Monitoring and analyzing service execution from business processes: an AXIS extension

Andrea Delgado

Instituto de Computación, Facultad de Ingeniería
Universidad de la República
adelgado@fing.edu.uy

***Abstract*—Implementing Business Processes (BPs) with services (and microservices) is nowadays the main way to support the execution of automated activities in processes, both within the organization itself, and externally interacting with customers, suppliers and other participants. In order to do so, it is important not only to model and implement services but also to define Quality of Service (QoS) characteristics for services, to monitor and evaluate their execution. Although there are many proposals for services monitoring and evaluation from the services point of view, there are not many from the BPs perspective. In this paper we present a reference architecture for service monitoring tools, along with a prototype implementation as an extension of the web services execution environment AXIS2. We show that existing service measures and new ones can be defined into the monitor to collect execution data and relate this data with BPs execution, to measure BPs and service execution in an integrated manner.**

Keywords: Business processes, measuring business processes and services, Quality of Service (QoS), service monitoring.

I. Introduction

Services (and micro-services) are designed as independent software pieces, at different granularity levels (supporting cohesive components with several functionalities or specific separated more fine grained functionalities) to provide support to highly decoupled distributed systems [1]–[4]. Qualities such as lose coupling, high cohesion, high interoperability, explicitly defined interfaces for interactions between systems, among others, are all desired characteristics of services.

Business Processes (BPs) can be modeled and executed [5]–[8] within BP Management Systems (BPMS), which are platforms providing support for systems that are based on BPs. Although there are many approaches and tools for analyzing and evaluating BPs execution [9], [10], and for defining, measuring and monitoring Quality of Services (QoS) characteristics [11]–[18], relating BPs execution with services implementing them, has not been much analyzed yet.

In previous work we have defined a BPs Execution Measurement Model (BPEMM) [22] relating BPs execution with services execution, and defining several measures grouped in defined categories. In [23] we presented an initial approach and discussion of elements, but with focus on modeling services and QoS characteristics, for services specified with the Service Oriented Modeling Architecture Language (SoaML) [24] and the UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms (QFTP) [25]. In [26] we defined a services lifecycle integrated with the BPs lifecycle as defined in [6].

In this paper we present an approach for monitoring, measuring and analyzing services execution within BPs execution. We defined a reference architecture for monitor tools based on the analysis of key functionalities that this kind of software must provide, and a prototype implementation as an extension of the web services execution environment AXIS2[1] Complete code is available here[2]. We integrate services measures within BPs measures, based on the BPEMM model we have defined, to be able to analyze their execution as a whole in order to find improvement opportunities.

The main contributions of the paper are: (i) a service monitoring reference architecture for defining monitor tools, in order to define and collect service execution measures, (ii) the integration of those service execution measures into a whole measurement environment which includes BPs execution measures, and the relationships between BPs and services execution measures, and (iii) a prototype implementation of such reference architecture extending the well-known open source web services environment AXIS2. In order to assess the feasibility of our definitions, we implemented some of the BPEMM service execution measures, but new measures can be defined and integrated.

The rest of the article is organized as follows: In Section II we present background and related work regarding services monitoring. In Section III we describe our proposal including an analysis of existing monitoring tools and the reference architecture for monitors we have defined. In Section IV we present an example of application regarding the monitoring of service execution from a collaborative BP, as a proof of concept. Finally in Section V we present some conclusions and future work.

II. Background and related work

Monitoring software (or services) execution generally implies to collect and analyze execution data on real time, while evaluating software (or services) execution can be viewed as a postmortem analysis of data from a selected period of time. In [18] a review providing an overview and discussion of

[1]Apache AXIS2 Java http://axis.apache.org/axis2/java/core/
[2]Complete code. https://gitlab.fing.edu.uy/open-coal/AXISmonitor

978-1-5386-3057-0/17/$31.00

TABLE I
MAIN CHARACTERISTICS FOR EVALUATION OF MONITORS FROM [19]

Characteristic	Sub-characteristic	Property	Description
Functionality	Suitability	Quality attributes	Quality attributes the monitor is able to monitor
		Monitor	Different configurations the monitor presents
		User	Different management operations for the monitor
		Web Services	Capacities of the monitor to define the WS to be monitored and the conditions
	Accuracy	Verifiableness	Defines if the monitor provides a way to track state
		Effectiveness	Capacity of the monitor to determine the quantity of correct results or effects
	Security	Application	Mechanisms to prevent unauthorized access to the system functionalities
		Data	Mechanisms to ensure data privacy and to prevent them from being corrupted
		Monitor	Mechanisms provided by the monitor to protect messages i.e. SOAP messages
Usability	Operability	System tailorability	Mechanisms to configure monitor tasks in a certain way
		Appearance tailorability	Mechanisms to configure the external appearance of the monitor
		Monitor operability	Mechanisms provided by the monitor to manage its capacities
Efficiency	Resource utilization	Deployment	Defines the monitor needed resources for deployment
		Runtime	Defines the monitor needed resources for runtime

existing proposals for monitors is presented. Existing academic and industrial proposals include: SALMon [18], Dynamo [27], Cremona [28], WebInject [29], Astro [30] and CLAMS [31].

