A Model-Driven Solution for Development of Multimedia Stream Processing Applications

ANNA BOBKOWSKA, MICHAŁ NYKIEL, JERZY PROFICZ
Faculty of Electronics, Telecommunications and Informatics
Gdansk University of Technology
Narutowicza 11/12, 80-233 Gdańsk
POLAND
annab@eti.pg.gda.pl, mnykiel@task.gda.pl, j.proficz@task.gda.pl, http://www.kaskada.pg.gda.pl

Abstract: This paper presents results of action research related to model-driven solutions in the area of multimedia stream processing. The practical problem to be solved was the need to support application developers who make their multimedia stream processing applications in a supercomputer environment. The solution consists of a domain-specific visual language for composing complex services from simple services called Multimedia Stream Processing Modeling Language (MSP-ML) and a tool integrated with the supercomputer environment which makes round-trip transformations between models and executable service structures. The contribution to the debate about optimal model-driven technology can be stated as follows. Effective application of modeling requires model-driven solutions characterized by a high level of integration of models and tools in their context of use including user goals and activities as well as target environment. Such solutions sometimes require combination of several available approaches, i.e. domain-specific modeling, use of standards and modeling tools embedded in the platform.

Key-Words: model-driven solution, domain-specific language, modeling tool integrated with supercomputer environment, services, multimedia stream processing.

1 Introduction

The goal of this paper is to present the results of action research related to model-driven solution for the development of applications which perform multimedia stream processing in supercomputer environment.

1.1 Idea of Model-Driven Solution

In model-driven engineering (MDE) approach, models direct the course of entire software development and evolution. Several standards, e.g. OMG Unified Modeling Language (UML) [10] or OMG Business Process Model and Notation (BPMN)[8], provide a common language for communication among software developers and allow for research on more advanced issues of MDE. It has been argued that more extensive use of models and advanced tools makes modeling technology more mature [12]. Other research results suggest that actual benefits from models can be achieved with the application of domain-specific modeling (DSM) [6] with the application of meta-case tools.

We argue that effective application of MDE requires a model-driven solution (MDS) which is customized exactly to the context of application development. It should be developed with a systemic approach and all its elements must smoothly interact. The context of MDS includes user goals and activities as well as the platform on which applications are developed and executed.

Depending on the requirements to MDS, standard modeling languages or domain-specific languages are more appropriate. For example, for communicating analysis results to a wide audience of stakeholders one should rather choose a standard language. But for support of design with full code generation in a given domain, a domain specific modeling might be a more suitable approach. When selecting the DSM approach, use of meta-case tools seems a natural consequence. However, in some cases development of specialized tools integrated with the target environment is an even better choice. What is interesting, MDS sometimes requires combination of all abovementioned approaches. The MDS described in this paper provides arguments and illustration for this statement.

1.2 Discussion of Model-Driven Solution to the Problem at Hand

The practical part of the action research was driven by the needs of developers of multimedia stream processing applications. They needed a technique which could elevate the level of abstraction they were working on as well as a tool fully integrated with the environment which allows for automatic generation of executable structures.

The first part of the delivered solution is a domain-specific visual language which fulfils the requirement of elevating the level of abstraction on which application developers operate and which fits exactly to the context of application development. The language was defined with the use of standard OMG Meta-Object Facility (MOF) [9]. The second part of the solution is a tool integrated with the environment which allows for modeling as well as round-trip transformations between models and executable service structures.

One could ask questions: Why this solutions is the most appropriate in this case? Why standard modeling languages are not suitable? Why research results related to service-oriented architecture or multimedia stream processing were not applied? And assuming the need of DSM, why the most popular MetaEdit+ tool [5] has not been used? The standard modeling languages, such as UML or BPMN, are inappropriate for this application as they were designed to model sequential processing of discrete data, e.g. objects or messages. Multimedia streams, which are processed by supercomputers as parallel tasks in continuous and pipeline style, are different from this type of data. Thus, application of these languages would be a misconception. As we describe in section 6, results of related work on modeling services do not fit to the specifics of this application. On the basis of this analysis we have made decision to design a domain-specific visual language, called Multimedia Stream Processing Modeling Language (MSP-ML). As one of the requirements was strong integration with the supercomputer environment, the only reasonable solution was a tool integrated with this environment.

1.3 Structure of the Paper

The paper is structured as follows. Section 2 presents the context of the MDS including the specifics of supercomputer environment and specifics of multimedia stream processing as well as several roles of developers involved in application development and patterns of their interaction. Section 3 describes MSP-ML with OMG MOF standard. Section 4 contains an overview of the features of the tool called Complex Service Designer. Section 5 presents a case study of using this solution for face detection. Section 6 compares MSP-ML to related work and section 7 draws conclusions.

2 Context of the Model-Driven Solution

The context of use of this MDS includes the supercomputer environment, specifics of multimedia stream processing and the needs of developers of multimedia stream processing applications.

