Melodic
Big data cloud

Title:

Integration and adaptation strategy

Multi-cloud Execution-ware for Large-scale Optimized Data-Intensive Computing

H2020-ICT-2016-2017
Leadership in Enabling and Industrial Technologies; Information and Communication Technologies

Grant Agreement No.:
731664

Duration:
1 December 2016
30 November 2019

www.melodic.cloud

Deliverable reference:
5.01

Date:
31 May 2018

Responsible partner:
7bulls

Editor(s):
Paweł Skrzypek

Author(s)
Jörg Domaschka, Sebastian Schork, Marcin Prusiński, Paweł Skrzypek, Yiannis Verginadis, Geir Horn

Approved by:
Ernst Gunnar Gran

ISBN number:
N/A

Document URL:
http://www.melodic.cloud/deliverables/D5.01 Integration and adaptation strategy.pdf

Abstract

One main factor for the successful design and implementation of Melodic is to provide a proper integration and adaptation strategy that integrates the platforms on which Melodic is built, such as Cloudiator and PaaSage. This includes not only the integration of components within the above frameworks, but also the development of new components and mechanisms in Melodic to handle Big Data management and security aspects. The integration plan may lead to the adaptation of components involved in an integration, which calls for a proper adaptation strategy. In terms of the integration architecture, we consider two layers of integration: a control plane and a monitoring plane. The former is for the integration of actions in a control flow and the latter is for gathering, processing, propagating, and storing monitoring events. From the viewpoint of integration models, we investigate four popular integration strategies, including point-to-point integration, queue-based middleware integration, Enterprise Application Integration (EAI) or Enterprise Service Bus (ESB) based integration, and EAI/ESB integration with Business Process Management (BPM) orchestration. To evaluate these integration strategies, a methodology is proposed for choosing the integration and adaptation strategy. The main steps of the methodology include identifying the integration requirements, evaluating integration methods, estimating the effort needed to implement a given integration strategy, ranking the methods, recommending a method, and finally determining the adaptation strategy based on the chosen integration method. The methodology has resulted in using ESB/BPM for integration at the control layer, and Active Message Queue (ActiveMQ) at the monitoring layer using the ActiveMQ infrastructure which is built into the selected MuleESB.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Document	
Period Covered	M2-12
Deliverable No.	D5.01
Deliverable Title	Integration and adaptation strategy
Editor(s)	Paweł Skrzypek
Author(s)	Jörg Domaschka, Sébastien Kicin, Marcin Pruisiński, Paweł Skrzypek, Yiannis Verginadis, Geir Horn
Reviewer(s)	Yiannis Verginadis, Kyriakos Kritikos, Thomas Dreibholz
Work Package No.	5
Work Package Title	Integration and security
Lead Beneficiary	7bulls
Distribution	PU
Version	Final
Draft/Final	Final
Total No. of Pages	45

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table of Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Index of Figures

Index of Tables

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

1 Introduction

The right choice of integration strategy is crucial for the successful implementation of the given project, as there are plenty of integration methods available, each of them with certain advantages and disadvantages. The proper methodology of choosing the best integration method for the given requirements of a system is a quite complex task. As stated in [1], a properly chosen integration strategy could provide significant benefits for the usability of the released Melodic[1] platform, in terms of stability, reliability, performance as well as reduced cost of development and maintenance.

The purpose of this deliverable is to evaluate different strategies for integration and adaptation (change components of underpinning frameworks to fit to Melodic platform) and to select the best possible according to the objectives of the project. The selected strategy will also be analysed in order to highlight its main benefits and advantages.

As one of the main directions of work in the Melodic project is focused around the integration of the underlying PaaSage[2] and Cloudiator[3] frameworks, the proper integration and adaptation strategy is crucial for the success of the project. Also, the context-aware access control mechanism developed in the PaaSword[4] project was planned to be integrated with Melodic but owing to licensing issues this was not possible. Nevertheless, a context-aware authorisation engine will be developed and integrated with the Melodic platform. More details about Melodic's way to mitigate the decisions of the PaaSword project are provided in D2.1 "System specification" deliverable, Subsection 6.3.

PaaSage is an open source integrated platform to support both the design and deployment of Cloud applications. Together with an accompanying methodology, PaaSage supports model-based configuration, optimization and deployment of these applications. PaaSage allows for deploying existing and new applications independently of the existing underlying Cloud infrastructures.

The Cloudiator framework has been developed in the PaaSage project and extended in the Cactos project. It is a cloud service orchestration framework that goes beyond the boundaries of a single cloud provider.

For the purpose of this document, the strategy is defined as a high level, general plan, which is used as a guidance for the implementation of a certain detailed method. To this end, the integration strategy defines a high-level plan for integrating components of underlying projects along with a number of new components to be developed by the Melodic consortium. The integration method, for the purpose of this document, is the detailed plan of integration, with a set of tools and procedures. The adaptation strategy is closely interrelated to the integration strategy and defines

[1] http://melodic.cloud/
[2] http://www.paasage.eu
[3] http://www.cactosfp7.eu/
[4] https://www.paasword.eu/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

a high-level plan for changing components of underlying projects in order to be usable in the Melodic platform. In this deliverable, the general adaptation strategy is presented, while details of the adaptation of integrated components in the form of the list of changes to the underlying frameworks are described in the D5.02 "Updates to OSS frameworks" deliverable.

1.1 Structure of document

The rest of this deliverable is divided into four logical parts. In the first part of the document, the current integration methods, as adopted in PaaSage and Cactos, are analysed along with their pros and cons. The second part of the document is dedicated to analysing the methodology for the selection of the right integration and adaptation strategies for Melodic. The third part of the document explains how the aforementioned methodology has been applied, and what are its application results. It also explains the rationale for selecting the respective integration and adaptation strategies for Melodic. Finally, the last document part elaborates more on Melodic's selected integration and framework component adaptation strategies.