Key requirements for monitor tools are described in [18] and [19] based on an analysis of the services lifecycle to be supported, and the ISO/IEC 9126-1 quality model. These characteristics and a way to evaluate how each tool support them is a key element when comparing and selecting service monitors. Table I presents the main characteristics and sub-characteristics identified for monitors from [19]. Following the concepts from an evaluation methodology we have defined and applied for evaluating Business Process Management Systems (BPMS) [20] [21], we evaluated a selection of monitors based on [18]. However, in most cases, it was not possible to obtain the software to install and execute the monitor in our environment.

Monitors can be of different types: passive when interactions between participants are only looked at, or active when QoS characteristics are also evaluated triggering some action.

Several configurations can be set for the execution of monitors with respect to the execution system that is being monitored, as discussed in [19]. These configurations have different impact on the whole system execution and the measures registered by the monitor, for example in performance, if the monitor executes in the same or different environment than the services. Figure1 puts these configurations in our context, where the BP plays the consumer role (i.e. service tasks in the BP invoking services execution) and the provider role corresponds to the Web services implementing service tasks, which are invoked from the BP.

As shown in Figure 1, these configurations correspond to:

1) Consumer, provider and monitor run all in the same system A
2) Consumer and monitor run in the same system A, the provider runs in a different system B
3) Consumer runs in a system A and the provider and monitor run in a different system B
4) Consumer, provider and monitor run in three different systems A, B and C

In [23] we discussed possible integration of monitors for services execution within BPs execution to support service

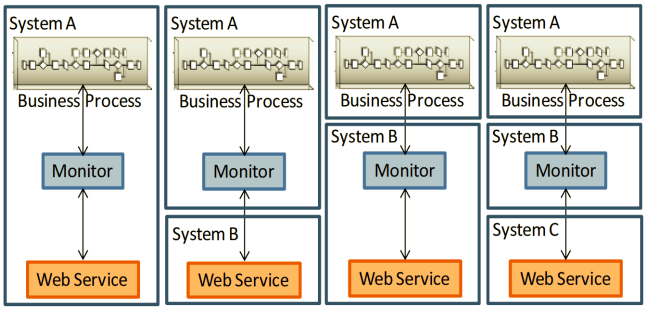

Fig. 1. Monitor configurations adapted from [19]

tasks, and its relation with the configurations presented above. Regarding BPs and services execution measures in [22] we present and analysis of existing models for QoS characteristics, and define the Business Process Excecution Measurement Model (BPEMM) in which we consolidated key measures for both BPs and services. Figure 2 presents the global picture for the time execution measures definitions we use as basis.

As it can be seen in Figure 2, we proposed to measure both BPs execution based on activities in the control flow of the process, and services execution for those activities that are automated based on service invocations. This will allow the identification of improvement opportunities also at the technical leval (i.e. services implementation and/or execution). Although in the definition we present the process engine and the service execution are in the same environment (i.e. server), they can be executed in different environments. Different monitor configurations can be applied depending on the distribution of the system. A complete discussion of the BPs and services measures integration, and different monitor configurations can be seen in [22] and [23].

As an example of the measures we have defined in BPEMM, we present in Figure 3 the definition of the service Response time measure, from the Time category. The defined service measures are integrated within the BPs measures as presented.

Although there are several monitors which provides the desire characteristics, they are mostly not available for free use. Differently to those, our approach provides the basis for defining monitors compliant with those characteristics, and

