# Universal Platform for Composite Data Stream Processing Services Management

Paweł Stelmach, Patryk Schauer, Adam Kokot, and Maciej Demkiewicz

Institute of Computer Science, Wrocław University of Technology,
Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland
{pawel.stelmach,patryk.schauer,adam.kokot,
maciej.demkiewicz}@pwr.wroc.pl

**Abstract.** Constant delivery of data and information through streaming methods is growing in popularity. Streaming is widely used for video and sensor data delivery, usually directly to the client. In this work we propose architecture of the platform for composition of several distributed streaming intermediaries, able to process the data stream on-line. Features of the platform consist of ability to create a composite stream on demand as well as update, delete or read its current state. Works on the platform are an on-going research effort and current work, presented in this work, focuses on separation of platform functionality into distributed software components for performance optimization. This distribution allows for optimizing each component's behaviour regarding its usage characteristics. As an effect the platform's streaming service management functionality is offered as a stateless service.

## 1 Introduction

With the rapid evolution of the Internet, which has taken place in the last decade, many ideas appeared and changed the way we create web applications. Even the concept of web application itself has replaced a simple and static web-accessible HTML document. Readers have evolved into contributors and web pages became web applications but another greatly influential idea came with the Service Oriented Architecture paradigm and brought a concept of services, already widely known in business, to computer sciences. The notion of remote object access precedes the current century but the increase in Internet adoption and quality forced web services into public awareness. Then came standards, like XML-RPC, followed by WS-* for SOAP-based web services and WSDL for their description. Web applications ceased to be monoliths and more and more services started to extend their functionality.

The Web Service standard is based on request and result behaviour, often requesting some operation to be executed. After that the service is expected to stop working, waiting for the next request. This seems natural; however, some use cases require a different kind of behaviour. New user needs for continuous access

W. Zamojski et al. (Eds.): *New Results in Dependability & Comput. Syst.*, AISC 224, pp. 399–407.
DOI: 10.1007/978-3-319-00945-2_36 

to ever-changing data, like video feeds or sensor data, require a different kind of service and put even more strain on the Internet transporting capabilities.

Streaming services can deliver audio/video surveillance, stock price tracing, sensor data ([7]) etc. With technology development both the size as well as the number of concurrent data streams has increased. On top of this we add middleware for on-line data stream processing ([1], [10]), changing video format or colour, calculating whether a patient had a heart attack (based on on-line heart rate monitoring) and more.

Distributed stream processing is becoming more popular, especially in domains like eHealth, rehabilitation and recreation fields where distributed measurement data acquisition and processing are indispensable ([11]). In case of such applications the processing time is crucial. Solution presented in [6] discusses the concept of distribution of processing services in a peer-to-peer computer networks in order to meet such requirements and, additionally, increase system availability and provide appropriate mechanisms for emergency handling. Also, data stream processing finds its use in computational science or meteorological applications, where there are multiple data sources and multiple recipients interested in the processed data stream ([8]).

From the architectural standpoint multiplication and distribution of data stream processing services in fact does not change much the already complex problem of data stream routing. Both data sources as well as endpoints could potentially be distributed and, assuming that processing services could be more atomic and composable ([12]), such approach brings more flexibility. Considering the geographical distribution of data sources and endpoints, distribution of data processing could be an optimal solution in many cases, especially if it had minimized the size of data being transferred over large distances at certain steps of the composed service. Individual data streams could be routed to subsequent processing services, where appropriate calculations would take place without braking the flow of the stream ([2], [9]). This approach allows for increasing performance, especially when streams should be processed in real-time ([1]) and each atomic service could be optimized for a specific processing task.

For this purpose we propose a platform called the ComSS Platform (abbreviation for COMposition of Streaming Services) that addresses the requirements mentioned above and offers management functionalities that allow for creating, reading the state, updating and stopping compositions of any data stream processing services, provided that they are compatible (that is each two services communicating have to be able to communicate via a common protocol).