The detailed structure of the document is as follows:

- Introduction (Section 1) – this section describes the main objectives and structure of this document.
- Integration in PaaSage and Cloudiator (Section 2) – the section contains a description of current integration methods used within the PaaSage and Cactos projects along with the list of the most important issues related to the current integration approach that was followed.
- Methodology of choosing integration and component adaptation strategy (Section 3) – description of the devised methodology for deciding on the integration and adaptation strategies for Melodic.
- Methodology Application (Section 4) – detailed application of the methodology with the supply of respective results as well as the final selection of the integration and adaptation strategies for Melodic.
- Integration and adaptation method for Melodic (Sections 5) – description of integration and adaptation strategies for the Melodic project, selected based on the methodology application results, for both the Control and the Monitor Planes, as presented in deliverables D2.1 "System specification" and D2.2 "Architecture and Initial Feature Definition".
- Summary (Section 6) – conclusions and next related steps.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

1.2 Glossary

Table 1: Specific terms used in the deliverable

Term used in deliverable	Explanation of the term
Active – passive mode	Mode of the High Availability (HA)/multi-instance configuration where one component's instance is active and handles requests. A second instance is up and will start handling request in case of failure of the first instance
Active – active mode	Mode of the HA/multi-instance configuration where all component's instances are up and running and handling requests
Business Process Management (BPM)	Standard way of describing and executing processes on the workflow/process engines
Strategy	General approach to accomplish a given task, with guidelines and overall description
Method	Detailed approach or solution to achieve a goal
Integration strategy	Set of guidelines, assumptions and general directives related to the integration of components within a given IT system
Adaptation strategy	Set of guidelines, assumptions and general directives related to adaptation of the technology and the components in a given IT system. For the purpose of the deliverable as adaptation we understand alignment (change) of the components from underpinning components to the Melodic plaform.
Integration	Ability to communicate, invoke method/interfaces by different components
Adaptation	Adjustment and changes of a given component or technology needed to fit it to a particular IT systems
Application Programming Interface (API)	The definition of the interfaces of a system or application made available to be invoked by external parties.
Enterprise Service Bus (ESB)	A method for integration of IT systems or components
Enterprise Application Integration (EAI)	All tasks, activities, methods and tools used for integrating applications within enterprise.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Term used in deliverable	Explanation of the term
High Availability (HA)	High level of availability of an IT system or application. Usually means that the system is installed in more than one instance.
Hyper Text Transfer Protocol (HTTP)	Protocol used as the communication backbone of Web Internet services to exchange data between the server and the browser. Also used as a transport protocol for modern integration methods like the Simple Object Access Protocol (SOAP) or Representational State Transfer (REST).
Microservices	An approach to develop a single application as a suite of small services, each running in its own process and communicating with dedicate API mechanisms, often an HTTP resource API. These services are built around business capabilities and are independently deployable by a fully automated deployment machinery[5]
Control Plane	Integration layer responsible for handling action and data flow in the system
Monitoring Plane	Integration layer responsible for handling all monitoring related events and operations
Simple Object Access Protocol (SOAP)	A method for the integration of IT systems
Representational State Transfer (REST)	A method for the integration of IT systems
Queue-based communication	Communication between IT systems based on a queue of messages, usually asynchronous
Synchronous communication method	Direct method of communication between IT systems, where the invoker is blocked until it receives a corresponding response
Asynchronous communication method	Indirect (usually through queue message broker) method of communication between IT systems, where the invoker is not blocked until it receives the respective response

[5] https://martinfowler.com/articles/microservices.html

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

2 Integration in PaaSage and Cloudiator

In this section, the current (as–is) state of the integration layer in the PaaSage and Cactos projects is provided. In particular, we describe the Control as well as the Monitoring Plane in these projects. For the Cactos project, only the Cloudiator part is mentioned, as this is the sole part of the project related to Melodic.

2.1 Description of integration in PaaSage

The current communication between PaaSage components is organized in a point-to-point manner with the use of ZeroMQ[6] as a messaging mechanism. The main advantage of this solution is the low-latency of transporting the messages between components, as ZeroMQ introduces very little processing overhead. The diagram below shows the current connections between the PaaSage components:

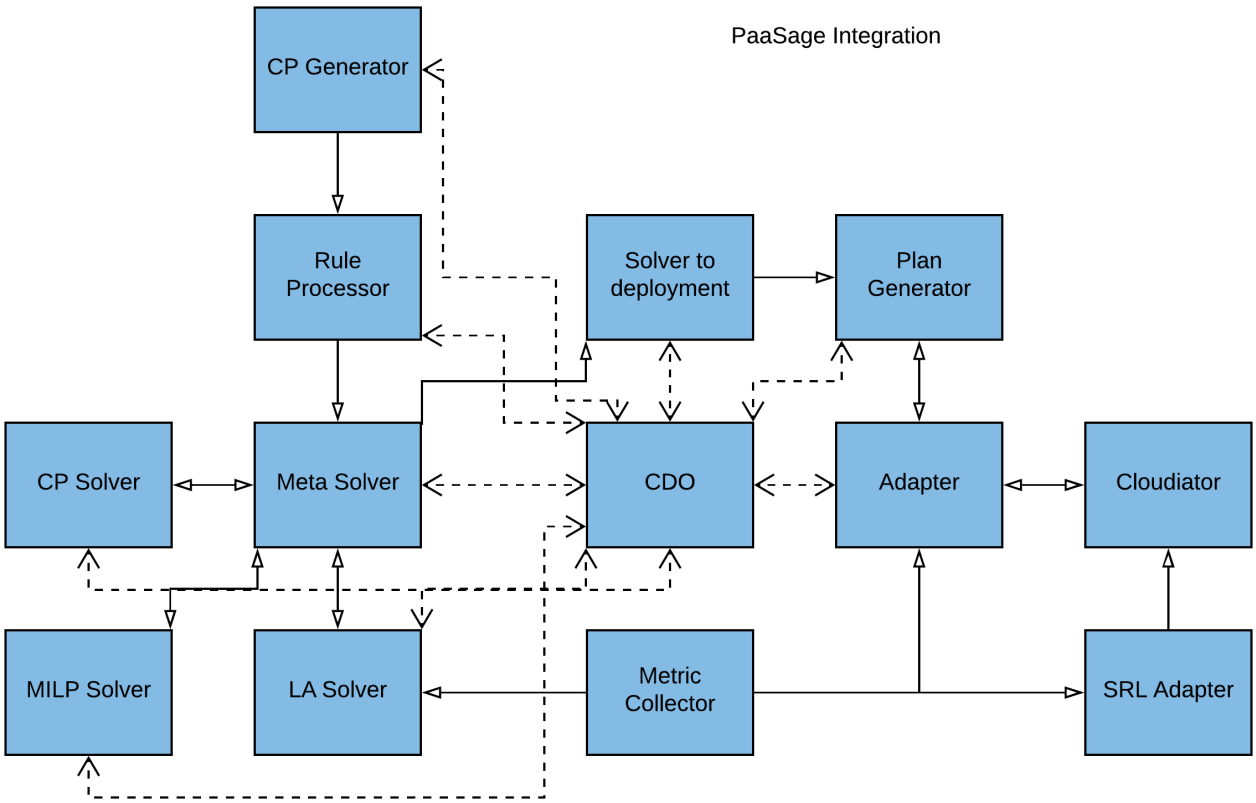

Figure 1: Current connections between the PaaSage components - demonstration diagram. Dashed lines show storing and retrieving models, solid lines show communication between components.

[6] http://zeromq.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

It is important to distinguish the two data flows present in the above demonstration diagram as they are differentiated in many ways:

- One related to the control messages (Control Flow)
- Another related to the metrics and monitoring (Monitoring flow)

The definition of these flows are provided in the Glossary.

Currently in PaaSage, both flows are implemented using ZeroMQ, with the model repository as a way of storing and sharing data. Integration with Cloudiator is implemented via REST API invocation.

2.2 Monitoring components in PaaSage

Monitoring and collection of metric values are fundamentally the first steps in building an Event Management System (EMS). The next step is how events are handled. In PaaSage, the monitoring and metrics collection is handled by central component called *Metric Collector* and distributed components of Cloudiator, deployed on each virtual machine. The Metric Collector in PaaSage receives all measurements from all deployed VMs and centrally computes composite metric values resulting from the updated measurements. The updated set of metric values represents the application's current execution context, and this is then used by the solvers, by the adapter, and by the components dedicated to evaluating and executing platform local scalability rules. Combined, the monitoring and the platform local scalability rule processing may be considered the first steps of an EMS.

As metric collection and monitoring are key features needed for continuous reconfiguration of the deployed application, the PaaSage mechanism has been carefully analysed and evaluated. The metric collection and monitoring components in PaaSage were built from scratch for PaaSage. This approach has the following shortcomings, which have been identified based on preliminary exploitation of PaaSage, our own experience, and new discoveries related to Event Management Systems:

- The main issue is that the PaaSage monitoring infrastructure is just a starting point for a proprietary EMS and for a professionally built and stable platform, as aimed for in Melodic, one would need to expand this EMS to ensure maintainability and scalability. The question is then whether it would be more efficient and more economical to use an available and proven solutions for the metrics collection and the platform local rule processing. There are many proven, open source solutions for event rule-based processing (e.g. Esper[7] and Drools[8]) and according to the open source philosophy it would be better to build on other

[7] http://www.espertech.com/esper/
[8] https://www.drools.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

open source projects than re-inventing the wheel. The available solutions already have established developer communities, and Melodic can use and contribute to a mature, stable, and feature rich solution instead of building one from scratch. Furthermore, such mature solutions are continuously developed, and their capabilities are increasing, which would require more effort on behalf of Melodic in case of a custom solution.

- The metric collection component in PaaSage is fixed to only two-layer architecture: VM level and PaaSage platform level. The Metric Collector component supports only a REST API in PaaSage. It is not possible to create many layers with different aggregation and filtering rules on each layer. This is a very important limitation in scalability of the solution, especially for very large applications. Consider, for example, an application defined metric "average response time to the user" for a data intensive application deployed in three Cloud providers with three regions used for each provider and 500 VMs in each region; and the metric values are averages over the response times recorded in time windows of 10 seconds. In the PaaSage two-layer architecture there will be 4500 REST requests from 9 locations to the Metric Collector component every 10 seconds. In case of multi-layer aggregation, each of the 500 VMs will report their average metric values to an EMS at the region level. Between the region level and the Cloud provider level there will be only three values collected per each 10 seconds' epoch, and between the Cloud provider and the Melodic platform level there will be only three metric values per epoch to collectively summarise the execution context of the whole deployment. There might of course be other metric types that cannot be aggregated hierarchically and must be communicated directly to the Melodic platform, but in this case an EMS is not worse than the current PaaSage implementation. Furthermore, an EMS allows events to be created from monitored metric values at low level. For example, if the above mentioned VMs are collecting the number of users connected to each VM as an application specific metric, there could be an overload event generated by the EMS in the VM if the number of users exceeds a given threshold. Then only this overload flag is the metric that must be transmitted out of the VM, whereas in PaaSage all the individual user counts must be transmitted individually to the Metric Collector before some external rule processor could conclude that one of the machines were overloaded.
- Event rules are not really supported in PaaSage. However, PaaSage has a Scalability Rule Language (SRL) processed by a dedicated component called SRL Adapter aimed at processing the metric values from the Metric Collector and making a Cloud provider local decision about a provider local scaling. The SRL rules are defined in CAMEL and the SRL Adapter extracts them from the CAMEL deployment model. However, the PaaSage SRL does not support the full expressivity for detecting complex events as the one offered by already established open-source tools since the PaaSage SRL is limited to a fixed set of processing, aggregating, and filtering rules. Extending the SRL to become a flexible event processing language requires changes to CAMEL as well as the parsers and rule processors as part of the SRL Adapter. This seems like an unnecessary effort when every standard EMS supports

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

an event processing language. Some of these languages can be directly characterised as event-condition-action (ECA) based languages, like the one that Drools CEP engine uses, but most can be characterized as SQL-like event processing languages able to support simple, or more complex, event algebra operators over time or event windows. Irrespective of how the language is formulated, event processing rules can easily be embedded into CAMEL as plain text strings that are forwarded to the EMS language parser, which must then only be extended to connect the event and condition parts of the rules to the involved metric values. Since the EMS also collects the metric values it will be relatively easy to ensure the correct internal mapping.

