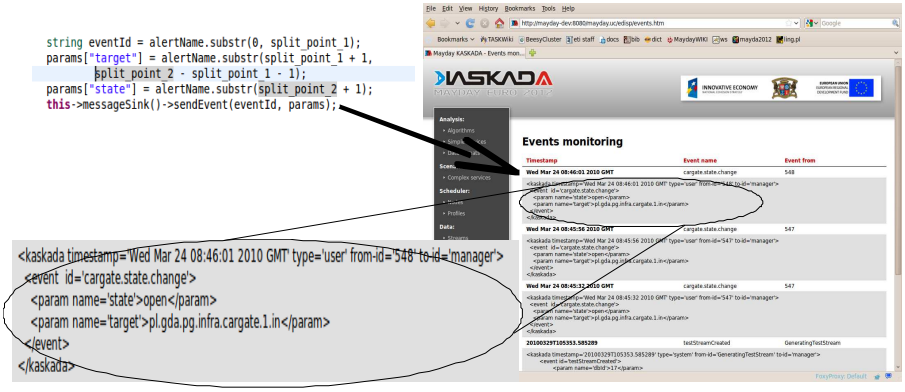


Fig. 7. Example of sending and monitoring the messages describing events

4.2 Algorithm to service transformation

The implemented algorithm, after linking with the KASKADA framework libraries and its preliminary tests, can be used as a base for creation of a simple service, which can be offered as its remote access. From the point of view of a programmer, the service is an interface backed up by the task and accessible through the platform.

It is possible to implement a few alternative algorithms realizing the same service, but providing different levels of processing quality, ie. dependability, performance. On the other hand, there can exist an algorithm matching two, or even more, services providing their functionalities. Fig. 8 presents an example of such multiple relations between algorithms' implementation and services.

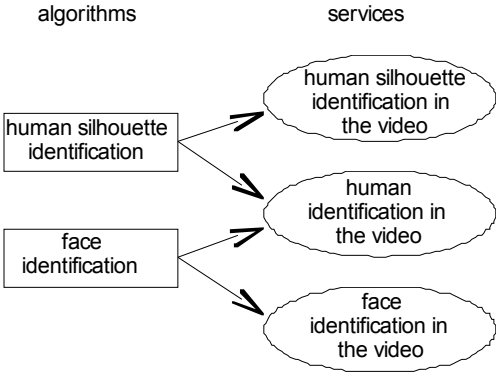


Fig. 8. An example of multiple relations between algorithms' implementation and services

The KASKADA platform stores algorithms' metadata, as well as providing the means to manage them. For this purpose an appropriate user interface was implemented including such functions as adding, removing and modification of the data: an algorithm's name, executable file path, input and quality parameters. An exemplary list of algorithms was presented in Fig. 9.

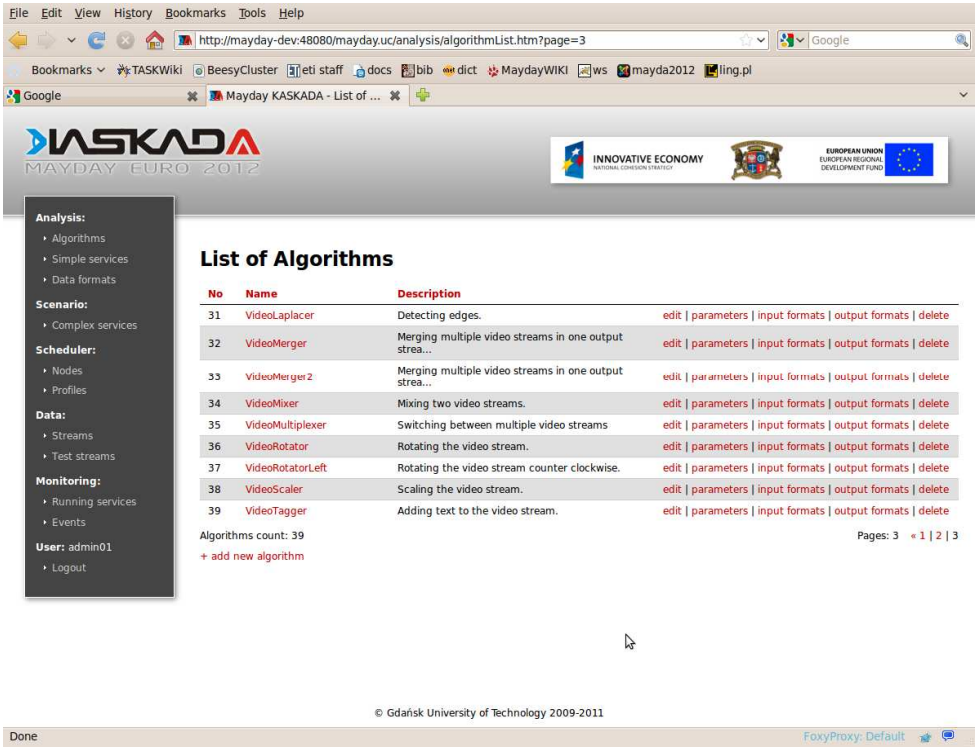


Fig. 9. An example of implemented algorithms in GUI of the KASAKDA platform

The algorithms, implemented in the KASKADA platform can be based on different processing paradigms. They vary from the typical high performance numeric computations, through different heuristics like different classifiers, Haar cascade, to more sophisticated artificial intelligence solutions like identification of objects by GHM (Gaussian Mixture Model), Kaman filter and codebook model, or even expert-systems for endoscopies disease recognition.