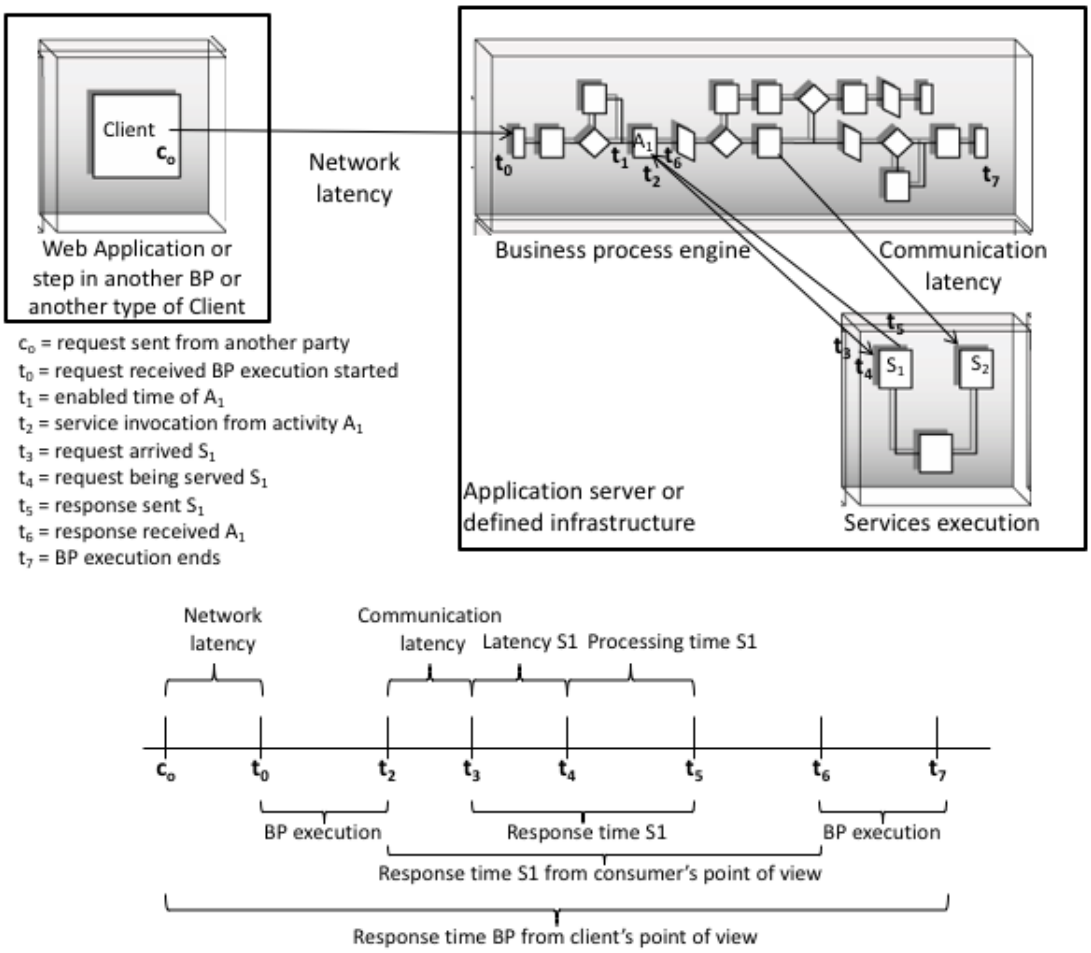

Fig. 2. Global view for time execution measures of BPs and services from [22]

our prototype provides a way to directly implement service execution measures upon an existing and well-known open source web service execution environment, making it easier to adopt.

III. Proposal for services monitoring

To define our proposal we firstly analyzed existing monitoring tools taking into account surveys such as [18] which defined key functionalities and evaluated several monitors, as presented in the related work section. Based on this analysis we selected key functionalities which allowed us to define a reference architecture for monitor tools, and to evaluate existing tools to be used as basis of our proposal. Finally, as we were not able to get most of the tools, and extending them was not straightforward, we decided to extend instead the web services execution environment Axis2.

A. Service monitoring reference architecture

Based on the analysis of key functionalities monitors must provide, we defined a reference architecture for monitor tools which is shown in Figure 4.

The modules we defined for the reference architecture are as follows:

- Monitoring module: main module of the tool. In this module web services to be monitored are defined as well as the measures to be applied to each one. The type of monitoring can also be set in within this module (passive, active, both).
- Integration module: this module allows integration with web services to be monitored. There is no restriction for this integration, it can be a consumer or provider environment, or a separate environment from both.
- Measuring module: this module is in charge of computing defined service measures. This measures can be both predefined and new providing a mechanism to add measures definition.
- Data access module: this module provides an interface for accessing data, which can be as simple as a log file or a complex graphic interface allowing management of the execution data registered within the monitor.
- Notification module: this module provides notification mechanisms to inform events that can occur when monitoring services. For example, when a service violates

Goal	**G1**	**Guarantee (average) service response time to (L1) seconds (L1 label to be changed)**
Question	**Q1**	**What is the actual (average) response time of the service**
Measures	M1 (base)	Invoke time of a service from the activity in the BP (IT = timestamp)
	M2 (base)	Enabled time of a service (ET = timestamp)
	M3 (base)	Start time of a service (ST = timestamp)
	M4 (base)	Completion time of a service (CT = timestamp)
	M5 (base)	Failed time of a service (FT = timestamp)
	M6 (base)	Answer time from the service to the activity in the BP (AT = timestamp)
	M7 (derived)	Service processing time (SPoT = CT - ST)
	M8 (derived)	Service latency time (SLaT = ST - ET)
	M9 (derived)	Service response time (SRpT = SPoT + SLaT)
	M10 (derived)	Service answer time from the BP (SAnT = AT - IT)
	M11 (indicator)	Service Processing time vs. Service Latency time index (STI = SLaT/SPoT) Decision criteria = Index DC
	M12 (indicator)	Average service response time in all BP cases (ASRpT = ∑SRpT/Total service executions in all BP cases) Decision criteria = Index DC
	M13 (indicator)	Average service answer time in all BP cases (ASAnT = SAnT/Total service executions in all BP cases) Decision criteria = Index DC
Decision criteria	Index DC:	R1: 0<=TTI<= L1 = "LOW" = GREEN; R2: L1<= TTI < L2 = "MEDIUM" = YELLOW; R3: L2<= TTI = "HIGH" = RED

Fig. 3. Example of services execution measures defined from [22]

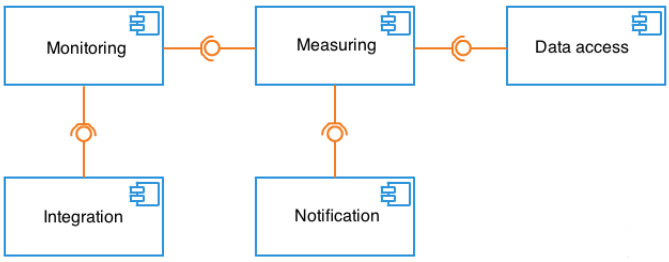

Fig. 4. Reference Architecture for service monitoring tools

SLAs defined, a notification to the service provider can be defined.