2.1 Supercomputer Environment

This project was implemented in Academic Computer Center TASK [1] placed in the Gdansk University of Technology. It uses a supercomputer called ‘Galera’. Its architecture is based on a computation cluster and utilizes 672 nodes with 8 computation cores each, providing in total 5376 cores and 50TFlops theoretical computational power. The nodes are interconnected using 20Gbps Infiniband, and they have access to a high performance mass storage with a capacity of 500TB, based on the Lustre file system. The computing center is connected with the Gdansk metropolis surveillance equipment including cameras and microphones in the university campus, train stations, sport stadiums and, in the near future, the airport. It is used by about thirty application developers who make their applications in areas of integrated video and audio data processing for security support, medical data processing and document analysis with the purpose of plagiarism detection.

2.2 Specifics of Multimedia Stream Processing

Multimedia stream processing requires an integrated hardware/software environment, where video and audio streams are forwarded, applications are deployed and analysis services have their execution environment. This environment is delivered by KASKADA platform, which is a middleware dedicated to support

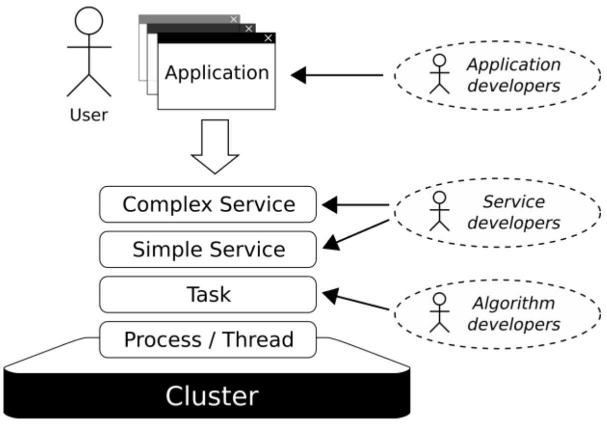


Fig 1. The model of the KASKADA platform architecture.

development and execution of multimedia stream processing applications [7]. As it is shown in Figure 1, its architecture consists of four layers: complex services, simple services, tasks and process/threads. Above this middleware, there are applications or users directly using the platform. The middleware uses hardware infrastructure of supercomputer (shown below the process/threads layer) which uses computational cluster nodes.

The highest layer of the KASKADA architecture operates with complex services which describe processing scenarios in terms of communication between simple services.

The second layer, called simple service, is responsible for managing basic functions with algorithm definitions and related metadata, i.e. execution and quality parameters, input/output types as well as computations characteristics such as expected CPU and memory load.

The third layer enables computational task execution. Tasks are executable programs which implements concrete stream processing algorithms. These programs are directly related to simple services. They run computations declared by the services when they are called.

The lowest layer is built directly on the operating system and uses processes and threads. The processes and threads perform all the computational task operations, cooperating with each other, utilizing shared memory and other parallelism mechanisms provided by cluster nodes.

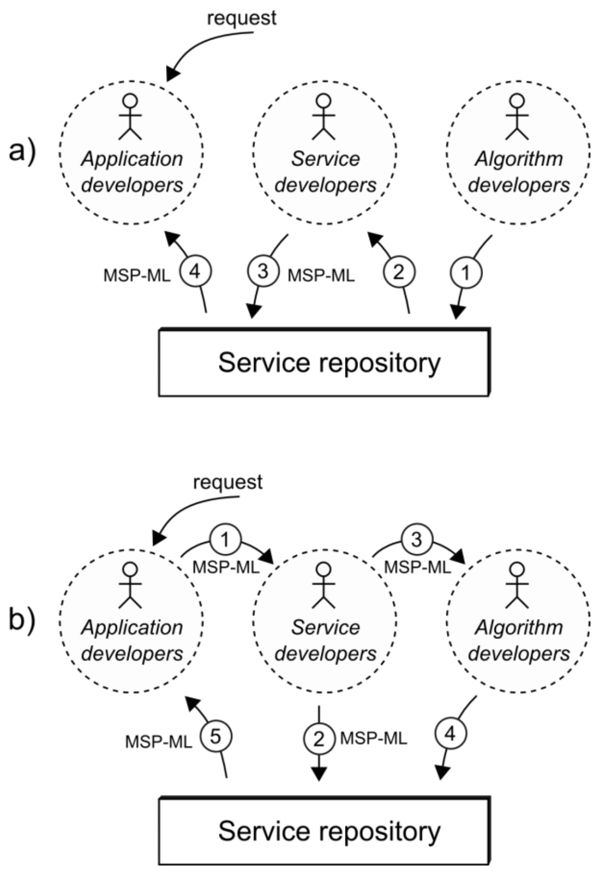


Fig 2. Interaction patterns: (a) reuse-driven development, (b) request-driven development.

2.3. Developers and Patterns of Interaction

The need for the solution has appeared in practice of application development. Developers expressed a request of a language for design of complex services and communication among their different roles. They had problems with editing long XML files which describe complex services. The problem was in assuring compliance to syntax and in memorizing the names of all elements and their attributes. Furthermore, problems with limited comprehension of the long XML files have caused annoying mistakes. The language together with an accompanying tool should eliminate these problems.