Similarly to algorithms, the simple services are described by their metadata, including: a service name, the description, input and quality parameters. Additionally, the implemented algorithms realizing the suitable service can be easily point out by simple name selection from the list. Every selected algorithm can use the service input parameters passed during its execution, however their handling is optional.

The developer can start the service in two modes: by remote call or using a web browser. For testing/debugging purposes the latter one is more feasible, when she/he just needs to introduce the parameter values and click the "start" button. Both modes are provided automatically by the platform, just after the service definition. Fig. 10 presents the steps for service definition and execution. After successful initiation, the developer can check the result data stream and the events in the messages generated by the algorithm and logged in the special logs.

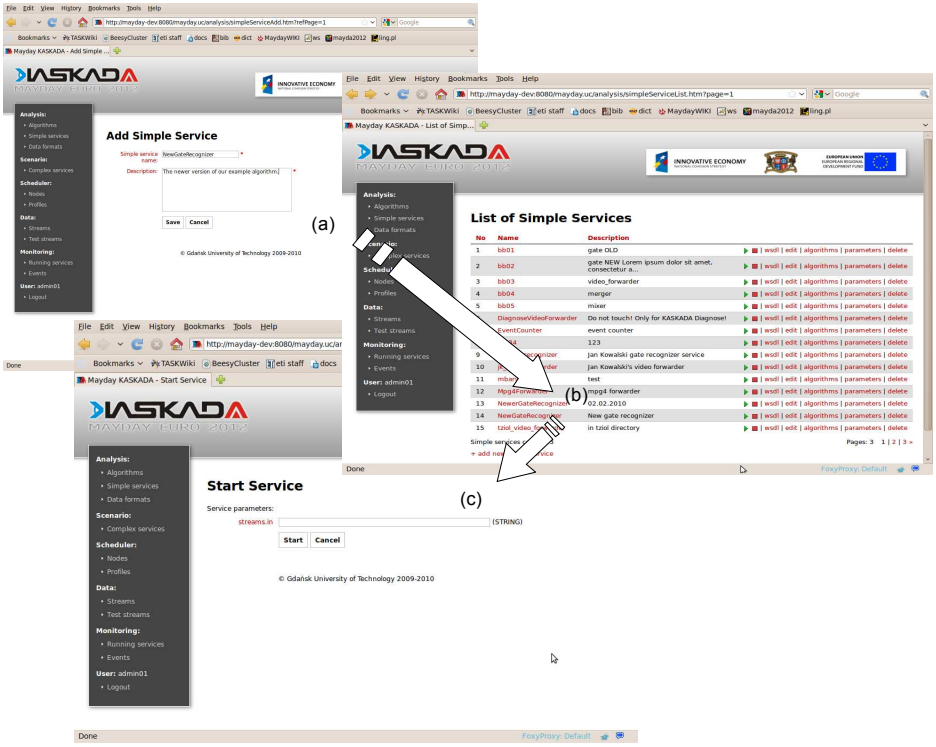


Fig. 10. Steps required to add and execute a simple service: (a) service definition, (b) service selection from the list, (c) entering input parameters

The remote call of a simple service can be realized using HTTP/SOAP protocols. In such a case the programmer is responsible for a proper implementation of client being part of user application. To simplify this process, the KASKADA platform provides its own UDDI registry, where the services are described using always-updated documents in WSDL format.

4.3 Simple to complex service transformation

Similarly to a simple service solution, the complex services are accessible by a web browser or through the external interface based on SOAP/HTTP protocol. The choice also depends on the input and quality parameters, and accept the multimedia streams as an input. They aggregate simple services within execution scenarios. ie. in more complex structures, which extends their functionality.

Fig. 11a presents an example of an execution scenario, discussed in section 3. The scenario can be described by XML language as a set of simple services and input/output data stream definitions, see Fig. 11b. The document is created by the developer using a typical text editor, or a specialized language: MSP-ML. Based on this we entered a GUI interface as the proper form as shown in Fig. 11c.

Every single service needs to have provided input and quality parameters, specific to its underlying algorithm. They can be introduced in two ways: forwarded from the complex service description, or fixed during the complex service definition. Using this approach, we achieve flexibility required for the service execution.