B. AXIS extension prototype

We implemented the reference architecture extending the web services execution environment Axis2 with functionalities to monitor web services execution, and the administration console to be able to analyze the measure calculations results.

1) Reference architecture implementation: The implementation for each defined module in the reference architecture in the AXIS 2 execution environment was as follows:

- Monitoring module: the type of monitoring implemented was passive with no possibility to add new measures at this time, although it can be extended to allow it.
- Integration module: as AXIS2 executes in the same environment as the web services being monitored, we use this as the basis integration option.
- Measuring module: it only calculates the predefined measures we included in the extension, although it can be extended for new measures.
- Data access module: we decided to log events in files so in this implementation this module only reads and writes these files.
- Notification module: we left this module outside the scope of the prototype, but can be easily added.

As a proof of concept, we integrated in the AXIS2 extension the logging of events that correspond to the base measures defined in the BPEMM model [22] to register several times and data regarding services execution:

- Enabled time (ET): this entry logs the event when a service has been dispatched but has not yet started its execution.
- Completed time (CT): this entry logs the event when a service successfully completes it execution.
- Failure time (FT): this entry logs the event when a service failed in its execution.
- Authentication failure time (AFT): this entry logs the event when the authentication to execute the service fails.

To implement an AXIS2 module, a class implementing the interface *org.apache.axis2.modules.Module* must be added, which allows to take actions in the lifecycle of the module. Also, a configuration file module.xml must be provided inside the META-INF directory, which defines the handlers implemented by the module, and the phase each one executes. Listing 1 shows the configuration file *module.xml* we added.

Listing 1. Configuration file module.xml with handlers definition

```
<module name ="logging"
    class="monitor.loggingmodule.LoginModule">
<InFlow>
   <handler name="InflowLogHandler"
       class="monitor.loggingmodule.LogHandler">
     <order phase="loggingPhase">
   </handler>
   <handler
       name="InflowCheckAuthenticationHandler"
       class="monitor.loggingmodule.CheckAuthen
    ticationHandler">
     <order phase="Addressing"
        phaseLast="true">
   </handler>
</InFlow>
<OutFlow>
   <handler name="OutflowLogHandler"
       class="monitor.loggingmodule.LogHandler">
     <order phase="loggingPhase">
   </handler>
</OutFlow>
<OutFaultFlow>
```

```
   <handler name="FaultOutflowLogHandler"
       class="monitor.loggingmodule.LogHandler">
     <order phase="loggingPhase">
    </handler>
</OutFaultFlow>
<InFaultFlow>
   <handler name="FaultInflowLogHandler"
       class="monitor.loggingmodule.LogHandler">
     <order phase="loggingPhase">
    </handler>
   <handler
       name="FaultInflowCheckAuthentication
    Handler"
        class="monitor.loggingmodule.CheckAuthen
    ticationHandler">
     <order phase="Addressing"
         phaseLast="true">
    </handler>
</InFaultFlow>
```

Two specific handlers were implemented for the module: CheckAuthentication which is responsible for checking whether there are failed authentications creating an entry in the log file for each case (measure AFT); and the LogHandler which is in charge of creating entries for the rest of the measures: ET, CT and FT.

We also added an entry to log the start time (ST) of a service, which corresponds to the time the service starts its execution. To do so for services defined by AXIS we added several classes implementing the interface *org.apache.axis2.engine.MessageReceiver*, since AXIS 2 supports several Message Exchange Patterns (MEPs) which describe the message patterns required by a communication protocol to establish or use a communication channel [2]. For each MEP natively supported by AXIS 2 we added an extension class with the implementation of the logic needed to register the ST entry into the logs event file. For example, a class extending org.apache.axis2.rpc.receivers.RPCMessageReceiver was implemented to support the bidirectional RPC MEP. Figure 5 presents the approach described to extend AXIS 2, and Listing 2 shows the definition of messages receivers in AXIS 2.

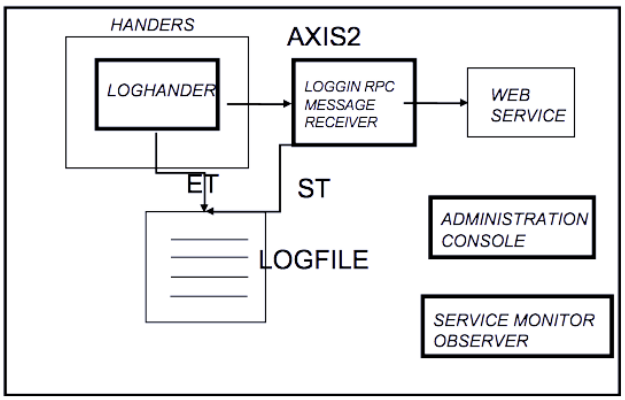

Fig. 5. Approach used to extend AXIS 2

Finally, we also added the logging of events corresponding to when services stop executing as well as when they resume execution, including the following:

Listing 2. Message receivers definition in AXIS 2

```
<messageReceivers>
  <messageReceiver
      mep="http://www.w3.org/2004/wsdl/in-only"
  class="monitor.receivers.LoggingRawXLMIN
      OnlyMessageReceiver"/>
   <messageReceiver
       mep="http://www.w3.org/2004/wsdl/in-out"
   class="monitor.receivers.LoggingRawXLMINOut
      MessageReceiver"/>
   <messageReceiver
       mep="http://www.w3.org/2006/wsdl/in-only"
   class="monitor.receivers.LoggingRawXLMIN
   OnlyMessageReceiver"/>
   <messageReceiver
       mep="http://www.w3.org/2006/wsdl/in-out"
   class="monitor.receivers.LoggingRawXLMINOut
MessageReceiver"/>
  <messageReceiver
      mep="http://www.w3.org/2004/wsdl/in-only"
   class="monitor.receivers.LoggingRPCInOnly
MessageReceiver"/>
   <messageReceiver
       mep="http://www.w3.org/2004/wsdl/in-out"
   class="monitor.receivers.LoggingRPCMessage
   Receiver"/>
</messageReceivers>
```

- Resuming time (RT): this entry logs the event when a service resume its execution after being down.
- Shutdown time (SDT): this entry logs the event when a service stops its execution.

In each case the entry is also registered associated to each operation defined in the service, for measures calculating purposes. To log these events we implemented the interface *org.apache.axis2.engine.AxisObserver* which allows to get events from AXIS 2 associated with services execution, as follows:

- AxisEvent.SERVICE_START: when a service resumes its execution from the administration console.
- AxisEvent.SERVICE_DEPLOY: when a service is deployed within AXIS, either when AXIS is starting up or dynamically after.
- AxisEvent.SERVICE_STOP: when a service execution is stopped from the administration console.
- AxisEvent.SERVICE_REMOVE: when a service is removed, either from the administration console or manually deleting the files from the server.

In the first two cases the entries corresponding to the RT measure will be registered in the event log, and when the two last events occur the entries corresponding to the SDT measure will be registered. We used the Log4J library to register the entries in the event log file.

To calculate the defined measures or new ones that can be added to the AXIS extension, we defined a general abstract class in which the operations for parsing and calculating measures are defined. In this way the extension is flexible enough to integrate new measures in an easy way, just adding the new class that implements the defined methods for the new measure. Figure 6 shows the defined hierarchy with the abstract class and an example of implemented measures.

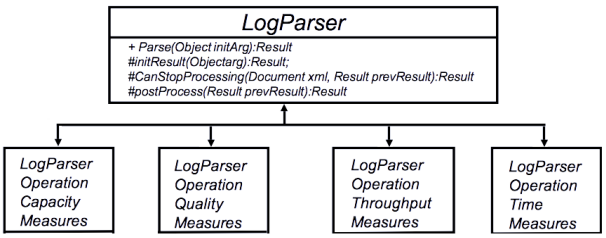

Fig. 6. Generic event log parsing for adding new measures

To support the implementation of quality measures such as confidentiality for when authentication of services failed, we used the WS-* family of standards. In particular WS-Security to add security to the communication channel and WS-SecurityPolicy to define policies between the service consumer and the provider for the exchange of user and password.

2) Administration console extension: Axis2 administration console already provides several functionalities to manage the web services that are deployed in the server, by means of several JSP pages. Among this functionalities are: upload a new service, visualize data from services, service groups, modules and phases, visualize the execution chain among handlers for each phase and order execution, enable a module, actions over services such as stop, resume, edit data, etc. in Figure 7 shows a (partial) snapshot of the extended administration console, showing the measures we added to the prototype: time, throughput, capacity and quality.

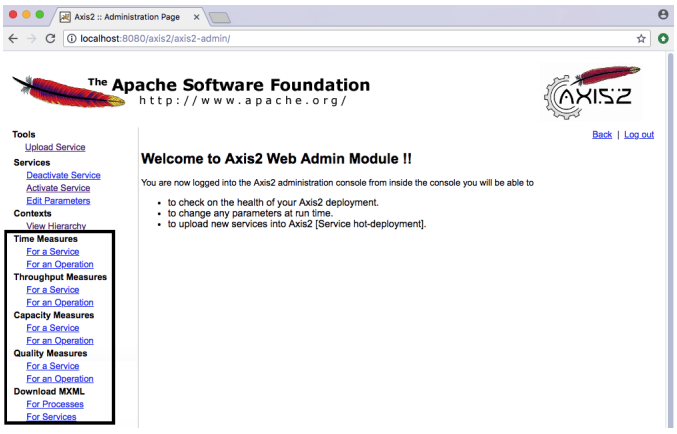

Fig. 7. Extended AXIS2 administration console

By extending the administration console we were able to reuse several functionalities such as the visualization capacity over AXIS components, and also add new functionalities. To do so, we implemented several new JSP pages containing the new options, and extended the administration servlet which is in charge of getting the data for each page and instantiate the parsers for calculating the defined measures. The administration console can be accessed navigating to the URL `<ip_servidor>:<puerto_servidor>/axis2/axis2-admin/` using the credentials that comes with the typical installation of AXIS 2.