## 2 Platform Description

### *2.1 Platform Goals*

The main goal of the platform is to manage data stream processing composite services (also called composite streaming services). Composite service is a collection

of several stream-processing services that typically operate on a single data stream. Each service transforms data (ranging from simple data manipulation – like determining average – to complex anomaly detection algorithms and more) and transfers it to the next service in a composition. For example a colour video stream can be transferred to a service that transforms it to a black and white video and then to a service that detects objects in a video (which could be more efficient with black and white video).

This simple example is based on two services in a series structure but composite services can be more complex: they can process multiple data streams and be described by complex workflows and not only series structures. Also, although composite service can merge two streams into a single data stream, this is not their sole purpose and composite streaming service (or composite data stream processing service) should not be confused with stream composition.

Provided the designer of streaming services would like to utilize the platform automated management capabilities, he can delegate all the tasks of assembling, disassembling or monitoring of such services to the platform, simply by using its basic web interface.

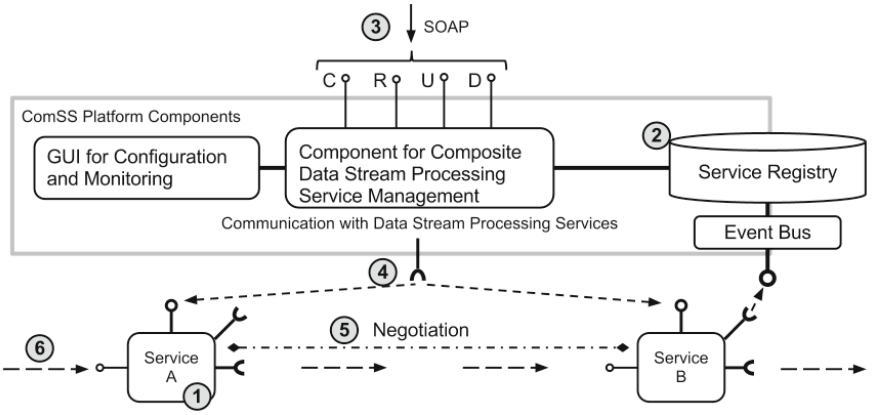


**Fig. 1** Overview of the Data Stream Processing Service Framework

In more detail, the ComSS Platform capabilities are mostly based on CRUD-based web interface, allowing for a specific composite service (treated as a resource) management; those are:

- creating a new composite streaming service based on a user request, containing a set of atomic services to be integrated,
- reading data about the current state of the composite streaming service execution,
- updating the composite streaming service, given new parameters for communication or replacing some of its atomic services,
- stopping and deleting the composite streaming service (not the atomic services themselves but their configuration and instances created for this composition).

Additionally, the platform is provided with internal components responsible for events distribution among its components and services, resources registration (especially atomic and composite streaming services) and communication with and among atomic streaming services during composite service control sequences (creation, reading, update, deleting).

## 2.2 Platform Overview

The ComSS Platform consists of several autonomous software components but also requires atomic streaming services (external to the platform, provided by the client) to implement specific communication libraries for control purposes. To facilitate implementation, those libraries are provided with a programming framework, which will be described in the next section.

In Fig. 1 separate boxes represent platform components: GUI, composite service management (e.g. service composition), communication with data stream services (negotiation supervisor), Event Bus and Service Registry and – outside of the platform – data stream processing services. We wanted to highlight particular service interfaces (line with a circle) and service calling capabilities (lines with unfinished circle) in some of those components when they are crucial to the lifecycle of the composite streaming service. Straight lines show that some components are interconnected but details about what are those internal connections and interfaces are out of scope of this work.

Components that are part of the platform are autonomous and communicate via before mentioned web services with SOAP protocol. Notice that the SOAP protocol is used only for control aspect of the platform, e.g., for communication protocol negotiation or starting and stopping of a composite streaming service. The streaming service itself is transferring data continuously through a data stream and uses a proprietary protocol that has been negotiated beforehand. In Fig. 1 only two streaming services are shown as an example of a simple composite service and the data stream is considered to come from an external source – not visible for the platform itself.