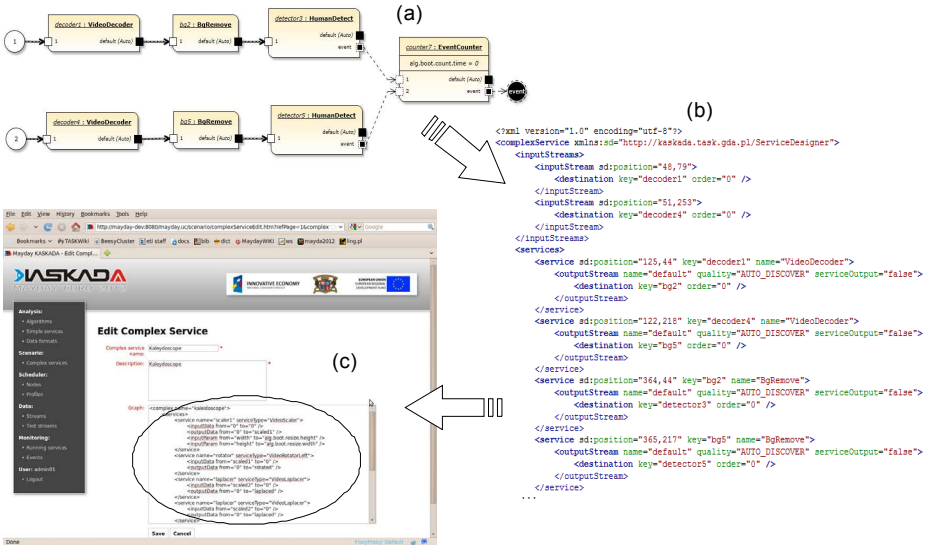


Fig. 11. From complex service scenario definition (a) trough XML document (b) to GUI service definition (c)

5. Execution environment for multimedia stream processing

We distinguish four domains of the platform management during service execution: (1) data, (2) computations, (3) communication, and (4) resources. Data as real-time streams of the incoming multimedia data either video or audio, need to be archived and distributed among computation tasks. Computation means the stream processing tasks, which are involved by instructions of program. Communication is understood as message exchange mechanisms enabling distribution of the events among service components and sending to the external client. Finally, the cluster resources are engaged in computations and constantly observed by the monitoring mechanisms, including checking and reacting on the inappropriate usage of the network, CPU and memory.

5.1 Data – Multimedia stream management

The multimedia data streams are generated by a geographically distributed set of video cameras and microphones. They need to be delivered to the computation center and preprocessed for further analysis. The number of streams, and their characteristics, demands the usage of a high bandwidth optic fiber network.

Fig. 12 shows the view of the multimedia streams in the considered platform regarding the users, as well as the external applications using the platform functionality. The arriving data

needs to be received, validated, and archived, due to additional offline analysis and legal concerns. The above operations are performed by the tasks controlled by the management server. The server can also cooperate directly with the stream sources. Preprocessing is used to unpack the stream data from its native protocol, usually RTSP, and forward it to the proper analysis tasks located on the computational nodes using an Infiniband network. The higher platform efficiency can be achieved by allocation of tasks with streams to more cluster nodes. A special set of the cluster's computational nodes is assigned to perform such tasks using the high performance network file system LUSTRE. Physically, the data is stored on a specific data server with high performance 500TB hard drives.

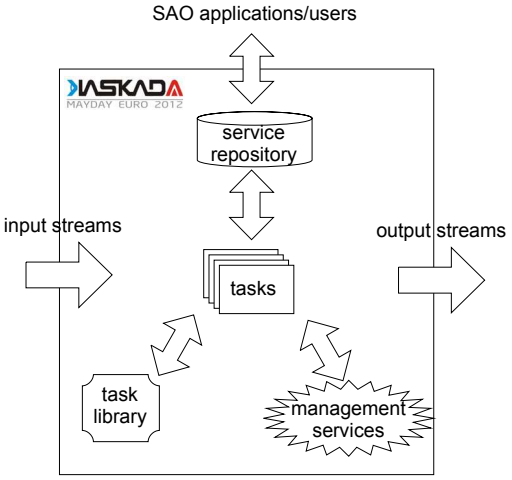


Fig. 12. Data and user interoperability of the KASKADA

The various algorithms exposed as services can consume incoming streams and produce messages (as events created during the analysis) as well as new output streams, which should be delivered to the user. See the next section for more details about the flow of events.

The information about produced output streams are sent back to the stream management server nodes and they are ready to be delivered to the selected users. It can be done by the user console module. The stream-related functionalities of the user console cover:

- registration and testing of new multimedia (audio or video) streams,
- configuring the meta-data of the stream, including codec, fps, width, height etc.,
- archive configuration with individual settings of each stream,
- creation (by upload or archive selection) of the test streams used for benchmarking and testing of the algorithms,
- playing and replaying of the live or archived streams.

5.2 Communication – Event management

An application controlling the processing of the started service can receive the results of the predicted analysis by means of event processing. An event, as specified in section 1, is information generated by a task belonging to the service, which is potentially important for

external applications or their users. Such information is expressed by a message transported in XML document format. The type of the message and its content are determined by the particular algorithm implemented by the task. It is not compulsory for the task to finish its work after creating and issuing an event; the multimedia streams can be processed continuously causing generation of many events for different situations detected in the streams during their processing. Such series of the events can be conceived as the events stream or the data stream (Olken & Gruenwald, 2008). The processing of such a data stream can be focused on the selected events, treating them independently, or as the sequence of events, where state-aware operations analyze one event after another.