The following three roles of developers are involved in application development:

- Application developers, who perform software engineering activities including requirements elicitation, analysis and design as well as composition of services in concrete scenarios,
- Service developers, who are responsible for providing complex services, which are developed with the use of simple services,
- Algorithm developers, who develop the algorithms of simple services.

MSP-ML is used mainly by service developers for designing complex service structures from simple services. However, it is used also as a means of communication between several roles of developers regarding details of existing or requested complex services.

Two typical patterns of cooperation between different developer roles can be distinguished. The first one, reuse-driven development, shown in Figure 2a, assumes that algorithm developers provide implementations of reusable simple services and place them in a service repository. Then, service developers design reusable complex service structures using MSP-ML and they place them in the service repository as well. When application developers receive a request for an application from customer, they simply take services from the service repository and use them in their applications. The second pattern, request-driven development, shown in Figure 2b, assumes that not all complex services are available for the application requested by the customer. In this case, application developers communicate the need for complex services to the service developers using specifications as well as diagrams in MSP-ML. If these complex services can be developed on the bases of existing simple services, the service designers just make the composition using MSP-

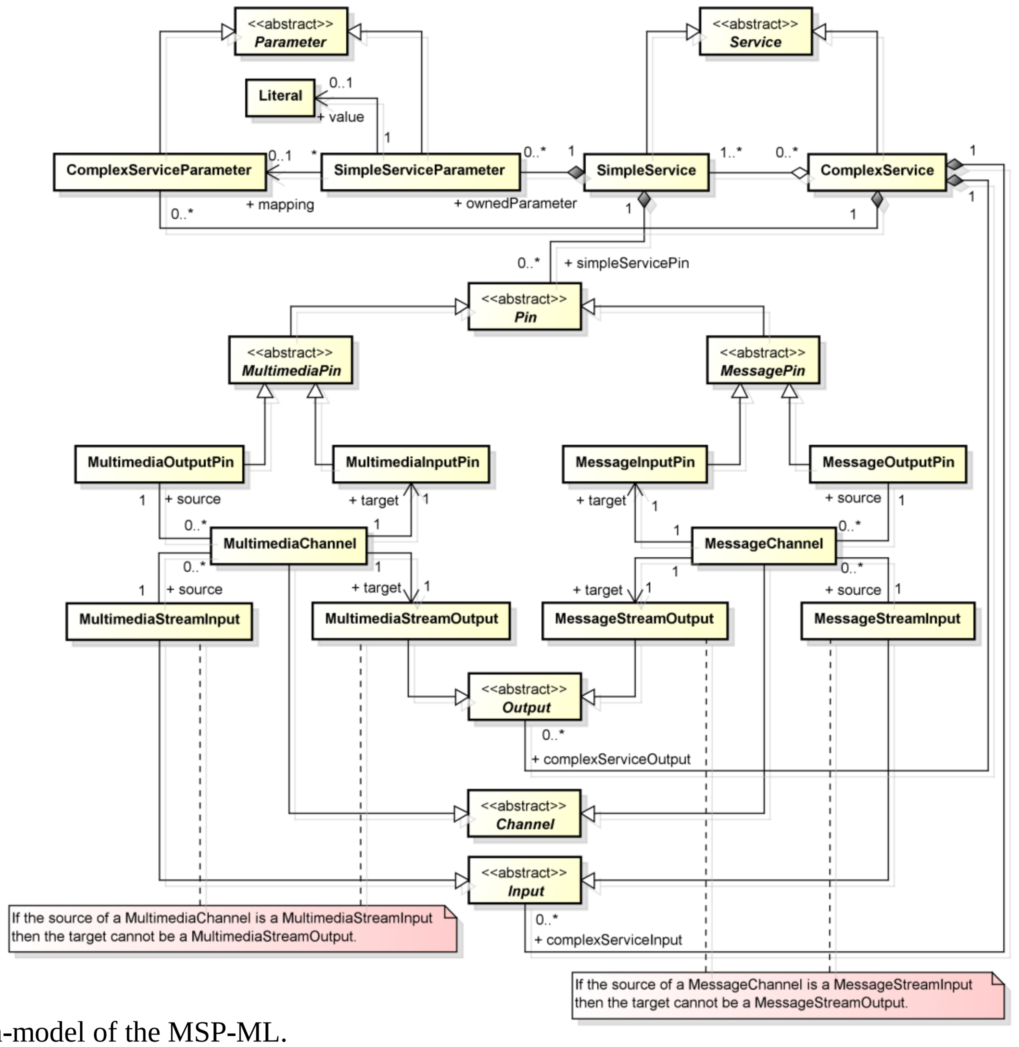


Fig 3. Meta-model of the MSP-ML.

ML and return the results to the application developers. However, if some simple services are missing, the request is sent to the algorithm developers, again using the specifications as well as the diagrams in MSP-ML. When requested simple services are placed in the service repository, the complex service structures can be tested and, finally, integrated by the application developers into the requested application.