The platform itself delivers to its clients a simple web interface to create, read, update and delete a composite streaming service. Rest of the interfaces were designed for internal use; however, Service Registry and Event Bus could be provided from outside of the platform, allowing for sharing of resources and thus having their own external web interfaces.

The main assumption for the ComSS Platform was to focus on performing operations, which were fully defined using the CRUD-based web interface, exactly when they were needed and shutting down when they were completed. Such stateless behaviour has been introduced to minimize the chance of error during runtime and to shift the responsibility for service performance to the streaming services themselves. Following this concept was the decision to register composite services only for identification and logging purposes, so that the ComSS Platform could support the composite service only when requested, for instance the platform

reacts to errors reported by the streaming service. For that purpose the Event Bus, (the always-up component) is listening for events coming from atomic streaming services informing about errors or exceptions needed be handled, both their own or based on the unexpected behaviour of other services in the composition.

The basic scenario for the ComSS Platform is to create a new composite streaming service given a graph of atomic streaming services. It is assumed that those services have been implemented using the provided framework or implement necessary libraries (fig 1.1) and are registered in the service registry (Fig. 1.2). This step is not necessary but is useful for logging and identification purposes or when dynamically searching for service candidates if user request was not precise enough or requested services are not responding. For user *create* request (fig 1.3) the ComSS Platform searches for appropriate atomic streaming services and requests their participation in the composite service (Fig. 1.4). Using SOAP protocol the Platform informs each of the atomic streaming services about their immediate neighbours in the composition (e.g. next in a sequence), with which they will have to negotiate the communication protocol.

Streaming services start negotiating (Fig. 1.5) and create new service instances to handle the new composite service request. Each physical service is actually a server that can handle multiple requests and multiple streams – one atomic service can be a part of various composite services. To avoid confusion in the implementation, parts of the service that handle different streams are separated from each other by creating instances. Physically it is still one service, but for the purpose of identification in the system, each service instance can be treated as a separate service with different id, address and port etc. but with the same functionality and quality. Using the concept of service instances we can assure that each service (instance) is used only in one composite service and data streams are always correctly separated and transferred among services in a single composite service.

Finally, the ComSS Platform finishes its work and the streaming to the composite streaming service can start.

## 3 Streaming Service Framework

Part of the effort to make the streaming service composable lies with the service designer himself. He has to follow conventions for the streaming service design and implement necessary libraries for control, negotiation and communication. Using the framework provided with the ComSS Platform allows him to focus on implementation of the data stream processing algorithms alone (Fig. 2).

It should be noted that – in contrast to Web Services using the SOAP protocol – streaming services are not limited to a single protocol. The proposed framework allows for the use of any protocols for communication; however, it uses web services for control over the atomic streaming services. With web services the ComSS Platform conveys requests for creating new instances, reading their status or updating their parameters and, finally, deleting the appropriate instance.

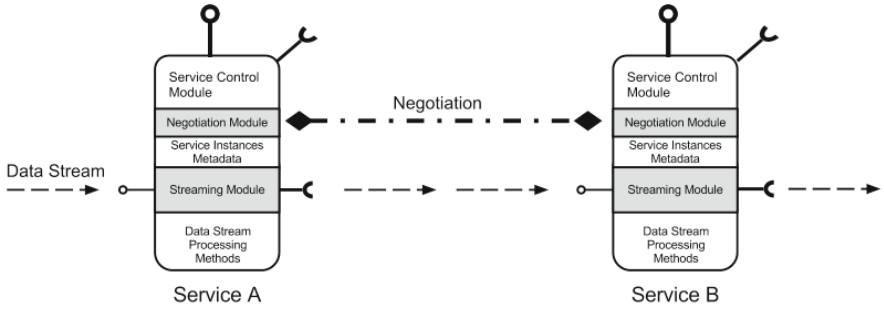


**Fig. 2** Overview of the Data Stream Processing Service Framework

The atomic streaming services also can send SOAP messages, mostly error or exception messages for the Event Bus to handle, but the designer can also implement other behaviours for scenario specific purpose – like email sending request etc. Some capabilities could be implemented in each streaming service, but they could be outsourced and shared via a web service, preferably using the Event Bus to execute specialized web services when necessary (which could implement various scenarios, like stopping or recomposing the composite streaming service).