Fig. 13 shows the event processing idea in the KASKADA platform; starting from the event creation, through event handling and ending with the special message being passed to the specified destinations.

The tasks belonging to the service searching media streams, with respect to detection of special object and/or particular situations. The successful detection of such situations causes generation of an event containing information about its origin and processed media time, apart from data related to the particular event type. Then, each event is passed to an Event Handling Module through the message queue maintained by Apache ActiveMQ. The choice of ActiveMQ as the message queue provider is, among other things, determined by different technologies, C++ and JEE, being the runtime environments for the event processing flow elements. The Event Handling Module performs the operations during event processing, such as save, apply events, create and send message (see Fig. 13).

Operation: Saves the event in repository: each event incoming to the Event Handling Module is stored for administrative and safety purposes, using the Events Repository mechanisms. The storage structures associate the event with the data stream related to the service in order to enable later backtracking of the selected service results. Using the information about the service-starting details, preserved by the KASKADA platform, and the archive of media streams, it is possible for the particular event to reconstruct the conditions which led to such an event generation.

Operation: Applies filters to the event: the user console of the KASKADA platform supports a choice of filters applicable to the event processing. A filter can be applied to check if particular properties of the event fulfill criteria defined for that filter. The criteria for a filter can specify a service or XPath expression. If there is consistency between XML content of the event and the specified filter settings, the service and the expression, the event becomes active.

Operation: Sends the event message to the channels: the filter applied to the processed event contains one or more channels. A channel represents a final destination for the information about processed event. The XML message describing the active event is delivered to the recipients specified by details of each channel. The information contained in the channel details is determined by the channel type which represents the transport mechanism used by the particular channel. The currently-supported channel types are the e-mail type and the JMS compliant message queue system type. Nevertheless, the KASKADA platform is ready to support additional types, and also for implementing the transport mechanisms related to them.

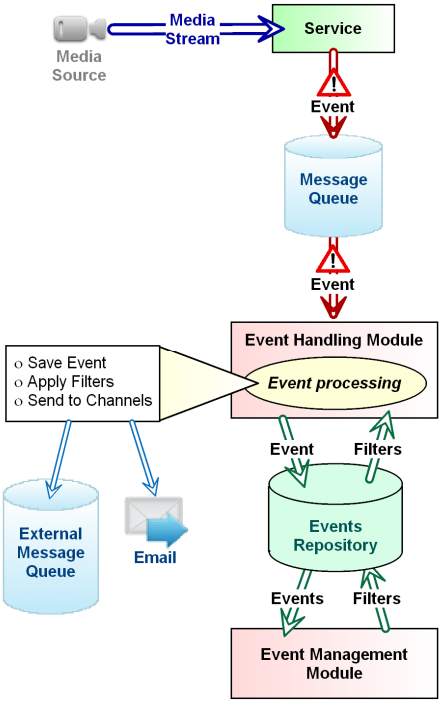


Fig. 13. Event processing idea in the KASKADA platform

An Event Management Module is a part of the user console of the KASKADA platform; and, as other modules of the console, is designed to work with an end-user in an interactive mode. There are a few functionalities supported by the module (apart from the described-above definition) and management of event filters. The module makes available a real-time monitoring of incoming events related to the particular service while it is still running. The console user can inspect the archived event streams, for example searching for particular circumstances. The module allows replaying the part of the archived media stream related to the selected event during its examination, adjusting the time frame of the played part if necessary.

Different types of destination channels open the KASKADA platform to the various technologies of implementation of the applications receiving the results of calculations of services. Apart from the technology, there are two typical scenarios of the interoperability of the user applications and the KASKADA platform. The scenarios can be described as follows:

- The operator of an external, interactive, application starts a service through the application monitoring mechanisms. Then, the application processes incoming events and stops the service; for example after receiving a particular event, presenting the operator with the processing results. The above sequence applies to the processing of a selected part of the stream, as a result of a more general, continuous, analysis.
- The advanced, distributed, application automatically starts a service with mechanisms contained in one of its components. The started service, without stopping its work,

generates events which are delivered to another component of the same application, or even to another application. The scenario separates the management of a service execution from the pure event processing, allowing the application to be more flexible and modularized on a high level.

5.3 Computation – Service and task management

As was described in section 1, the complex services are executed as a set of simple services according to the describing scenario. The simple services are realized by the computation tasks, which in turn are represented by the processes and threads in the cluster environment (see Fig. 1 for comparison). The tasks need to be placed in a suitable computation node, according to the appropriate task schedule achieving the required computation time.

The typical problem of task scheduling is defined as assignment of a set of tasks into the set of computational nodes in an order to execute them in minimal time. It is proved, the above problem is NP-hard for the general case, and there exists a polynomial solution for task scheduling on two computational nodes (El-Rewini & Lewis, 1994).

The proposed architecture assumes that tasks are executed continuously, consuming and producing data streams. Each of them requires concrete computational power to realize the provided functionality. We assume that due to the character of multimedia stream processing, the tasks require real-time execution, and they cannot be queued and started with delay in the sequence one by one.

