Planning-Based Method for Communication Protocol Negotiation in a Composition of Data Stream Processing Services

Paweł Stelmach, Paweł Świątek, Łukasz Falas, Patryk Schauer, Adam Kokot, and Maciej Demkiewicz

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

Abstract. Data streaming is often used for video and sensor data delivery, nowadays gaining in popularity along with the development of mobile devices. In this paper we briefly describe a platform for automated composition of distributed data stream processing services. It decreases the complexity of composite service building process from the users point of view by introducing automation mainly in appropriate service selection and their communication protocols negotiation. This paper is focused on automated negotiation of communication protocols, which will be further used to transfer data among services. Data stream processing services are designed independently, often with many possible communication methods and not pointing directly to other services. With the use of the platform, they can be loosely coupled, forming a composite service od-demand. We present several approaches to communication protocol negotiation in a composition of data stream processing services and introduce planning-based approach. Finally, we discuss consequences of various approaches on an example composite service from image processing domain.

Keywords: Service Oriented Architecture, data stream processing services, service management.

1 Introduction

With the development of Internet more and more applications are available via Web. Also, they no longer have to be monoliths but developers outsource many functions to web services. This service orientation became more popular over the recent decade with the SOA paradigm (Service Oriented Architecture), which introduced a way to build large distributed systems. It uses standards, like WS-* for SOAP-based web services and WSDL for their description. However, the Web Service standard is based on request and result behaviour. After that the service is expected to stop working, waiting for the next request. This seems natural, however, the need for continuous access to ever-changing data, like video

A. Kwiecień, P. Gaj, and P. Stera (Eds.): CN 2013, CCIS 370, pp. 531–540, 2013.

feeds or sensor data, requires a different kind of service, one that puts more strain on the Internet transporting capabilities. Such streaming services can deliver audio/video surveillance, stock price tracing, sensor data [1], etc. With time we have introduced specialized middleware for data stream processing [2,3], changing video format or colour, calculating on-line whether a patient did not have a heart attack and more.

Distributed stream processing, adding more intermediary services forwarding the stream from one peer to another, is growing in popularity [2,4,5], especially in eHealth, rehabilitation and recreation fields where distributed measurement data acquisition is natural [6,12] or in computational science and meteorological applications, where there are multiple data sources and multiple recipients interested in the processed data stream [7].

In this paper we describe the ComSS Platform (COMposition of Streaming Services) that is a result of ongoing work in the area of Future Internet [8]. It offers management over 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). The communication negotiation is of extreme importance and is reflected in negotiation process that takes place during creation of a distributed data stream processing service. Many papers, which take on the subject of composition of services, often refer to services in the WS-* standard [9] or consider stream processing services [10,11], but still treat them similarly to WS-* service, omitting their unique characteristics (namely the flexibility in various formats and communication protocols employment).

2 Platform Description

2.1 Platform Overview

The main goal of the platform is to manage data stream processing composite services (also called streaming services). Provided the designer of such services would like to utilize the platform automated management capabilities, he can delegate the all tasks of assembling, disassembling or monitoring of such services to the platform simply by using its basic services.

The ComSS Platform consists of several autonomous software components but also requires atomic streaming services to implement specific communication libraries for control purposes (see Fig. 1 for reference).

The basic scenario for the ComSS Platform is to create a new composite streaming service given a graph of atomic services (composite service plan). It is assumed that those services have been implemented using the provided framework (Fig. 1.1) and are registered in the service registry (Fig. 1.2). The platform searches for appropriate atomic streaming services to fulfil the user composite service request and forwards them information on neighbour services (Fig. 1.4), 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.

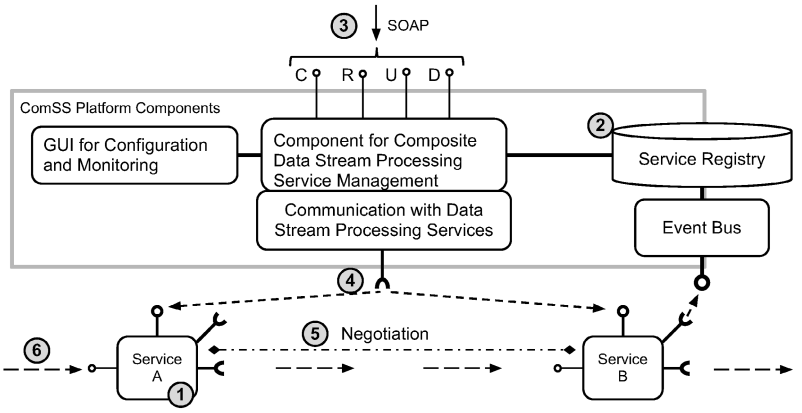

Fig. 1. ComSS Platform overview

2.2 Framework for Data Stream Processing Service

Part of the effort to make the streaming service composable lies with the service designer himself. He has to follow conventions for the 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).