IV. Example of Application

In this section we present the example of application we carried out to validate the proposal and the prototype implemented. To do so, we used a real but simplified collaborative business process from a Hospital which defines several service tasks implemented by web services, and involves three different participants, as shown in Figure 8.

The collaborative process involves three participants: the hospital, the central health organization, and the patient. Web services implementation to support the execution of automated service tasks are marked with an ellipse, two in the hospital, and one in each other participant. In a real execution environment, each process will execute in a different environment within each participant, and invocations to web services will have to cross the borders of each one via Internet channels.

To simulate the execution environment we provide two web servers Tomcat in which we distributed the execution of processes in Activiti BPM, and the web services and monitor in the AXIS 2 extension prototype. In the first server we executed Activiti BPM with the hospital process, and AXIS 2 prototype with the hospital web services, in the other we executed Activiti BPM with the patient and central health processes, and AXIS 2 prototype with their web services. Invocation of web services are from one process to the other so execution times and context are close to a real distribution in different organizations. Figure 9 shows the distribution of processes, web services and AXIS 2 prototype.

To be able to collect several execution events adding the corresponding entries to the events log file, we launched several executions using Jmeter [3] which allows to initiate the patient process which starts the execution of the complete collaborative process. We tried several 500 execution instances to be able to get a considerable amount of execution data. In Figure 10 we present as an example, the measures result table for the execution of the ReceiveRequestAppointmentWS defined in the hospital process.

In Figure 10 the average times and other consolidated measures are shown on the left corner; each row of the table shows the times for each service instance execution with the basic measures that are registered as entries in the event log.

V. Conclusions and Future Work

In this paper we presented an approach for monitoring, measuring and analyzing services execution within BPs execution. We defined a reference architecture for service monitor tools based on decoupled modules and a proof of concept implementation as an extension of the web services execution environment AXIS2. We integrated services measures within BPs measures, to be able to analyze their execution as a whole, to find improvement opportunities. To do so, we applied service measures from our previous work in the BPEMM model, to collect, register, calculate and visualize service execution measures. Although we did not implemented all measures in the simulation we carried out, we selected key ones which allowed us to show the feasibility of the approach.