3 MSP-ML description

The goal of MSP-ML construction was to provide a visual modeling language which allows application developers to elevate the level of abstraction they operate on when they compose complex services from simple services. The language should be precise enough to allow for automatic generation of executable service structures in XML format.

3.1 MSP-ML meta-model

The meta-model of the MSP-ML is presented in Figure 3. The basic elements of the model are: ComplexService, which is a container for services, inputs and outputs; and SimpleService, which aggregates parameters and pins. MultimediaPins and MessagePins are connected with each other through MultimediaChannels or MessageChannels.

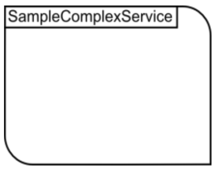
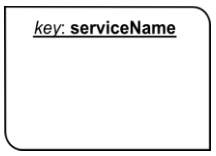
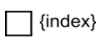
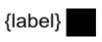
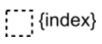
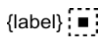
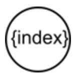
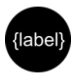
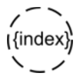
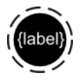
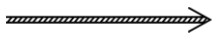
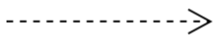
3.2 Notation and semantics

Descriptions of the model elements (meta-classes) in terms of their notation and semantics are presented in Table 1. The meta-classes for the MSP-ML reflect the requirements of understandability by application developers and precision necessary to transform them structures which can be executed by KASKADA platform. When designing MSP-ML we have used the following criteria in order to assure a high understandability by application developers [4]:

- conceptual expressiveness,
- visual expressiveness,
- consistency with the tradition of modeling.

In consequence, the following decisions regarding consistency were made. All elements related to messages are modeled with dashed lines, all input pins and input streams are modeled as empty elements of notation while output streams and output pins are modeled as filled elements of notation. The multimedia channels have the thicker icon as they represent transportation of heavy loaded streams in opposition to discrete and small messages. The language has gone through assessment in tests of XML file generation, evaluation from the perspective of cognitive dimensions and empirical studies with its users. They confirmed that the design decisions were optimal.

Table 1. Notation and semantics of MSP-ML meta-classes.

Meta-class name	**Notation**
ComplexService	
Complex Service is a container for all other kinds of element in the diagram. It can contain many Simple Services, inputs, outputs and channels between them.	
SimpleService	
Simple Service is a basic functional element of the complex service diagram. It is responsible for analysis and processing the data. The service definition contains the service name, a unique key (an identifier within a diagram) and a set of parameters.	
SimpleServiceParameter with Constant	{parameter} = {constant}
Represents a service parameter with a constant value. It can be either a quality parameter that determines which algorithm is used, or an execution parameter whose value is passed to the algorithm.	
SimpleServiceParameter	{parameter} ~ {mapping}
Represents a service parameter acquired by mapping. The value of Complex Service Parameter is assigned to the Simple Service Parameter.	
MultimediaInputPin	
Multimedia Input Pin represents an input of the Simple Service. It can be connected to a single multimedia stream. Every input pin is identified with a number related to the containing simple service.	
MultimediaOutputPin	
Multimedia Output Stream represents an output of the Simple Service. It generates a multimedia stream. Every output pin identified with a label related to the containing simple service.	
MessageInputPin	
Message Input Pin represents an input of the Simple Service. It can be connected to a single message stream. Every input pin is identified with a number related to the containing simple service.	
MessageOutputPin	
Message Output Stream represents an output of the Simple Service. It generates a message stream. Every output pin is identified with a label related to the containing simple service.	
MultimediaStreamInput	
Multimedia Stream Input is an element that represents a source of a multimedia stream, e.g. a video camera. It delivers multimedia streams to Simple Services. Stream Input is identified with a number.	
MultimediaStreamOutput	
Multimedia Stream Output is an element representing an output of the complex service. Multimedia streams which are connected to the Multimedia Stream Output will be available to customers.	
MessageStreamInput	
Message Stream Input is an element that represents a source of a message stream. It delivers message streams to Simple Services. Message Stream Input is identified with a number.	
MessageStreamOutput	
Message Stream Output is an element representing an output of the complex service. Messages which are sent to the Message Stream Output will be available to customers.	
MultimediaChannel	
Multimedia Channel represents a kind of channel, which transmits a single multimedia stream.	
MessageChannel	
Message Channel represents a kind of channel, which transmits a single message stream.	

3.3 Example of diagram

An example of the complex service diagram with the descriptions of model elements is presented in Figure 4. It consists of five simple services, two multimedia stream inputs and one multimedia stream output, one message stream input and one message stream output connected by several multimedia stream channels and message stream channels.