Changing the metric management from a two-level collection to a multi-level EMS implies changes to the ways metric values are exchanged among the components. Using REST to push the value events from the VMs to the Metric Collector was possible in PaaSage as the unique receiver end-point was known to new VMs. The synchronous operation of the REST protocol was acceptable since the connection closed once the metric value was received and stored by the Metric Collector as all derived composite metric values where computed by the Metric Collector and their updated values stored asynchronously in its time series database.

In a multi-level EMS it is not known *a priori* which components will need an updated metric value or a computed derived event. For instance, continuing the above example, the EMS component in a Cloud provider region could be the end-point for some of the metric values computed by the VM local EMS, whereas other metric values should be transferred directly to the EMS component in the Melodic platform. Only the EMS component using a metric value should receive it, and the EMS component should not need to really care about where the metric value is generated. This implies that the metric values should be distributed via a publish-subscribe mechanism.

The above considerations imply that the monitoring system is a major decision for the successful integration of the Melodic platform. Fundamentally, two choices existed:

1. Improve the home-grown monitoring system of PaaSage
2. Replace the monitoring system and the scalability rules of PaaSage with a standard EMS

Considering the general requirements on the Melodic platform and its long-term sustainability, the second option was adopted. This led to the requirement to properly select and implement a hierarchical EMS system for Melodic, and this is further discussed in Section 4.5.14.5.3.

2.3 List of issues and risks, with suggested mitigation actions

The ZeroMQ is a broker less queue-based middleware solution, which is very efficient in terms of performance, but with lack of support for transactions and monitoring.

We have identified a number of issues related to the used integration methods in the PaaSage and Cactos frameworks that we present in Table 2. The issues have been discovered through own

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

experience and general knowledge, as well as the testing process of PaaSage, while some of them have been described in PaaSage deliverables.

The table contains the following columns:

- Issue name – the unique name of the issue
- Issue description – brief description of the issue
- Mitigation actions – list of possible mitigation actions.

Table 2: List of integration issues in the PaaSage and Cactos projects

Issue name	Issue description	Mitigation actions
Point-to-point communication	Due to many components in the system, point-to-point communication increases the complexity of the whole solution.	Replace ZeroMQ due to lack of support for creating advanced queue and topic configuration with a different integration solution or use topics configuration to mimic more advanced types of communication. The second solution alternative could be only used as a workaround and is not recommended for more complex systems.
Asynchronous communication in some parts of the system	Use of asynchronous communication for all kinds of integration, even for those that require a synchronous one based on the respective (integration) requirements, increases possibility of faults and lowers the reliability of the solution.	Replace ZeroMQ as it does not support synchronous communication or implement a custom solution on top of ZeroMQ to mimic the synchronous communication using an asynchronous queue-based system.
No data transformation and canonical model[9]	Each component needs to interpret the data model in a specific way; this is error-proof and increases complexity.	Replace ZeroMQ as it does not support data transformation out of the box. It would be very difficult to implement and maintain the data transformation and canonical model handling on top of ZeroMQ.
Not possible to scale the integration layers or	Enterprise grade systems require the ability to scale and follow a HA configuration.	Replace ZeroMQ as it does not support HA configuration, or allow only for vertical scaling. Due to the design of ZeroMQ, it is not possible

[9] https://www.techopedia.com/definition/30598/canonical-data-model-cdm

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Issue name	Issue description	Mitigation actions
install them in HA configuration[10]		to scale horizontally. Also, HA configuration could be only deployed in active-passive mode.
There is no easy method to monitor the Control flow	Monitoring and troubleshooting is difficult.	Replace ZeroMQ due to lack of monitoring capabilities with another messaging system, or find a tool which allows for ZeroMQ-based monitoring
Messages can be lost as the ZeroMQ communication is unreliable	The reliability of the system is low.	Replace ZeroMQ due to lack of reliable transfer with another integration solution, due to lack of possibility to achieve reliability and support for transactions over ZeroMQ.
No retry-mechanism can be introduced within the components in case of connection problems	There is no support for retrying operations in case of error.	Replace ZeroMQ with another integration solution or implement a custom retry-mechanism

Most of the presented issues are related to the lack of capabilities and features in ZeroMQ. Therefore, an evaluation of other integration methods to find the best integration strategy for Melodic is needed.

3 Methodology of choosing integration and component adaptation strategy

This section contains the description of the methodology for choosing the integration and adaptation strategy for Melodic. The result of applying the described methodology is presented in Section 4.

For the selection of the most appropriate integration and adaptation strategy for the Melodic project, the following methodology has been used. This methodology has been devised according to our experience and the actual objectives that must be fulfilled:

1. The first step of the methodology is to identify the objectives and general requirements for the integration and adaptation strategy of the project, as well as the purpose of the

[10] http://searchdatacenter.techtarget.com/definition/high-availability

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

integration and alignment of the components. The requirements are identified separately for the Control Plane as well as the Monitoring Plane.

2. The second step is to research, review and evaluate typical integration methods used to integrate IT systems. There are plenty of such methods but – based on professional experience and knowledge – the most typical and suitable methods were chosen. This step is broken down into the following sub-steps:
    a. A research over state-of-the-art integration methods is conducted. A small set of the most suitable integration methods is then selected from the state-of-the-art.
    b. Each of the integration methods considered is compared against the fulfilment of the integration requirements for the Melodic project identified in the first step of the methodology. For each requirement per each method of integration the level of fulfilment is assigned. The estimated effort needed to implement a given integration strategy in Melodic project is also provided as a value in the range 1 ... 5, as explained in section 4. Lower values mean higher effort, so the scale is reversed. The reversed scale is used for easier comparison in the next point. The effort is related to the current architecture of the project; thus, the effort for the implementation of the already used integration method is minimal (adjustments only).
    c. After completing the previous step, a certain score is assigned to each method of integration. The score is computed by a weighted sum approach: in the first level, we compute the overall method score from the weighted sum of the scores calculated for each plane; in the second level, we apply a weighted sum of the partial scores of requirement fulfilment and the level of effort in order to compute the method score per each plane; in the third level, we calculate the requirement fulfilment partial score through dividing the sum of the points of the actual fulfilment of the method across all requirements, with the sum of the maximum points that a method can take over all requirements. The partial score of the level of effort is computed by dividing the actual evaluation value of the method divided by the maximum possible one (i.e., 5). For the evaluation of each integration requirement, we map the level of fulfilment of the requirement into the range 0 ... 5. In particular, fulfilled requirement maps to 5 points, a partially fulfilled one to 3 points and a non-fulfilled requirement to 0 points. The score for Control Plane has weight 0.6 and the score for the Monitor Plane has weight 0.4. The Control Plane is considered more important for the working of the whole platform.