[3]JMeter. http://jmeter.apache.org/

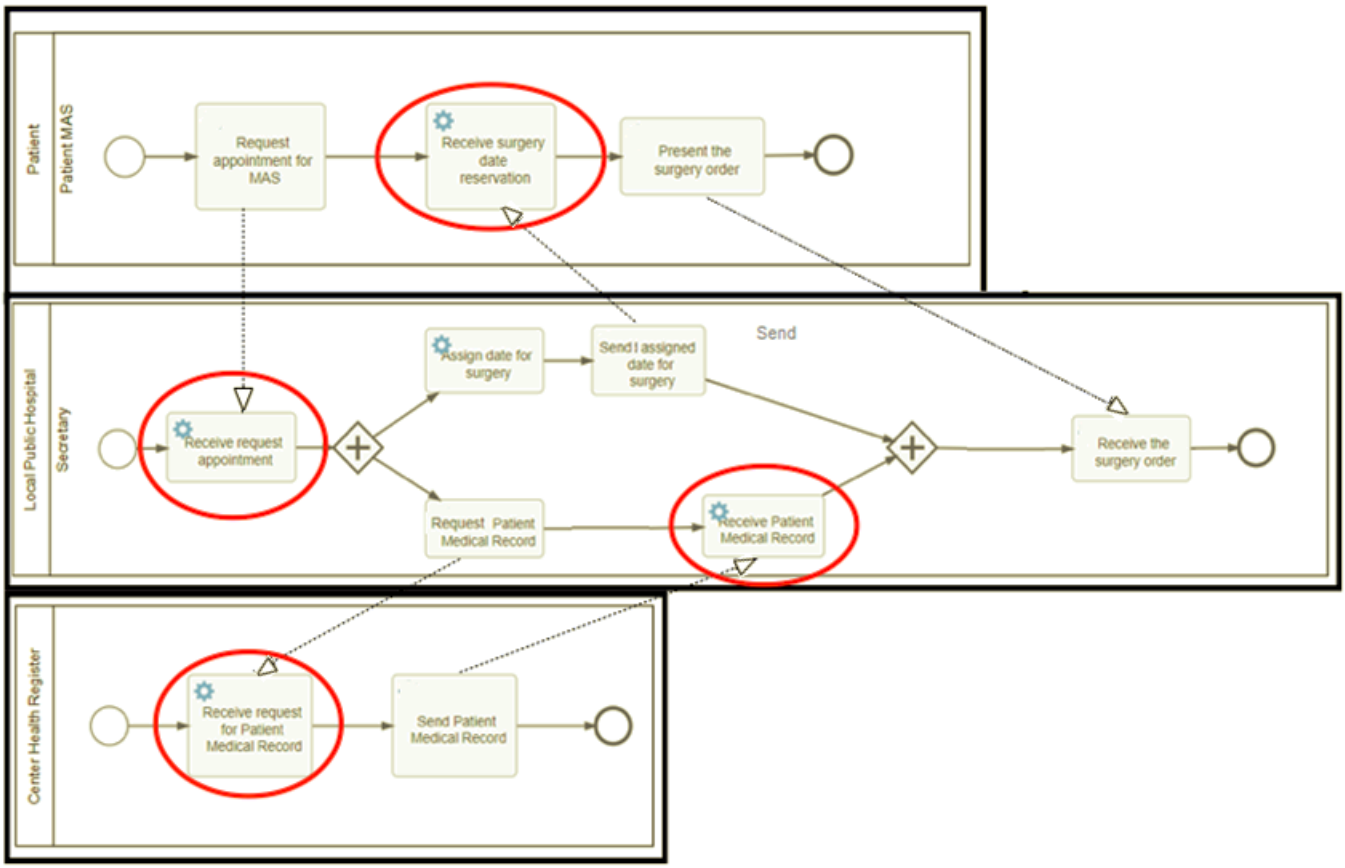

Fig. 8. Case study Business Process from a Hospital

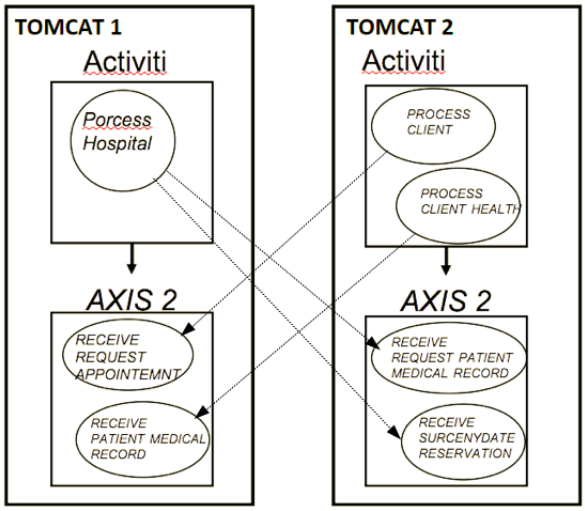

Fig. 9. Case study execution: processes, WS and prototype distribution

Although when compared to existing tools there are many monitors that provide more advanced and/or complete functionalities, they are mostly not available for free use or to be extended (the code is also not available). Our tool is based on standards, extending an existing open source tool, which provides an open base for use and extension. We are also able to generate event logs in MXML format for the measures to be integrated within the ProM framework[4], as presented in [23]. We believe that integrating BPs and services execution measures is a key element to support the continuous improvement of organizations, both from the point of view of their BPs and their services implementation. Our proposal provides conceptual support and and initial tool support, which we plan to enhanced in the future to support more execution measures and monitoring functionalities, and to improve the integration between monitoring and evaluation tools.

VI. Acknowledgement

I would like to thank student Martín Vázquez who worked in the proposal and the AXIS2 extension implementation.

References

[1] Papazoglou, M., Traverso, P., Dustdar, S., Leymann, F.: Service-Oriented Computing: a research roadmap. Int. J. of Coop. Inf. Systems. 17(2), 223–255 (2008)
[2] Erl, T. : Service-Oriented Architecture: Analysis and Design for Services and Microservices, 2nd Edition. Prentice Hall (2016)
[3] Krafzig,D., Banke, K., Slama,D.: Enterprise SOA: Service Oriented Architecture Best Practices. Pearson Education (2005)
[4] Papazoglou, M.:Web Services and SOA: Principles and Technology, 2nd edition, Pearson Education Canada (2012)
[5] van der Aalst, Wil M. P. and ter Hofstede, Arthur H. M., Weske, M.: Business Process Management: A Survey. In: Int. Conf. of BPM (BPM), pp. 1–12, (2003)

[4]ProM framework. http://www.promtools.org/doku.php

Time Measures for a Service

ReceiveRequestAppointmentWS

Service processing time average = 14.99
Service latency time average = 1.05
Service response time average = 16.04

Show 10 entries Search:

Operation name	Enabled date	Begin date	End date	State	Service processing time (ms)	Service latency time (ms)	Service response time (ms)	Service index (ms)
addRequestedAppointment	2017-11-17T23:38:02.472	2017-11-17T23:38:02.491	2017-11-17T23:38:02.914	Completed	423	19	442	0.04
addRequestedAppointment	2017-11-17T23:38:04.468	2017-11-17T23:38:04.468	2017-11-17T23:38:04.478	Completed	10	0	10	0.00
addRequestedAppointment	2017-11-17T23:38:04.582	2017-11-17T23:38:04.583	2017-11-17T23:38:04.589	Completed	6	1	7	0.17
addRequestedAppointment	2017-11-17T23:38:04.671	2017-11-17T23:38:04.671	2017-11-17T23:38:04.678	Completed	7	0	7	0.00
addRequestedAppointment	2017-11-17T23:38:04.746	2017-11-17T23:38:04.747	2017-11-17T23:38:04.753	Completed	6	1	7	0.17
addRequestedAppointment	2017-11-17T23:38:04.819	2017-11-17T23:38:04.819	2017-11-17T23:38:04.827	Completed	8	0	8	0.00
addRequestedAppointment	2017-11-17T23:38:04.929	2017-11-17T23:38:04.929	2017-11-17T23:38:04.936	Completed	7	0	7	0.00
addRequestedAppointment	2017-11-17T23:38:05.010	2017-11-17T23:38:05.010	2017-11-17T23:38:05.054	Completed	44	0	44	0.00
addRequestedAppointment	2017-11-17T23:38:05.162	2017-11-17T23:38:05.162	2017-11-17T23:38:05.170	Completed	8	0	8	0.00
addRequestedAppointment	2017-11-17T23:38:05.238	2017-11-17T23:38:05.238	2017-11-17T23:38:05.246	Completed	8	0	8	0.00