We propose a specific a task-to-nodes assignment strategy regarding the above constraints. The tasks could be assigned to the computational nodes, which are able to execute them directly. Otherwise, if there no such nodes exist in the cluster, the service scenario (scheduled tasks) is not executed, its execution is refused and signalized to the user. To minimize the occurrences of such cases the platform should optimize its selection using the criteria of minimizing the fragmentation.

In such a way the assignment strategy is quite similar to the well-known bin packing problem (BPP) (Garey & Johnson, 1979), especially its version with the variable bin sizes (VBPP) (Haouari & Serairi, 2009). Typical BPP minimizes the number of baskets (computational nodes in our case) used for packing a set of objects (tasks). The version with the basket variable sizes introduces additionally a finite set of basket types with different sizes. An exact algorithm for (V)BPP is NP-hard (Garey & Johnson, 1979).

In the proposed assignment strategy we can use as many baskets of possible type (size) as we need, but the number of each type is finite. Moreover, the optimisation goals are different, VBPP minimizes the number of used baskets and our startegy considers the number of partially used nodes.

The fragmentation factor indicates the number of nodes partially engaged in task processing. Fig. 14 shows a situation where a new heavy task cannot be assigned to any node, because all nodes (c_1- c_4) offer less computation power than it is needed for the task t ($\Phi(t)=6$). In (Krawczyk & Proficz, 2010a), we proposed a few heuristic algorithms to minimize the fragmentation what in turn enables execution of more tasks with high load, ie. $\Phi(t)=6$ in the example.

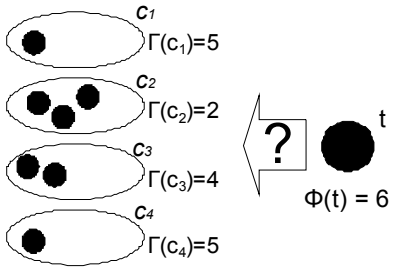


Fig. 14. Example of the cluster fragmentation preventing a new task assignment

5.4 Resources monitoring

In the case of the KASKADA platform we distinguish three kinds of resource characteristics: network bandwidth, memory size and CPU load. According to the assumptions described in the previous section, each computation task needs to have guaranteed the proper amount of the resources, and the tasks behaving incorrectly have to be deactivated and their service stopped. Moreover the monitoring information needs to be presented to the external client using either GUI or by the webservice interface.

The KASKADA platform monitor is responsible for the following functionality:

- managing the services and tasks,
- monitoring the current resources utilization,
- informing the user about the tasks' errors and exceptions,
- logging the information about the platform, services and tasks behavior,
- checking the computational node states related to network connection and file system behavior.

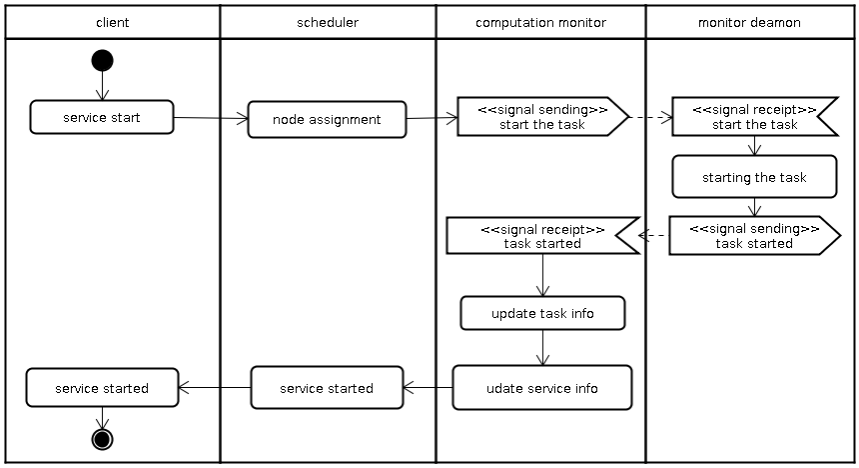


Fig. 15. The basic activities for starting a service in the KASKADA platform

The KASKADA monitor consists of two functional components: computation monitor – the central component keeping the states of the services and their tasks and monitor daemon, running on every single cluster node, which cooperates directly with the tasks, gathering the information and executing orders from the computation monitor. Fig. 15 shows an example diagram of the scheduler and both monitor components when the user starts a service.

6. KASKADA – Architectural design and realization

6.1 Software architecture

The proposed in section 3 processing model was implemented as the KASKADA platform. Fig. 16 presents the main classes to satisfy functional requirements. From the user's point of view the main goal of the platform is to provide the webservices based on SOA architecture. They will be responsible for execution of the complex service scenarios supported by simple services. The example sequential diagram of the scenario execution is presented in figure 17.