       The calculation of the overall score for the methods is performed as follow:

       - *Partial_score_level_effort* = actual effort needed for method implementation divided by the maximum possible one (number 5).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

- *Partial_score_control_plane_req* = sum of fully fulfilled requirements for the Control Plane times 5 plus sum of partially fulfilled requirements for the Control Plane times 3.
- *Partial_score_monitor_plane_req* = sum of fulfilled requirements for the Monitor Plane times 5 plus sum of partially fulfilled requirements for the Monitor Plane times 3.
- Overall score for the method = [(*Partial_score_control_plane_req/65 * 0,75) + (Partial_score_level_effort * 0,25)*] * 0,6 + [(*Partial_score_monitor_plane_req/15 * 0,75) + (Partial_score_level_effort * 0,25)*] * 0,4

Please note that in order to apply the weighted sum approach, the respective partial scores have been mapped to the same set of reals ([0.0, 1.0]), thus performing a certain form of normalisation.

d. The methods of integration are ranked from the highest to the lowest overall score.

3. In this step, the selection of the best integration strategy for the Melodic project is performed. This step maps to the execution of the following two sub-steps:
   a. Verify selected integration method by two certified architects based on their experience and professional knowledge, to confirm the results of the quantitative assessment.
   b. In case of a blocking issue, the method with the second highest score is selected to be verified by experts and, thus, point 3.a. is repeated.
4. Based on the selected integration methods, the integration strategy for Melodic is determined.
5. Based on the chosen integration strategy, the adaptation strategy will be determined, as elaborated later in this deliverable.
6. The final step is the selection of the right and most suitable tools to implement the selected integration method in the Melodic project.

The above steps are summarised in Figure 2.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

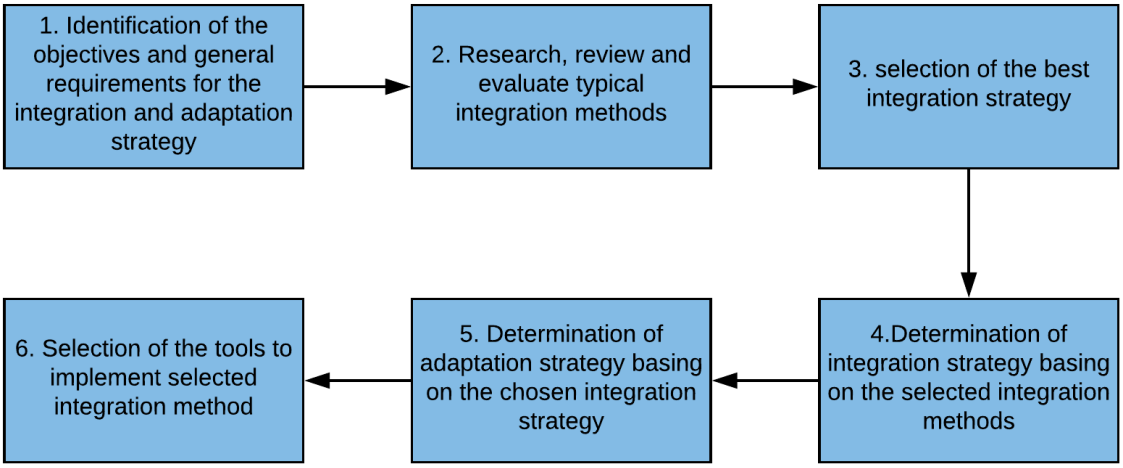

Figure 2: Diagram of methodology for choosing integration and adaptation strategy for the Melodic project

4 Methodology Application

In the following sections, we elaborate on how the methodology analysed in the previous section is applied in case of the Melodic project. The analysis is performed according to the structure of the methodology in a step-wise manner. Each step is analysed in its own section.

4.1 Requirements Collection

The main direction of work for the Melodic project is the integration and adaptation of the underlying frameworks PaaSage and Cloudiator, as well as the introduction of the support for Big Data management (data awareness and locality). For this reason, the integration and adaptation strategy for Melodic should be carefully evaluated and precisely designed.

The fundamental objective of integration in Melodic is to achieve seamless cooperation of the components, independent from their underlying frameworks. Such an approach is very important for this project due to the use of different integration methods in the key underlying frameworks:

- The PaaSage project has been built using over 20 components; 11 of these components will be re-used and will be integrated. These components are integrated using ZeroMQ (asynchronous).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

- The Cloudiator[11], part of the Cactos project, has also a certain component structure, but the features are exposed by one, unified API. The components of Cloudiator are integrated by a REST API (synchronous).

In the above projects, there are at least four different methods of integration used:

- Asynchronous, queue-based communication (e.g., the Metasolver uses ZeroMQ[12] to communicate with the solvers)
- Synchronous, via a database model repository (e.g., CP Generator exchanges the CP model with solvers)
- Synchronous, via a REST API[13] (e.g., in case of the Adapter's integration with Cloudiator)
- Asynchronous, file-based (e.g., in case of the generation of input files for the LA Solver)

Furthermore, there are two separate layers of integration, each with its own purpose and requirements for integration:

- Control Plane – integration layer for controlling the flow of the process/actions in the system
- Monitoring Plane – integration layer for gathering, processing and storing all the monitoring events and respective measurements.