Showing 1 to 10 of 500 entries First Previous 1 2 3 4 5 … 50 Next Last

Fig. 10. Time measures for the execution of service ReceiveRequestAppointmentWS

[6] Weske, M.: Business Process Management: Concepts, Languages, Architectures, 2nd Edition. Springer (2012)
[7] Dumas, M., La Rosa, M., Mendling, J., Reijers,H.: Fundamentals of Business Process Management. Springer (2013)
[8] Chang, J.F.: Business Process Management Systems: Strategy and Implementation. Auerbach Publications, Taylor & Francis Group (2005)
[9] Wil M.P. van der Aalst, Process Mining: Discovery, Conformance and Enhancement of Business Processes, Springer-Verlag, (2011)
[10] Wil M.P. van der Aalst, Data Science in Action, Springer-Verlag, (2016)
[11] Barbacci, M., Klein, M., Longsta, T., Weinstock, C. (1995). Quality Attributes, SW Engineering Institute (SEI), CMU/SEI-95-TR-021.
[12] O'Brien, L., Bass, L., Merson, P. , (2005), Quality Attributes and SOA, SW Engineering Institute (SEI), CMU/SEI-20055-TN-014.
[13] ISO/IEC 25010 Systems and Software Engineering Square (Systems and Software Quality Requirements and Evaluation), (2005-11). http://www.iso.org/.
[14] W3C, QoS for WS: Requirements and Possible Approaches, (2003). http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/
[15] OASIS, Web Services Quality Factors v 1.0, (2012). http://docs.oasis-open.org/wsqm/wsqf/v1.0/WS-Quality-Factors.pdf .
[16] D'Ambrogio, A., A Model-driven WSDL Extension for Describing the QoS of WS. 13th International Conference on WS (ICWS), (2006).
[17] Oriol M., Marco J.,Franch X., Quality models for web services: A systematic mapping, Information and Software Technology, 56 (3), (2014).
[18] Oriol M., Marco J.,Franch X., Marco J., Monitoring the service-based system lifecycle with SALMon, Expert Systems with Applications (42), pp. 6507–6521, (2015)
[19] O. Cabrera and X. Franch, A quality model for analysing web service monitoring tools, 6th International Conference on Research Challenges in Information Science (RCIS), pp. 1-12, (2012)
[20] A. Delgado and D. Calegari and P. Milanese and R. Falcon and E. Garcia, A Systematic Approach for Evaluating BPM Systems: Case Studies on Open Source and Proprietary Tools, 11th IFIP 2.13 Int. Conference on Open Source Systems: Adoption and Impact OSS, pp. 81-90, (2015)
[21] A. Delgado and D. Calegari, Evaluating non-functional aspects of business process management systems, XLIII Latin American Computer Conference (CLEI), pp. 1-10, (2017)
[22] Delgado, A.: An integrated approach based on execution measures for the continuous improvement of BPs realized by services, Information and Software Technology,56 (2), pp. 134–162, (2014)
[23] Delgado A., Modeling and measuring services to support BPs execution: from SoaML and QoS models to WS, Int. Journal Services Comp (IJSC),4,pp.65, (2016)
[24] OMG: Service Oriented Architecture Modeling Language (SoaML), https://www.omg.org/spec/SoaML/, (2012)
[25] OMG: UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms (QFTP), https://www.omg.org/spec/SoaML/, (2008)
[26] Delgado, A.: A Services Lifecycle to Support the BPs Lifecycle: From Modeling to Execution and Beyond. In: IEEE International Conference on Services Computing (SCC), pp.831–835 (2016)
[27] L. Baresi, S. Guinea, Dynamo: Dynamic Monitoring of WS-BPEL Processes, 3rd. International Service-Oriented Computing (ICSOC), pp. 478-483, (2005).
[28] H. Ludwig, A. Dan, and R. Kearney, Cremona: An architecture and library for creation and monitoring of ws-agreements, 2nd international conference on Service oriented computing (ICSOC), pp. 65-74, (2004).
[29] WebInject - Web (HTTP) testing and monitoring tool, http://www.webinject.org
[30] Barbon F., Traverso P., Pistore M., Trainotti M., Run-Time Monitoring of Instances and Classes of WS Compositions, IEEE Int. Conf. on WS (ICWS), pp. 63–71, (2006).
[31] Alhamazani K., Ranjan R., Mitra K., Jayaraman P., Huang Z., Wang L., Rabhi F., CLAMS: Cross-Layer Multi-Cloud Application Monitoring-as-a-Service Framework, IEEE International Conference on Services Computing (SCC), (2014)