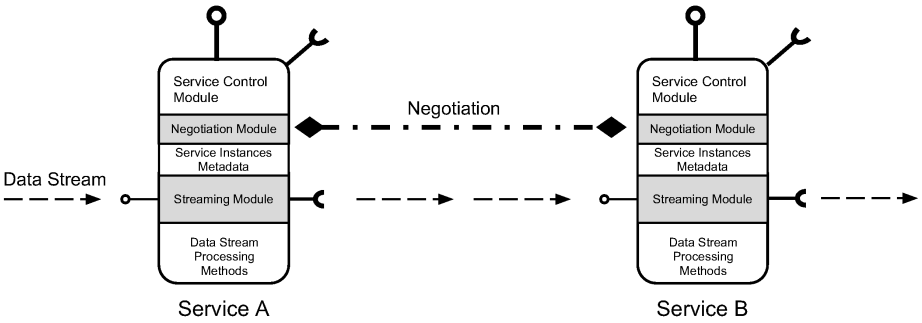

Fig. 2. Overview of the Data Stream Processing Service Framework

It should be noted that in contrast to the Web Services using the SOAP protocol, data stream processing services are not limited to one single protocol or format – most popular uses, and completely different protocols, are video stream processing and sensor data stream processing.

In the basic scenario of creating a new composite streaming service a request for a new service instance is sent via the Web Service interface (Fig. 2). Before

that, in the negotiation phase, the framework communicates with streaming services indicated by the ComSS Platform as neighbour services. If the service responds that it knows the requested protocol and 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. The role of the streaming module is to receive the data stream and prepare it for processing and send it to the next service afterwards.

2.3 Negotiation as a Part of Composite Service Creation Process

Modules responsible for negotiation are: Signalization Supervisor (in the ComSS Platform) and Service Signalization Module (part of Service Control Module of the streaming service framework). Former is a part of Platform Core and the latter is part of atomic service application. There are two types of communication inside the ComSS Platform: signalization (between core component and atomic services) and negotiation (between pairs of services). The composite service request is a fundament for these two types of communication. The service composition plan is used to establish which what services are neighbours in the sense of the composition and what negotiation requests to send to each of them. Secondly, some negotiation approaches may use the information on the order of services in the composition to maximize the chance for success. In our model negotiation information is distributed directly to services and then among pairs of neighbours. Services don't need to use core component to mediate between them. In distributed systems minimizing traffic through the central point it is often an important feature.

The composite service creation procedure is as follows:

- User forwards the request to the Create Interface.
- Request correctness is verified through Composite Service Graph validation.
- The Service Management Component (SMC) requests a list of descriptions of all atomic services in a composite service request.
- SCM Communication Module connects to each of the atomic services (signalling unit).
- SCM Communication Module forwards to each of the atomic services a request stating that their resources have to be reserved.
- The Service Control Module in each atomic service (AS) verifies the type of request and its correctness.
- After all Atomic Services confirm the reservation then SCM sends each of them a list of services they should start negotiating with.
- AS Service Control Module (sender) starts negotiations with recipient services, providing them a list of types and formats in which it is able to transmit (this list can vary depending on the received request, the list is sorted by priority).
- AS Service Control Module (receiver) selects the data type and format it can receive and sends back this information to the inquiring AS.

- AS Service Control Module (sender) confirms the determination of the type and format of the recipient and communicates the outcome of negotiations to the SCM communication module.
- SCM Communication Module collects the answers from the AS.
- SCM sends to the User a confirmation along with information on the newly established composite service.

3 Motivation

Author of the atomic streaming services is unaware of what composite services will be built using them. With the ComSS Platform he can register his services and their capabilities: location, type of processing if offers, what kind of input data it can process – what format, codec etc. Provided with this information, the ComSS Platform can automatically determine how to connect services in a composite service requested by the user.