This variety of used integration methods, planes and components – along with efforts to achieve the most efficient and seamless integration of all components – has resulted in the creation of a unified method of integration.

The high-level integration and adaptation requirements for each plane are listed below. A more detailed description of these requirements is provided in the D5.04 "Integration and testing requirements" deliverable. These requirements are listed and characterised by an ID which indicates, through its suffix, the actual plane on which the requirement is dedicated (CP – Control Plane, MP - Monitor Plane, CMP - both planes).

The integration and adaptation requirements for the Control Plane are the following:

- *Req1CP – Reliability:* to achieve a reliable flow of the invoked operations, with full control over an operation's execution and returned results.
- *Req2CMP – Performance:* for the Control Plane, performance is not a critical issue, but the integration layer should achieve a sufficient level of performance.
- *Req3CP – Scalability:* ability to scale the integration layer both horizontally and vertically.
- *Req4CP – High availability:* support for highly available, multi-node configuration, at least in active-passive mode – active configuration will be an additional benefit.

[11] https://www.uni-ulm.de/en/in/omi/research/results/cloudiator/
[12] http://zguide.zeromq.org/page:all
[13] http://searchcloudstorage.techtarget.com/definition/RESTful-API

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

- *Req5CP – Flexible orchestration*: the ability to easily set up and reconfigure the orchestration of method invocations of underlying components. It should be possible to configure such orchestration without the need to code and recompile the whole platform.
- *Req6CP – Support for synchronous and asynchronous communication*: the selected integration solution should support both synchronous and asynchronous communication methods with an easy way to switch from one to the other.
- *Req7CP – Security:* support for both authentication and authorisation, as well as definition of the access rights to invoke a given operation.
- *Req8CP – Monitoring:* the ability to monitor all operations invoked on the integration layer, with a configurable level of detail.
- *Req9CP – Logging:* configurable and easy usage of a single logging mechanism for all the invoked operations.
- *Req10CP – Support for different integration protocols:* the chosen solution should have support for the most commonly used integration protocols; at least SOAP, REST and the Java Message Service (JMS)[14].
- *Req11CP – Data model transformation:* ability to perform complex data model transformations.
- *Req12CP – Exception handling and support for retrying:* unified exception handling and retrying of operations.
- *Req14CMP – Easy to use:* the integration method should be relatively simple as it needs to be executed by every single Melodic application.

The integration and adaptation requirements for the Monitoring Plane are the following:

- *Req2CMP – Performance:* due to the high volume of messages being exchanged, achieving high performance is a crucial requirement.
- *Req13MP – Low resource usage:* The Monitoring Plane is used by all installed applications to properly deliver metric values, so low usage of resources is very important.
- *Req14CMP – Easy to use:* the integration method should be relatively simple as it needs to be executed by every single Melodic application.

A more detailed list and description of the above requirements, with explanations about the purpose of each requirement for the Melodic platform, is provided in D5.04 the "Integration and testing requirements" deliverable. In Table 3, we provide a summary of the requirements collected along with their mapping to the respective planes of the Melodic platform.

[14] https://www.techopedia.com/definition/4298/java-message-service-jms

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 3: The relation of integration requirements to the two planes

Req. Id	Requirement	Which plane is affected by requirement (Control Flow, Monitoring, All)
Req1CP	Reliability	Control Flow
Req2CMP	Performance	All
Req3CP	Scalability	Control Flow
Req4CP	High availability	Control Flow
Req5CP	Flexible orchestration	Control Flow
Req6CP	Support for synchronous and asynchronous communication	Control Flow; for the Monitoring Plane only asynchronous communication
Req7CP	Security	Control Flow
Req8CP	Monitoring	Control Flow
Req9CP	Logging	Control Flow
Req10CP	Support for different integration protocols	Control Flow
Req11CP	Data model transformation	Control Flow
Req12CP	Exception handling and support for retrying	Control Flow
Req13MP	Low resource usage	Monitoring
Req14CMP	Easy to use	All

4.2 Integration Method Research and Review

In this section, we analyse the application of the 2nd methodology step concerning the integration method research, review and evaluation. Our focus is on explaining why certain integration methods have been picked up from the state-of-the-art, what they stand for and what are their main pros and cons, and finally how well they fulfil the integration requirements collected based on the previous methodology step.

There are many definitions of the integration of IT systems [2] [3] [4]. For the purpose of this document, the following definition of integration [2] will be used: the interoperability between separate IT systems or components. The purpose of the integration is to allow the interoperability between components and systems according to the defined requirements. In the following subsections, the most typical types of integration are described, along with a summary of their strengths and weaknesses.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

For each type of integration method, the given method is compared to the requirements for integration. A given requirement is first evaluated so as to determine its fulfilment. The possible levels of requirement fulfilment by a particular method are discussed below:

- *Fulfilled* – a given requirement is completely fulfilled by the particular method, without a necessity to implement custom code or to use any workaround. This maps to a quantitative score of 5 for the respective method based on this requirement.
- *Partially fulfilled* – a given requirement is partially fulfilled by the particular method; there could be a need to either implement custom code, to use a workaround or to handle the requirement at the local level and, thus, not at the integration level. The custom code or workaround does not need significant effort to be implemented, but it increases the complexity of the solution and it might have some negative impact on performance - but not a severe one. This maps to a quantitative score of 3 for the respective method based on this requirement.
- *Not fulfilled* – a given requirement is not fulfilled by the particular method. Thus, there is no possibility to use custom code or any workaround. The implementation of custom code or workaround may require significant effort and increases complexity of the whole solution to an unacceptable level. It can also severely impact performance in a quite negative way. This maps to a quantitative score of 0 for the respective method based on this requirement.

In addition, for each integration method, the overall estimation of the complexity of implementation in the Melodic project has been presented. In order to quantitatively compare the integration methods reviewed, we use an indication of the implementation effort required using range values from 1 to 5 (1 for the highest effort and 5 for the lowest effort).

In the following subsections (4.2.1 to 4.2.4) we evaluate, by also providing respective justifications, the level of fulfilment of integration requirements, and the level of complexity and effort, for each integration method. In the end, an overview table is presented which summarises the evaluation results across all methods and requirements.