In general, streaming services perform two types of communication.

SOAP-based communication, which is used for:

- sending a negotiation request (streaming service is delivered a list of neighbour services in a composite service with which it will start a communication protocol negotiation process),
- creating a new service instance (this is not to be confused with starting the data stream, because after creation each service is *ready* to receive, process and transfer the data stream and actually awaits for the stream to be transferred from external sources),
- control over the service when it is running (transferring and processing data stream) – reading its state, stopping it (deleting a service instance),
- standard error messages (events) propagation when services express unwanted behaviour,
- personalized messages (events) propagation, when service designer deliberately implemented such behaviour (sometimes sending a message via a Web Service is more natural then sending it via stream – e.g. when an anomaly is discovered then alert is a single event that should be directed to a specific recipient that can react to it, and not transferred with the stream).

Transferring the data stream:

- there are many different data stream formats and communication protocols and each service can support a selection of those solutions,
- it is the negotiation phase task to determine which protocol, format, port etc., will be used for the data stream transfer,

- the data stream is transferred independently of the control messages transfer (via Web Services) – usually the exchange of control messages takes place before the data is streamed or can end the stream transfer (requesting its deleting); during the data stream the Platform should be offline and only respond to errors or special events handled via the Event Bus.

In the basic scenario of creating a new composite streaming service, a request for a new service instance is sent to the streaming service via the web service interface (Fig. 2) in the negotiation phase. The framework communicates with neighbour streaming services, indicated by the ComSS Platform as next or previous in the composition. If the service confirms that it knows the requested protocol, gives an address and opens a port for communication, then a new atomic streaming service instance can be generated.

Then, when all services are ready to communicate, first messages are sent to the streaming interface of the streaming service. The role of the streaming module is to receive the data stream and prepare it for processing, which is de facto the function-providing part of the service. Next, data is prepared to be sent to the appropriate address and port of the next atomic streaming service in the composition.

## 4 Relation to Previous Works

Work presented in this work is part of an on-going research on a Universal Communication Platform ([3], [4], [5]). As a result of that research streaming services composition mechanisms were proposed in a prototype form. Our goal is to outsource those tasks to a specialized platform and extend its scope to end-to-end composite streaming service management. Compared to the earlier, prototype version a greater focus was put on performance of the service management layer as well as the performance and method of creating instances of data stream processing services. A complete separation of the service composition and usage – making them time and physically independent (by loose coupling of interfaces) – increased the flexibility and scalability of the system. The new version of the platform is stateless and mostly active only on demand, which results in better resource management. However, in effect the composite streaming service is not directly monitored during execution. With the ComSS Platform libraries implemented in streaming services, each service can report on its own state as well as on the state of neighbour services, especially if they behave contrary to the expectations, but the platform does not actively request those reports and if some services stop working it is up to their neighbours to report this behaviour.

Another major improvement lies in using external services in various management scenarios, mainly during emergency and unpredicted situations. Web services are used for recomposition or replanning purposes but it is planned to further develop this mechanism to use automatically composed reactions to emergency scenarios, based on user preferences, system policies and current situation.

## 5 Example of Platform Use

Consider a simplified example that some developer has built a streaming service that detects specific objects or people in a video stream. He has noticed that the algorithm performs better when it is provided with a black and white video. He could further develop his service so that it would change the video signal to black and white but two things stop him from doing that: one the service would be too specialized that he would want it to be, because perhaps someday he would want to analyse video in colour and, more importantly, he has found another service that already does what he needs. What he has to do is to somehow connect those services. But he cannot access the other services' code to statically implement his service address or communication protocol and in fact he should not do this.