In most papers on streaming services composition authors neglect the unique nature of streaming services, namely that they can accept a specific data stream in various forms: with different coding and format. Some services are capable of converting those formats (for a cost) while other cannot, and offer simple processing or monitoring capabilities on a range of formats, incapable of their modification. Using services with conversion capabilities offers more flexibility but also can increase processing costs over more simple versions of services. However, using only processing services can lead to bottlenecks in service compositions where two services cannot communicate due to incompatibility in the format of the data stream.

In this work it assumed that a service composition has been defined either automatically or by hand and now this service has to be physically initiated. In order to do this, appropriate resources have to be reserved, service instances created and communication among atomic services directed to other services ports, suitable of processing a given format.

A simple, greedy approach used to date can lead to rejection of service composition request due to incompatibility of formats among some atomic services. In the next section a planning-based approach is presented to guarantee that if there exists a feasible solution than a communication between atomic services can be established. What is more, this solution will be optimal due to both processing and communication cost.

4 Communication Negotiation

4.1 AI Planning in Negotiation

Typically AI Planning is used in service composition. In this situation all services and a sequence in which they are connected are already defined, but – as stated in the previous section – an incorrect choice of streamed data formats can lead to incompatibility of services in the sequence.

The proposed approach is in opposition to currently implemented greedy approach, where each atomic service can determine its own communication with another service. We suggest that formats and communication protocols were defined centrally in the ComSS Platform and then propagated to all services.

The approach we describe is based on backward search. Below you can see both the main body of the algorithm as well the recursive plan search procedure in a form of pseudocode. In this approach a general goal is defined based on user format requirements and while iterating through all services (starting from the last), a temporary goal format is defined, based on the input format of currently selected service. The algorithm utilizes information on the user input and output format requirements (F_{in},F_{out}) – which are based on the format of the external source data stream that will be transferred to the composite web service and the format of the goal of the data stream that the composite service will deliver the stream to. Finally, the algorithm is not a service composition algorithm and thus it is limited to the services and their order defined prior to the negotiation.

The algorithm gives the guarantee that appropriate formats, ensuring communication at each step of the data stream processing will be selected if possible and that the selection will minimize both the processing and transport cost.

- **Input**:
  F_{in} – format of the external input data stream,
  F_{out} – format of the data stream to be delivered on the output,
  AS – a set of atomic services, which communication formats have to be negotiated; those services comprise the composite.
- **Output**:
  Optimal plan determining communication formats for each of the services in a composite service.

1. $P = \oslash$
2. $AS \leftarrow sortServicesInIncreasingOrder$
3. $Plans = findPlans(P, F_{\text{out}}, F_{\text{in}}, AS)$
4. **if** $Plans = \oslash$ **then** "Communication cannot be established"
5. **else**
6. **foreach** $Plan$ **in** $Plans$
7. $Q = calculatePlanQuality(Plan)$
8. **if** $Q > Q_{\max}$ **then** $Q_{\max} = Q$ **and** $P_{\max} = Plan$
9. **end foreach**
10. "Optimal plan is $P_{\max}$ with quality $Q_{\max}$"
11. **end if**

The above algorithm comprises of two parts. Firstly, it initializes the search for valid plans that, with the services in AS, transform the input data stream with format F_{in} to the output data stream of format F_{out}. Secondly, it determines the quality of each of the plans, selecting the plan with the best quality. The quality is calculated based on the processing cost of each service and cost of transporting data stream between them. The former takes in to consideration that converting the format usually leads to increased processing cost and the latter takes into account that some formats can decrease the volume of data, thus decreasing the cost of transport of such data.

Algorithm: Plans Backwards Search

- **Input**:
  P – current plan to achieve the goal format F_{out},
  F_{current} – current goal format, for selecting appropriate input formats of a service,
  F_{in} – format of the input data stream,
  AS – the remaining web services, which formats have to be established.
- **Output**: Valid plans determining communication formats for each of the services in the current AS
  $Plans = findPlans(P, F_{\text{current}}, F_{\text{in}}, AS)$:

1. $as = AS.\textbf{last}; AS = AS \setminus as; Plans = \oslash$
2. $V = generateVersionsOf(as)$ such that each $v \in V : v_{\text{out}} = F_{\text{current}}$
3. **if** $V \neq \oslash$ **then foreach** $v \in V$
4. **if** $AS == \oslash$ **and** $v_{\text{in}} == F_{\text{in}}$ then $Plans := Plans \cup (P \cup v)$
5. **else if** $AS \neq \oslash$
6. $Plans^* = findPlans(P \cup v, F_{\text{current}} = v_{\text{in}}, F_{\text{in}}, AS)$
7. **if** $Plans^* \neq \oslash$ **then** $Plans := Plans \cup Plans^*$
8. **end if**
9. **end foreach**
10. **end if**
11. **return** $Plans$

The algorithm above takes the last atomic service in AS and will establish possible format transformations. A service as can be treated as a container for multiple abstract services that each can take one input format and generate one output format. Each of whose abstract versions of service as has its own processing cost. In (step 2) only those abstract versions of the service are generated which output format is equal to the current goal format F_{current}.

If there are versions that meet the requirements and there are still services left in AS (step 6) then each version is considered separately in an alternative plan searched recursively.

The procedure stops if no services are left in AS and returns a valid plan only if the last version can take as an input the format defined in F_{in}.

All valid plans are returned and compared in the main body of the algorithm.

4.2 Example and Comparison to Greedy Approach

The differences of various approaches will be briefly described using examples of negotiation procedure in those approaches and possible outcomes.

The current implementation does not enforce any order of negotiation requests thus it can provide unexpected results. Below we show two most popular approaches: forward and backward negotiation requests propagation and their outcomes, compared to the planning-based approach.

In Fig. 3 and Fig. 4 we observe that each service is provided with information about services it has to negotiate the communication format with. In contrast, in

Fig. 5 we see that requests are not typical negotiation requests but the planner enforces the format on each service (in any order). Additionally, both forward and backward negotiation are extended with initial pre-configuration phase in which appropriate services (first and last in the composition) are configured with information about requested input and output stream, thus preventing the situations when the service would choose a format incompatible with the request.

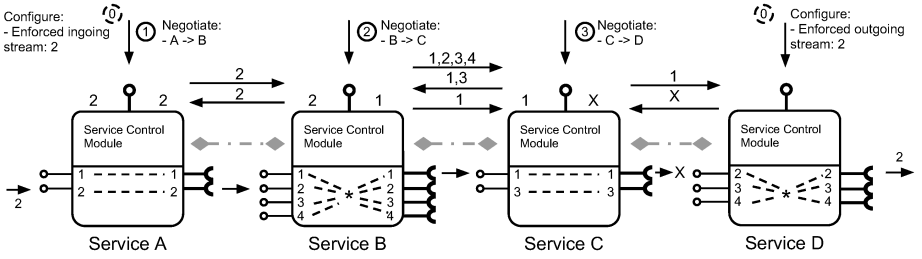

Fig. 3. Example of forward negotiation without planning

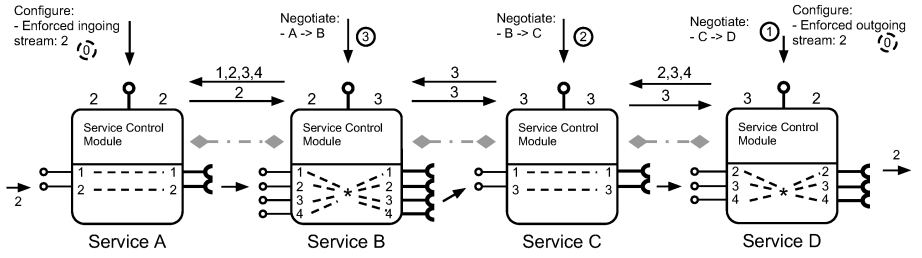

Fig. 4. Example of backward negotiation without planning

In Figure 3 we observe forward propagation of negotiation requests. Service A negotiates with service B. It sends formats in which it can communicate with service B. Notice that service A cannot transform the input format and following the requirement from external data source it limits its formats list. Service B responds with a format from its list that is on service A's list. Here, the negotiation is trivial but typically service B would select a format it "prefers" – usually higher in a static rank. Now service B has its input format set to "2", but considering that it is capable of converting it, it is not limiting its output format. Following the next negotiation request service B sends all its output formats to service C and service C responds with the same list limited to the formats in can process. Service B selects format and relays its decision to service C. Next,