4.2.1 Point-to-point integration

Point-to-point integration is a direct connection between two systems, without any layer in between. The systems usually are connected in a synchronous manner and there is no common data model transformation layer. The point-to-point integration is the most expensive integration method [2] for medium and large number of components and systems that need to be integrated. For a very small number of components and systems it could be acceptable, but for a medium or large number of components and systems, the number of connections between systems increases dramatically. A more detailed analysis of point-to-point communication is provided in [2].

In Figure 3 we present a typical point-to-point integration of IT systems.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

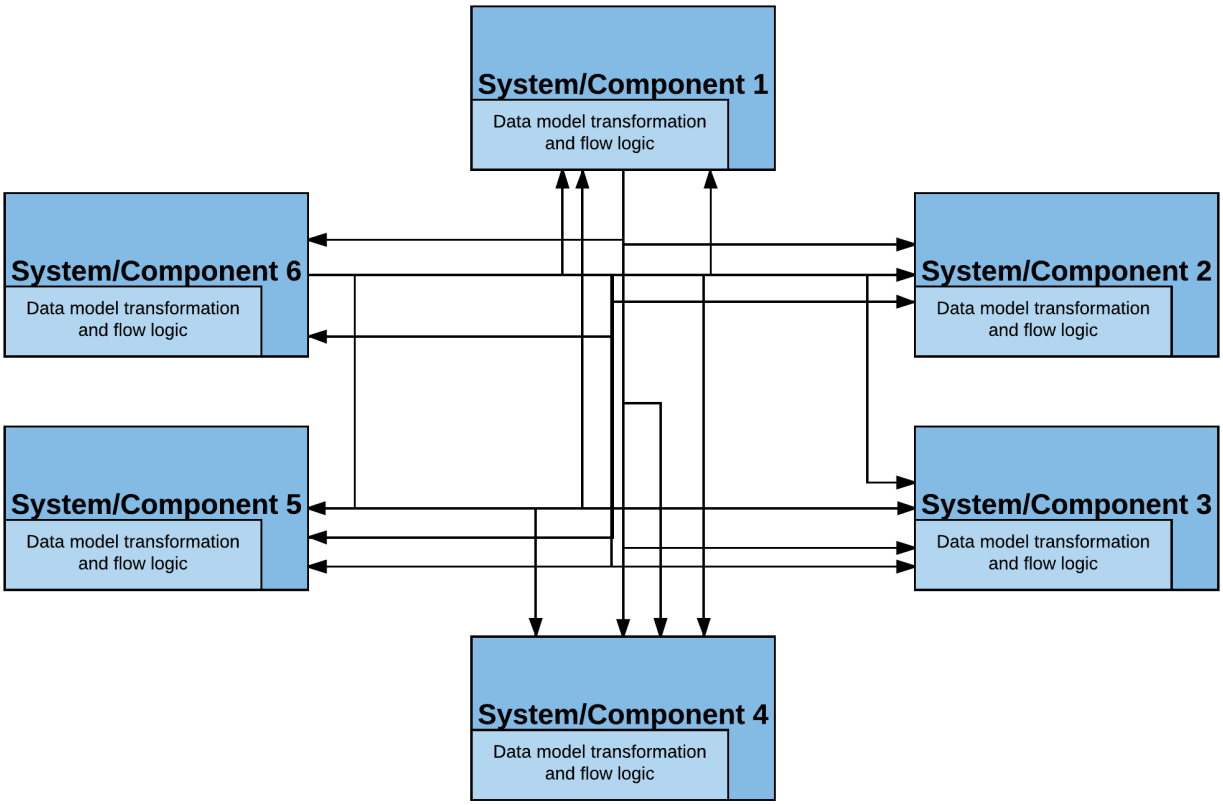

Figure 3: Point-to-point architecture

In Table 4, the evaluation of the point-to-point integration method is presented.

Table 4: Fulfilment of integration requirements by the Point-to-point integration method

Req. Id	Requirement	Fulfilment by given integration method	Comments
Req1	Reliability	Not fulfilled	Reliability of all the integrated systems depends on the minimum (individual) reliability across all the systems, e.g., a weak point of one system impacts equally other integrated systems.
Req2	Performance	Partially fulfilled	Performance depends on the performance of each system and it cannot be increased by scalability of the integration layer. However, as the solution is simple enough, it is not penalised with respect to its performance; that's the reason for partial fulfilment. So, it is only limited by the performance of the respective sub-systems involved.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Req. Id	Requirement	Fulfilment by given integration method	Comments
Req3	Scalability	Not fulfilled	Due to point-to-point communication, there is no possibility to scale the whole solution. Introduction of scalability needs custom implementation, which is very difficult to maintain, requires significant effort and is thus not recommended.
Req4	High availability	Not fulfilled	Due to point-to-point communication, there is no possibility to create a complete HA solution; the process needs custom implementation and configuration, but such a solution is very difficult to maintain and extend. It also requires significant effort to be introduced.
Req5	Flexible orchestration	Not fulfilled	It is not possible to use external orchestration in this case, due to the lack of any external integration/orchestration layer.
Req6	Support for both synchronous and asynchronous communication	Not fulfilled	There is no built-in support for both types of communication; the support needs to be custom implemented, which is very difficult to maintain and extend.
Req7	Security	Not fulfilled	Implemented at each interaction point between systems, there is no centralised security control and maintenance; security has to be implemented at each system level, not the integration layer level.
Req8	Monitoring	Partially fulfilled	Monitoring is established at each of the integrated system's level. There is no centralized solution, which means that again a great effort will be required to implement it.
Req9	Logging	Partially fulfilled	Logging is supported at each of the integrated system's level. There is no centralised solution, which means that again a great effort will be required to implement it.
Req10	Support for different integration protocols	Not fulfilled	Each of the integration protocols needs to be realised at each integration level of the overall system.
Req11	Data model transformation	Not fulfilled	There is no common (domain-specific) data model and ability to transform data models in a unified manner.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Req. Id	Requirement	Fulfilment by given integration method	Comments
Req12	Exception handling and support for retrying	Partially fulfilled	Handled at the level of each integrated system.
Req13	Low resource usage	Fulfilled	Resource usage is low due to the lack of a separate integration layer in this case.
Req14	Easy to use	Fulfilled	No additional work is needed for integration except from invoking methods of the other system. In case of many systems, the complexity of the solution(s) is very difficult to maintain.