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

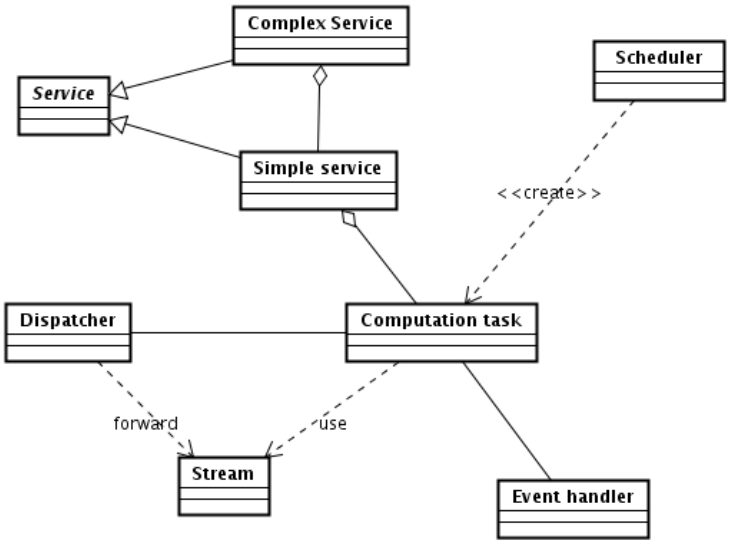


Fig. 16. Domain class diagram of the KASKADA platform

According to the assumed processing model, simple services manage the distribution of the input and output data streams among 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. The example sequential diagram of the simple service execution is presented in Fig. 18.

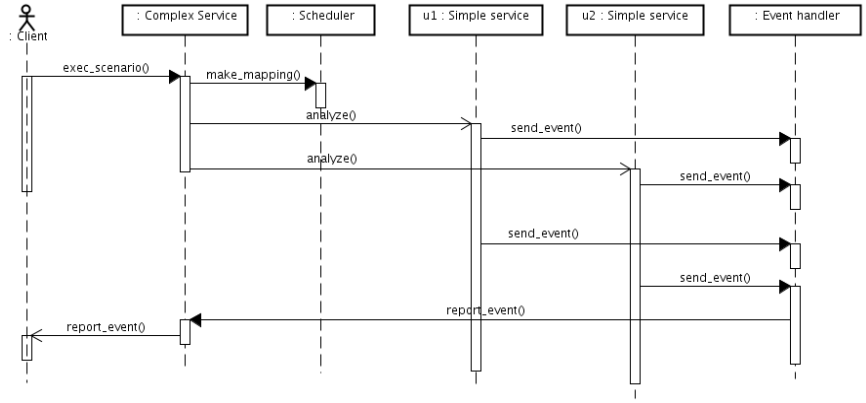


Fig. 17. A sequence diagram of the complex service execution within the domain model (see Fig. 16)

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 a camera, microphone, or other device (e.g. medical equipment), make the required analysis and transformation and send an output data stream including discovered events to the proper components, mainly to Event Handler and Dispatcher, forwarding them through the service layers to the clients – a users or an external applications (see Fig. 19).

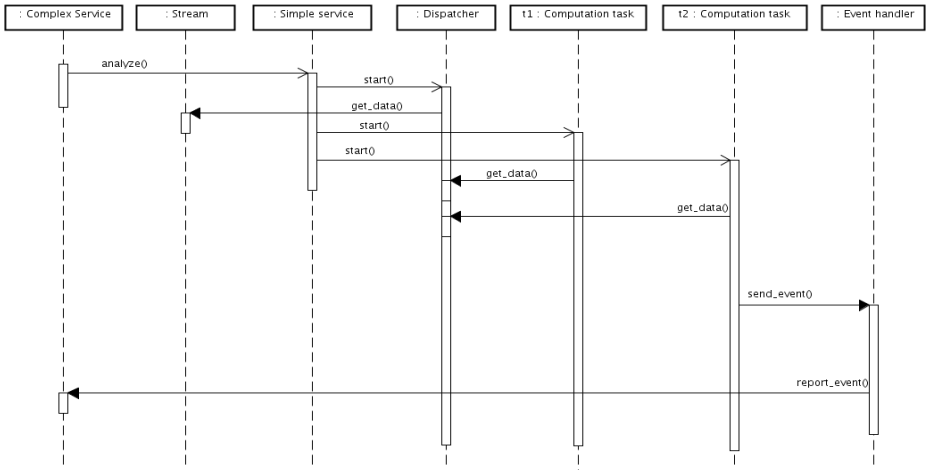


Fig. 18. A sequence diagram of the simple service execution within the domain model (see Fig. 16)

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

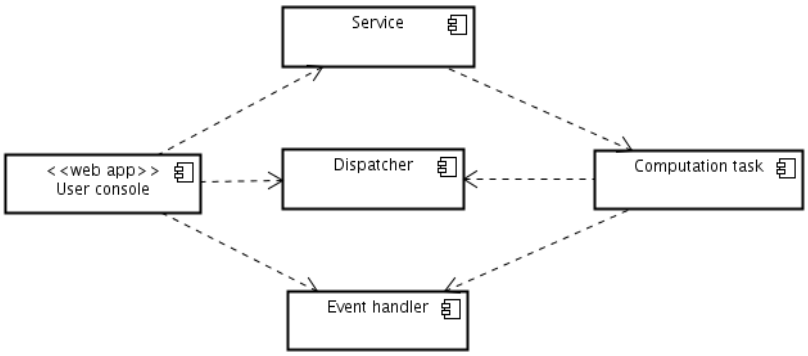


Fig. 19. Component diagram of the KASKADA platform

Almost all the above domain classes presented in Fig. 16 can be straightforwardly converted into the software components of the proposed platform. The only exception is the User Console component which 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 (a service repository with the WSDL and UDDI support).