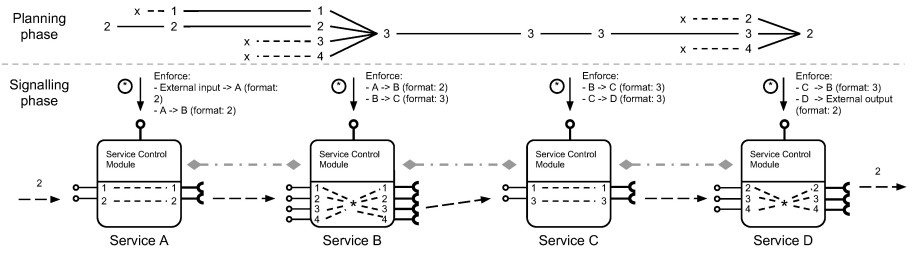

Fig. 5. Example of negotiation with planning

service C sends to service D a limited list of formats, which has no common format with service D and service D has to report an error. Please note that if a different format were negotiated between services B and C then the negotiation would end in success. However, neither service B nor C had any knowledge that could point the negotiation in that direction.

In Figure 4 we observe the inverse of the previous process. Following the initial configuration of the first and last service in the composition, the negotiation starts with service D. It relays its formats to service C and then service C to service B etc. The simple change in direction of negotiation resulted in negotiation success. However, were it not for the initial configuration of service A, services B and A would negotiate a different format ("1" had the priority) and the final verification with the external source would end in failure.

The planning-based approach in Fig. 5 allows for analysis of possible connections and enforces the communication for each service pair. Additionally, it allows for selection of optimal formats from the both processing and transport cost point of view. In this example there was only one valid solution, but if service D could process format "1" on the input then there would be two possible solutions: one transporting the data through services B, C and D in format "1" and another in format "3".

5 Conclusions and Further Work

Research presented in this paper describes the communication negotiation phase during construction of a composite data stream processing service. A ComSS Platform has been briefly described as an example of a tool that delivers such capability, relaying negotiation requests in accordance with the composite service structure.

In this work a planning-based approach to negotiation was presented. This approach extends basic capabilities of the ComSS Platform and limits negotiations in the atomic services. As a result, more calculations have to be performed centrally, but – as presented in examples – the proposed negotiation approach guarantees not only finding a valid result if one exists but also the solution is optimal with regard to data stream processing and transport time.

Future work will focus on testing all approaches on real streaming data and real processing services, showing typical cost of communication plans negotiated – both calculation cost and processing and transfers costs.

Acknowledgments. The research presented in this paper 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. Gu, X., Yu, P.S., Nahrstedt, K.: Optimal component composition for scalable stream processing. In: Proceedings of 25th IEEE International Conference on Distributed Computing Systems, ICDCS 2005, pp. 773–782 (June 2005)
2. Chen, L., Reddy, K., Agrawal, G.: Gates: a grid-based middleware for processing distributed data streams. In: Proceedings of 13th IEEE International Symposium on High performance Distributed Computing, 2004, pp. 192–201 (June 2004)
3. Schmidt, S., Legler, T., Schaller, D., Lehner, W.: Real-time scheduling for data stream management systems. In: Proceedings of 17th Euromicro Conference on Real-Time Systems (ECRTS 2005), pp. 167–176 (July 2005)
4. Gu, X., Nahrstedt, K.: On composing stream applications in peer-to-peer environments. IEEE Trans. Parallel Distrib. Syst. 17(8), 824–837 (2006)
5. 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, 2006, pp. 285–290 (2006)
6. 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)
7. 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 (September 2006)
8. 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 (August 2012)
9. Świątek, P., Stelmach, P., Prusiewicz, A., Juszczyszyn, K.: Service composition in knowledge-based soa systems. New Generation Computing 30, 165–188 (2012)
10. Riabov, A., Liu, Z.: Planning for Stream Processing Systems. In: Proceedings of the National Conference on Artificial Intelligence (2005)
11. Riabov, A., Liu, Z.: Scalable Planning for Distributed Stream Processing Systems. In: Proceedings of ICAPS (2006)
12. Brzostowski, K., Drapała, J., Grzech, A., Świątek, P.: Adaptive Decision Support System For Automatic Physical Effort Plan Generation–Data-Driven Approach. Cybernetics and Systems: An International Journal 44(2-3), 204–221 (2013)