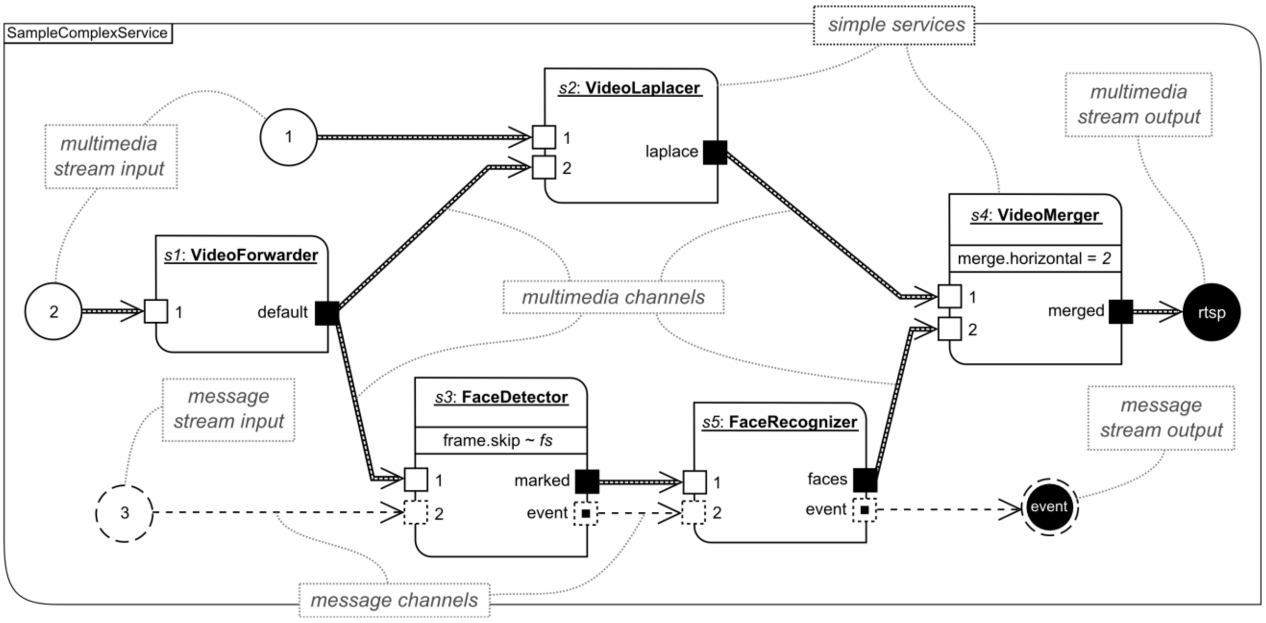


Fig 4. An example of a MSP-ML diagram with the descriptions of model elements.

4 Complex Service Designer

4.1 Overview

Complex Service Designer is a tool for modeling in MSP-ML which allows for round-trip transformations of MSP-ML models to executable service structures in XML format. The tool is fully integrated with the User Console, which is a web application used as a front-end for managing the platform. This approach to the service composition is much more user-friendly and efficient comparing to the necessity to edit XML files manually.

4.2 Technology and components

The tool is implemented in Microsoft Silverlight technology including framework for RIA (Rich Internet Application) development based on .NET platform and C# object-oriented language. It offers a good support for vector graphics, animations, interactivity and multimedia, which was very useful when implementing features related to modeling in MSP-ML. The architecture of the tool is shown in Figure 5.

Complex Service Designer is decomposed into four components:

- Modeler – a component responsible for modeling;
- DataProvider – a component used for communication between the Complex Service Designer and the service repository;
- Coverter – a component responsible for round-trip transformation between MSP-ML models and XML files;
- Validator – a component responsible fo validation of diagrams.

Communication between Complex Service Designer and the service repository is required on multiple stages of complex service scenario development. Simple services names and descriptions must be obtained from the repository. Additionally, Complex Service Designer requires detailed data about the selected services for the validation process, which checks parameters and input/output formats.

4.3 Round-trip transformation

Round-trip transformations are made by Converter component. It is based on mappings between MSP-ML meta-classes and XML constructs which are recognized and interpreted by KASKADA

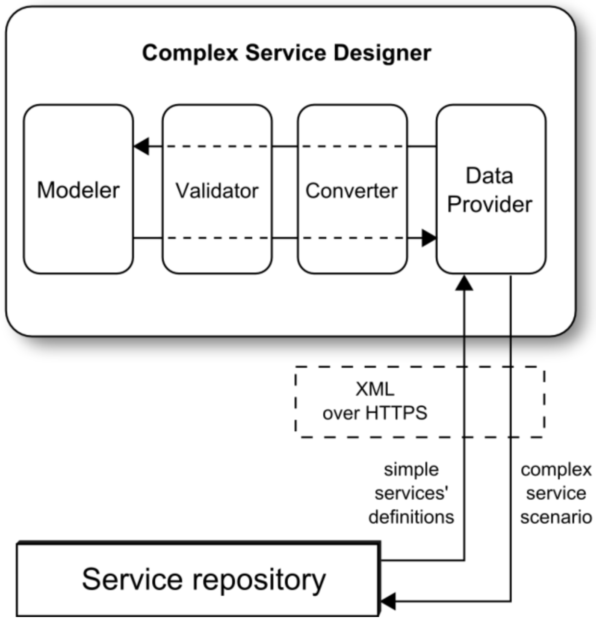


Fig 5. Complex Service Designer integration with the platform.

platform. The most typical use of the convertor is the forward way, i.e. from MSP-ML models to executable XML service structures. However, the tool supports also reverse transformation, i.e. from executable XML files to MSP-ML models.