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

6.2 Hardware architecture

To execute computation tasks all software components should be deployed on the computer cluster. Fig. 20 presents the deployment diagram including hardware nodes with the assigned software components. The core of the platform is the cluster which consists of 672 two-processor (Intel) nodes connected by the fast Infiniband network, each processor has 8 cores, which gives in total 5376 cores.

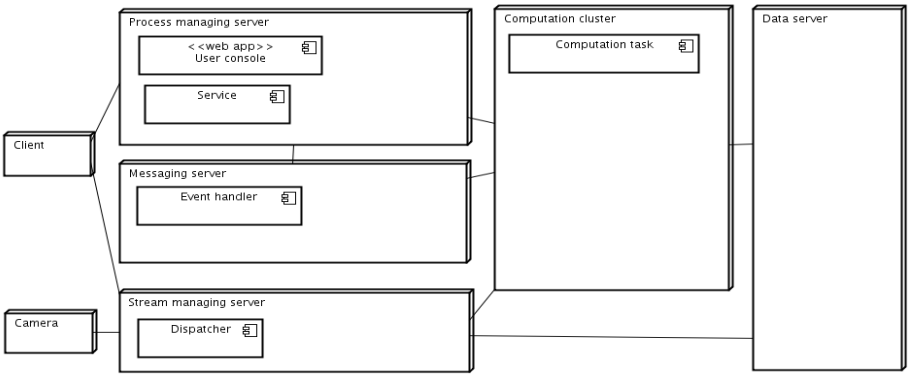


Fig. 20. Deployment diagram of KASKADA platform

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 multicast 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 use high performance hard drives with 500TB capacity and the Lustre file system, the server is going to be connected to the cluster and other servers by the Infiniband network, for its low delay and high bandwidth.

7. Applications available in KASKADA platform

The KASKADA platform was heavily tested for two pilot applications, developed as a proof of concept of the proposed multimedia processing model: identification of dangerous objects and unusual situations (DOUS) occurring in multimedia streams coming from cameras located in different places, and medical recommender for endoscopy examinations (MREE) showing some disease changes in film taken during gastroscopy track. The former is used for automation of typical video and audio surveillance tasks, like dangerous person identification or left luggage detection. The latter helps medical staff to quickly find the possible lesions in the video recorded during endoscopy examinations. Table 1 presents the usage of the platform features by the applications.

Functional feature	DOUS	MREE
video processing	yes	yes
audio processing	yes	no
real-time analysis	yes	no
streaming server	yes	yes
simple services	yes	yes
complex services with scenarios	yes	yes
test streams	yes	yes

Table 1. The main KASKADA platform functional features available for the applications

Quality characteristics are one of the most important features evaluating the whole platform behavior. During the platform analysis and design, we found out we need to focus on four key factors: performance, scalability, dependability and security. Performance is crucial for real-time processing, both processor speed and network bandwidth need to be examined, for the platform to work in the real environment. The scalability is another important factor, especially when large numbers of the input streams need to be archived, processed, and transmitted back to the customer. Moreover, even the high performance solutions can't provide the satisfactory results until they are dependable (the designed algorithms are correct and acceptable for various conditions), and finally the character of the processed streams, including surveillance and medical data, requires high security protecting access from unauthorized persons. The above applications, especially DOUS, can achieve the appropriate public security level using strategy illustrated in Fig. 21.

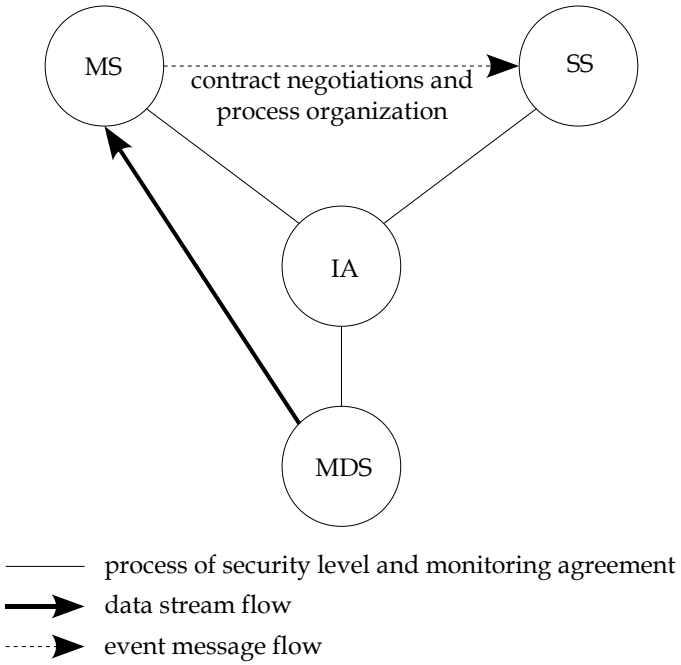


Fig. 21. Public security agreement strategy, MS – monitoring systems, SS – service services, IA – intermediary agency to provide SLA (service level agreement), MDS – multiple data sources

8. Conclusion

The KASKADA platform is designed to cooperate with external systems and applications. We decided to use a typical approach based on the SOA architecture with support for both synchronous and asynchronous communication, and implement a typical HTTP/SOAP protocol to start the exposed functionality, and message-passing queue system, supporting JMS and XMPP protocols to return the results of the long-lasting processing.