Using the ComSS Platform he only has to point which services should be combined in a composite service and used in his specific scenario. The protocol negotiation, starting and stopping of those services will be handled by the platform with no need for additional code implementation. Also, with no direct addressing (and creating instances) those services can services multiple purposes and after this purpose is fulfilled no legacy code (statically connecting one service to another) remains.

The above example is simplified and intentionally omits matters concerning data transport through the network (which in case of video will introduce delay). In fact, with this platform and our on-going work on service composition and optimization such services could be discovered and composed dynamically and after automated consideration of various scenarios non-functional properties (service and transport costs, delay, etc.) an optimal solution can be selected and provided as a single service (treated as a black box).

## 6 Conclusions

This work presents a platform for distributed data stream processing services management. All its software components have been discussed from an architectural perspective, describing how the platform realizes its basic goals, which are lightweight and stateless composite services creation, monitoring, update and deletion. The framework for atomic streaming services enables service communication protocol negotiation and error reporting to the platform via the Event Bus.

**Acknowledgments.** The research presented in this work has been co-financed by the European Union as part of the European Social Fund and within the European Regional Development Fund programs no. POIG.01.01.02-00-045/09 \& POIG.01.03.01-00-008/08.

## References

[1] Chen, L., Reddy, K., Agrawal, G.: Gates: a grid-based middleware for processing distributed data streams. In: Proceedings of the 13th IEEE International Symposium on High performance Distributed Computing, pp. 192–201 (2004)

[2] Frossard, P., Verscheure, O., Venkatramani, C.: Signal processing challenges in distributed stream processing systems. In: Proceedings of the 2006 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2006, vol. 5, p. V (2006)

[3] Grzech, A., Juszczyszyn, K., Świątek, P., Mazurek, C., Sochan, A.: Applications of the future internet engineering project. In: 2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel Distributed Computing (SNPD), pp. 635–642 (2012)

[4] Grzech, A., Rygielski, P., Świątek, P.: Translations of service level agreement in systems based on service-oriented architectures. Cybernetics and Systems 41(8), 610–627 (2010)

[5] Grzech, A., Świątek, P., Rygielski, P.: Dynamic resources allocation for delivery of personalized services. In: Cellary, W., Estevez, E. (eds.) Software Services for e-World. IFIP AICT, vol. 341, pp. 17–28. Springer, Heidelberg (2010)

[6] Gu, X., Nahrstedt, K.: On composing stream applications in peer-to-peer environments. IEEE Trans. Parallel Distrib. Syst. 17(8), 824–837 (2006)

[7] Gu, X., Yu, P., Nahrstedt, K.: Optimal component composition for scalable stream processing. In: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, ICDCS 2005, pp. 773–782 (2005)

[8] Liu, Y., Vijayakumar, N., Plale, B.: Stream processing in data-driven computational science. In: 7th IEEE/ACM International Conference on Grid Computing, pp. 160–167 (2006)

[9] Rueda, C., Gertz, M., Ludascher, B., Hamann, B.: An extensible infrastructure for processing distributed geospatial data streams. In: 18th International Conference on Scientific and Statistical Database Management, pp. 285–290 (2006)

[10] Schmidt, S., Legler, T., Schaller, D., Lehner, W.: Real-time scheduling for data stream management systems. In: Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS 2005), pp. 167–176 (2005)

[11] Świątek, P., Klukowski, P., Brzostowski, K., Drapała, J.: Application of wearable smart system to support physical activity. In: Advances in Knowledge-based and Intelligent Information and Engineering Systems, pp. 1418–1427. IOS Press (2012)

[12] Świątek, P., Stelmach, P., Prusiewicz, A., Juszczyszyn, K.: Service composition in knowledge-based soa systems. New Generation Computing 30, 165–188 (2012)