Transformation from diagrams to the XML file is done in the following way:

1. Every Simple Service is converted to a `<simpleService/>` construct with the name and key attributes.
2. Parameters of each Simple Service are represented as `<parameter/>` elements which contain literal value or `<mapping/>` construct.
3. Each Multimedia/Message Stream Input is converted to an `<inputStream/>` element.
4. Multimedia Output Pins are transformed to `<outputStream/>` and Message Output Pins are converted to the `<event/>` construct within the `<simpleService/>`. The pin name is represented as the name attribute.
5. All of the Multimedia/Message Channels are converted to `<destination/>` element within `<outputStream/>` or `<inputStream/>` constructs, according to the channel source. The key attribute determines the target of the channel.
6. Every Multimedia/Message Stream Output is represented as a serviceOutput attribute of the associated `<outputStream/>`.

5 A Case Study

5.1 Problem of Face Detection

Let's assume that police requests an application for monitoring security of some area in a city. Surveillance devices, including PAL and HD cameras, are connected to a security center. However, in order to increase efficency, automation of the monitoring facilities is required. The application should deliver real-time face detection in the moving crowd as well as archive the processed data for further usage. The real-time face detection should allow for tracing suspected persons and support fast-reaction in case of crime.

5.2. Development process

As the request comes from the customer, the first task for an application developer is searching through the service repository for complex services performing this kind of multimedia stream processing. As no such complex service exists in the service repository, the application developer sends requests to service developers to deliver it. The service developers search for related simple services and composes a complex service using MSP-ML with Complex Service Designer. In this case, face detection requires decoding of incoming streams, algorithms of face detection with message sending, services of video images trimming and storing them on the disk and encoding the stream before sending it to the security center (output of the complex service). Currently, KASKADA service repository contains Codec Services, including H.264 and MPEG-2, which can be reused for decoding and encoding. Storing services also can be reused. The request for the missing Face Detect service is sent to algorithm developers. Thus, this is an example of the most complex case of request-driven pattern. When the service of Face Detect is delivered, the service developer can automatically generate executable service structure which can be used by the application developer. For testing purposes, such a service can be called directly, using the user console. However, the application developer can also use it in an application located at another application server. Such application usually requires a dedicated GUI, including alarms, additional business logic and deployment settings. An example of a deployment diagram is shown in Figure 6. It presents an application server with an application called Face Web Application, a managing server responsible for coordination of the complex service and a cluster which performs the actual processing. After configuration of cameras and processing parameters the application is ready to be used by the police.

5.3 Overview of application

Application overview is presented in Figure 7. Face Web Application requires input stream from surveillance camera to be transmitted to the supercomputer.