The platform is developed using agile development process principles, so the possible updates and new features can be added quite quickly, from one iteration to another, with release of a new version. The platform itself is flexible with easy and extensive configuration enabling quick adaptation to a new environment, or volume of the problem. With possible virtualization and cloud computing support.

The KASKADA platform is already developed and deployed in the Academic Computer Center of Gdansk University of Technology in Poland. The current development is focused on the quality tests related mostly to the performance and scalability characteristics. The whole project, including the proposed applications, is going to be finalized in 2012.

In the future we plan to continue development of new applications based on the currently available services, as well as new algorithms for the various multimedia processing problems, with a special focus on massive stream processing and various quality parameters, like dependability and security.

9. Acknowledgment

The work was realized as a part of MAYDAY EURO 2012 project, Operational Program Innovative Economy 2007-2013, Priority 2 "Infrastructure area R&D".

10. References

Breuer D., Erwin D., Mallmann D., Menday R., Romberg M., Sander V., Schuller B., Wieder P. (2004) Scientific Computing with UNICORE, NIC Symposium, ISBN: 3-00-012372-5, Forschungszentrum Jülich, February 2004

El-Rewini H., Lewis T. G., Ali H. H. (1994) Task Scheduling in Parallel and Distributed Systems, Prentice-Hall Series In Innovative Technology, ISBN: 978-013-0992-35-2

Foster I., Kesselman C. (1997) Globus: A metacomputing infrastructure toolkit, International Journal of Supercomputer Applications, Vol. 11, No. 2, (June 1997) pp. 115-128, ISSN: 1094-3420

Garey M. R., Johnson D. S. (1979) Computer and Intractability: A guide to the Theory of NP-Completeness, W. H. Freeman, ISBN: 978-071-6710-45-5

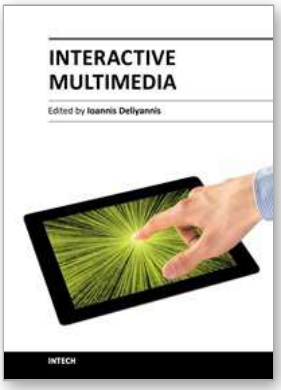
Haouari M., Serairi M. (2009) Heuristics for the variable sized bin-packing problem, Computers & Operational Research, Vol. 36, No. 10, (October 2009), pp. 2877-2884, ISSN: 0305-0548

Krawczyk H., Proficz J. (2010) The task graph assignment for KASKADA platform, Proceedings of International Conference on Software and Data Technologies, ISBN: 978-989-8425-22-5, Greece Athens, July 2010

Krawczyk H., Proficz J. (2010b) KASKADA – multimedia processing platform architecture, Proceedings of Signal Processing and Multimedia Applications, ISBN: 978-989-8425-19-5, Greece Athens, July 2010

Olken F., Gruenwald L. (2008) Data Stream Management: Aggregation, Classification, Modeling, and Operator Placement, IEEE Internet Computing, Vol. 12, No. 6, (November 2008), pp. 9-12, ISSN: 1089-7801

Yu T., Zhou B., Li Q., Liu R., Wang W., Chang Ch. (2009) The Design of Distributed Real-time Video Analytic System, Proceedings of CloudDB'09, ISBN: 978-160-5588-02-5, China, Hong Kong, November 2009



Interactive Multimedia

Edited by Dr Ioannis Deliyannis

ISBN 978-953-51-0224-3
Hard cover, 312 pages
Publisher InTech
Published online 07, March, 2012
Published in print edition March, 2012

Interactive multimedia is clearly a field of fundamental research, social, educational and economical importance, as it combines multiple disciplines for the development of multimedia systems that are capable to sense the environment and dynamically process, edit, adjust or generate new content. For this purpose, ideas, theories, methodologies and inventions are combined in order to form novel applications and systems. This book presents novel scientific research, proven methodologies and interdisciplinary case studies that exhibit advances under Interfaces and Interaction, Interactive Multimedia Learning, Teaching and Competence Diagnosis Systems, Interactive TV, Film and Multimedia Production and Video Processing. The chapters selected for this volume offer new perspectives in terms of strategies, tested practices and solutions that, beyond describing the state-of-the-art, may be utilised as a solid basis for the development of new interactive systems and applications.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Henryk Krawczyk and Jerzy Proficz (2012). Real-Time Multimedia Stream Data Processing in a Supercomputer Environment, Interactive Multimedia, Dr Ioannis Deliyannis (Ed.), ISBN: 978-953-51-0224-3, InTech, Available from: http://www.intechopen.com/books/interactive-multimedia/real-time-multimedia-stream-data-processing-in-a-supercomputer-environment

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821