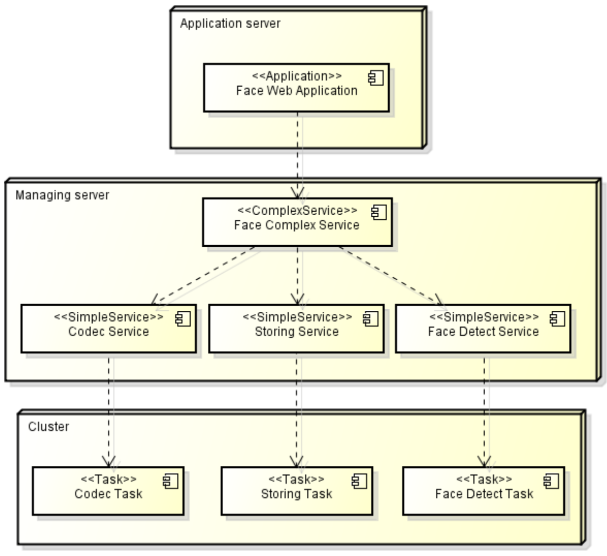
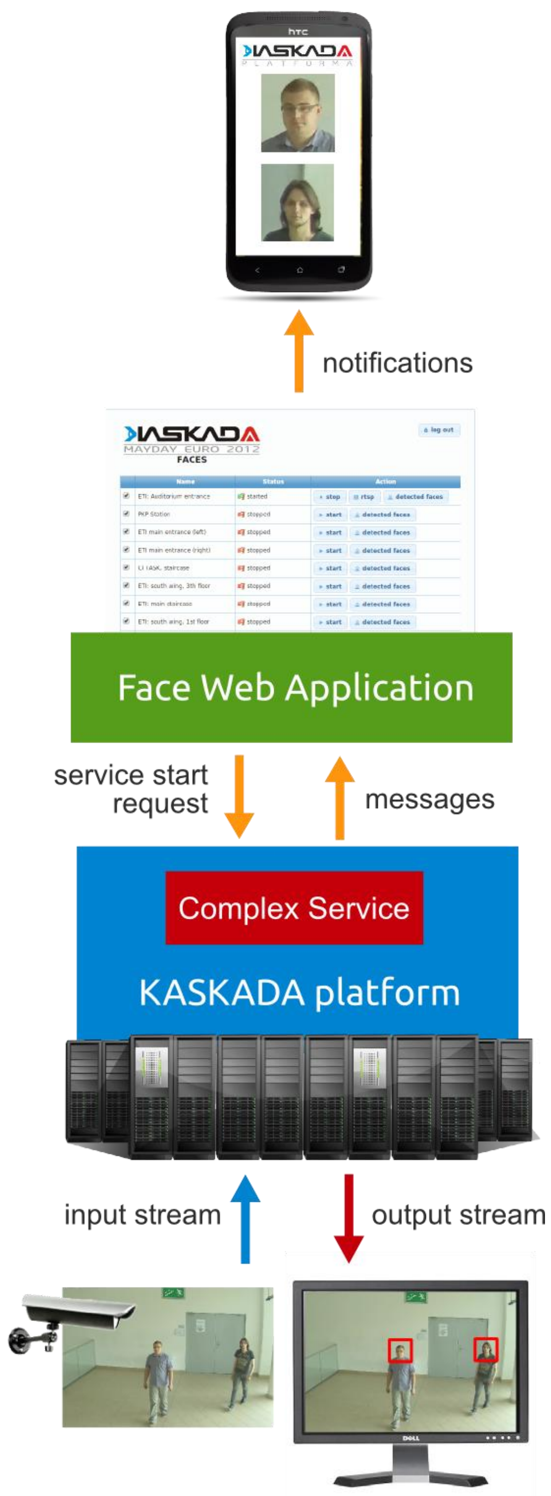


Fig 6. An example of the deployment diagram of the face detection.



	Name	Status	Action
✓	ETI: Auditorium entrance	started	stop rtsp detected faces
✓	PKP Station	stopped	start detected faces
✓	ETI main entrance (left)	stopped	start detected faces
✓	ETI main entrance (right)	stopped	start detected faces
✓	LI IASK, staircase	stopped	start detected faces
✓	ETI: south wing, 3th floor	stopped	start detected faces
✓	ETI: main staircase	stopped	start detected faces
✓	ETI: south wing, 1st floor	stopped	start detected faces

Fig 7. An overview of the face detector application.

When the user decides to begin face detection the application sends a request to the platform to start the complex service. KASKADA platform manages the entire lifecycle of the service. After receiving a request the repository is searched for necessary simple services. The next step is selection of algorithms that match requested quality criteria, such as precision or processing speed. Afterwards, streams that have been selected by the user for the analysis are retrieved from the data repository. Required multimedia and message stream channels between the services are also created in this step. After necessary validation of data formats and parameter values, the computation tasks are started. With continuous monitoring of cluster CPU, memory and network load, the platform is able to distribute new tasks on nodes optimally and whole analysis process may run in real time alongside other services.

The complex service generates an output stream where detected faces are marked with red rectangle. Furthermore, the service is able to send a message with stored image to the application, which can be forwarded to a mobile device. This feature allows police officer to be aware of current situation even when he has no access to a computer. Multimedia processing service delivered by KASKADA platform is efficient and successfully performs real-time face detection on streams from surveillance cameras.

6 Related work

In literature, one can find three main approaches to the problem of modeling stream processing services. The first approach concentrates on the service-oriented architecture of such systems. Related work includes Service-oriented Architecture Modeling Language (SoaML) [11], which is a well-known standard for modeling SOA systems based on UML. It was developed by Object Management Group as a UML profile because pure OMG UML [10] does not provide proper methods for identifying service providers, consumers and relations between them. This language is suitable for designing most of service-oriented systems, but it does not support modeling interfaces other than data flow between services, which is necessary in multimedia stream processing applications.

Another solution for modeling SOA systems is Service-Oriented Modeling Framework [3], designed by Michael Bell for managing the life cycle of service-oriented systems. It introduces five major modeling activities; one of them is logical design modeling which is focused on establishing relationships between services and data exchange paths. Although this language seems to fit better to our purpose than SoaML in terms of modeling the data flow, it is still not suitable for designing multimedia processing services. The main problem is a high level of abstraction, to general concepts of service inputs and outputs and a lack of differentiation between data and message streams.

The second approach is the use of languages designed for modeling processing scenarios. The most popular language in this category is OMG

Business Process Model and Notation (BPMN) [8], which is widely used for modeling various business processes. The problem with adapting this language in our context of application was related to its high level of abstraction and the missing concept of multimedia stream flow. Without proper elements for modeling input and output streams or service parameters, it is impossible to create an accurate description of a multimedia processing scenario.

The third approach to modeling multimedia processing services is focused on multimedia processing. A good example of such language is 4MPS (Metamodel for Multimedia Processing Systems) [2], which was proposed as a solution for describing systems with any multimedia processing design. The meta-model offers high-level semantics for the domain and introduces some abstract concepts such as processing objects and signal flow, which can be interpreted as processing services and multimedia streams in our platform. However, 4MPS have not allowed for modeling of communication patterns needed in our application, the most important of which is message passing between services.

To sum up, none of the discussed solutions has fit well to our context of multimedia stream processing applications. Furthermore, attempts to apply and extend them, would require to implement a number of useless constructs while the usefull constructs must have been complicated in some cases.

MSP-ML together with Complex Service Designer is a best-fit solution in this context. It constitutes a general contribution to the field of multimedia stream processing applications for the following reasons. It provides a successful combination of two originally separated fields of service-oriented architecture and multimedia stream processing. It was developed with solid methodology of model-driven approach, i.e. combination of DSM, meta-model in OMG MOF and a tool integrated with the execution environment. It delivers a complete model-driven solution to the needs of developers of multimedia stream processing applications. As such, it might be of interest for other solution providers in the area of multimedia stream processing applications.

7 Conclusions

This paper has presented the results of action research related to model-driven solutions for development of multimedia stream processing applications in supercomputer environment. The problem domain is appropriate for such action research because it is innovative and common solutions cannot be directly applied. In conclusion, we would like to stress the following:

- Need to focus on the context in terms of user goals and activities as well as environment - When one develops a model-driven solution, one should focus on needs in the context rather than available MDE technology;
- The design of the MSP-ML with the use of OMG MOF as well as cognitive dimensions and empirical studies with the users - MSP-ML is a kind of domain-specific visual language which contains only these categories which are needed in this context of use. The meta-model of this language is made with OMG MOF standard which should result in easy extensions of the language when needed. MSP-ML is precise enough to be the basis for round-trip transformations as well as for communication among developers of multimedia stream processing applications;
- Automation of round-trip transformations delivered by Complex Service Designer which is embedded in the supercomputer environment;
- Unique combination of several approaches related to modeling: standards, domain-specific modeling and tools integrated with the platform, which appeared to be the right solution to the real needs of application developers;
- Practical application of this model-driven solution - The solution is used by about thirty application developers and it satisfies its goals;
- Contribution to techniques of multimedia stream processing applications - As this solution proved to be useful in this specific context of development of multimedia stream processing applications in supercomputer environment, some elements of the solution, e.g. approach or MSP-ML, might be of interest to solution providers of other multimedia stream processing applications;
- Contribution to the debate on effective and efficient use of model-driven engineering - Several distinctive approaches in MDE were proposed and decision makers sometimes wonder which of them should be applied. This paper argues for treatment of MDE approaches as a kind of patterns. They appear to be complementary and they can be combined in the development of model-driven solutions.

Acknowledgments:
The work was done as a part of MAYDAY EURO 2012 project, Operational Program Innovative Economy 2007-2013, Priority 2 “Infrastructure area R&D”. The platform is currently developed as a part of the Center of Excellence in Scientific

Application Development Infrastructure "NIWA".

References:

[1] *Academic Computer Centre at Gdańsk University of Technology*, www.task.gda.pl (access on 5.03.2014)

[2] Amatriain X., *A Domain-Specific Metamodel for Multimedia Processing Systems*, IEEE Transactions On Multimedia, Vol. 9, No. 6, October 2007

[3] Bell M., *Service-Oriented Modeling: Service Analysis, Design, and Architecture*. John Wiley and Sons, 2008.

[4] Bobkowska A., Nykiel M., Proficz J., *Evaluation of Multimedia Stream Processing Modeling Language from the Perspective of Cognitive Dimensions*, Proceedings of PPIG 2011 - Psychology in Programming Interest Group Annual Conference, York, UK, 2011

[5] *Domain Specific Modeling with Meta-Edit+*, www metacase.com (access on 10.06.2011)

[6] Kelly S., Tolvanen J-P., *Domain-Specific Modeling: Enabling Full Code Generation*, John Wiley &Sons, 2008.

[7] Krawczyk H., Proficz J., *KASKADA – multimedia processing platform architecture*, In: SIGMAP 2010, Proceedings of the International Conference on Signal Processing and Multimedia Applications, 2010.

[8] Object Management Group, *OMG Business Process Model and Notation*, www.omg.org (access on 5.03.2014)

[9] Object Management Group, *OMG Meta-Object Facility*, www.omg.org (access on 5.03.2014)

[10] Object Management Group, *OMG Unified Modeling Language*, www.omg.org (access on 5.03.2014)

[11] Object Management Group, *Service oriented architecture Modeling Language (SoaML)* - Specification for the UML Profile and Metamodel for Services (UPMS), 2008.

[12] Rios E., Bozheva T., Bediaga A., Guilloreau N., *MDD Maturity Model: A Roadmap for Introducing Model-Driven Development*, LNCS 4066, 2006.