The estimated effort level needed to implement this method for the Melodic project is 3. The estimated effort level is based on the complexity of the integration method as well as the scope of changes needed for introducing the method for the PaaSage and Cactos projects based on related expertise.

4.2.2 Queue-based middleware integration

Message-oriented middleware uses messages transported in a queue as means of communication. The message queuing model allows messages to be stored in a queue where they may be picked up by an application at any time. Thanks to that, the communication is reliable, but the only supported method of communication is asynchronous communication. A more detailed analysis of the queue-based middleware, also known as message-oriented middleware, is provided in [4].

In Figure 4 we present a typical queue-based integration of IT systems.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

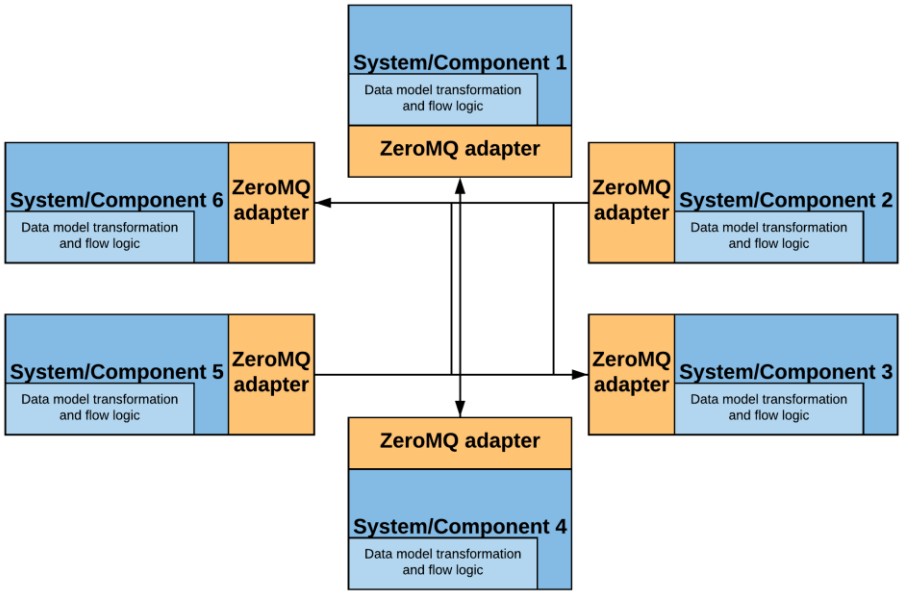

Figure 4: Queue based middleware architecture

Currently, in the PaaSage framework, ZeroMQ is used as an integration method. As we need to guarantee a lower effort in the transitioning to the new integration realisation in Melodic, ZeroMQ is considered only for the evaluation, although it is not a typical implementation of the Queue-based middleware.

In Table 5 the evaluation of the ZeroMQ integration method is presented.

Table 5: Fulfilment of integration requirements by the Queue based integration method

Req. Id	Requirement	Fulfilment by given integration method	Comments
Req1	Reliability	Partially fulfilled	ZeroMQ, due to its design, is not fully reliable.
Req2	Performance	Fulfilled	Performance is high due to asynchronous communication and efficient method of communication.
Req3	Scalability	Partially fulfilled	ZeroMQ - due to its design without a message broker - is more difficult to scale.
Req4	High availability	Partially Fulfilled	ZeroMQ - due to its design without a message broker - makes it hard to create a HA configuration.
Req5	Flexible orchestration	Not fulfilled	It is not possible to use external orchestration in this case due to asynchronous communication.
Req6	Support for both	Not fulfilled	This method of integration, by design, supports only asynchronous communication. There is no built-in

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Req. Id	Requirement	Fulfilment by given integration method	Comments
	synchronous and asynchronous communication		support for synchronous communication; the support needs to be custom implemented, which is very difficult to maintain and extend.
Req7	Security	Fulfilled	Implemented at the integration level.
Req8	Monitoring	Partially fulfilled	Monitoring needs some custom implementation in the case of ZeroMQ
Req9	Logging	Partially fulfilled	Logging needs some custom implementation in the case of ZeroMQ
Req10	Support for different integration protocols	Not fulfilled	Supports by design only asynchronous integration protocols. The requirement is not fulfilled, because the whole area of synchronous methods of communications and protocols is not covered.
Req11	Data model transformation	Not fulfilled	ZeroMQ does not support this at all. It is very difficult to implement full canonical model transformation with only a queue-based solution, as it usually requires additional layers/solutions.
Req12	Exception handling and support for retrying	Partially fulfilled	Supported for asynchronous communication.
Req13	Low resource usage	Fulfilled	ZeroMQ has very low resource requirements.
Req14	Easy to use	Fulfilled	There are common patterns on how to use this type of integration. Installation and maintenance is also quite easy to set up and administer.

The estimated effort needed to implement this method for the Melodic project is 4, as it is already implemented for PaaSage. This effort does not cover the implementation of the additional fixes and custom improvements to ZeroMQ; it assumes using features available in ZeroMQ as a standard feature.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

4.2.3 EAI/ESB based integration

The Enterprise Service Bus[15] architecture uses a central messaging backbone (bus) for message propagation. Systems publish messages to this bus using adapters. These messages flow to any subscribing application that uses the same message bus. These subscribing applications should have adapters in order to receive messages from the bus, and transform them into a format required by them [5]. A more detailed elaboration and research related to the integration approach using EAI/ESB could be found in[16] [6]. A typical ESB solution implements support for both synchronous and asynchronous communication. Asynchronous communication is usually implemented using a queue-based middleware (for example, MuleESB default broker uses ActiveMQ for asynchronous communication).

In Figure 5 we present a typical EAI/ESB integration of an IT system.

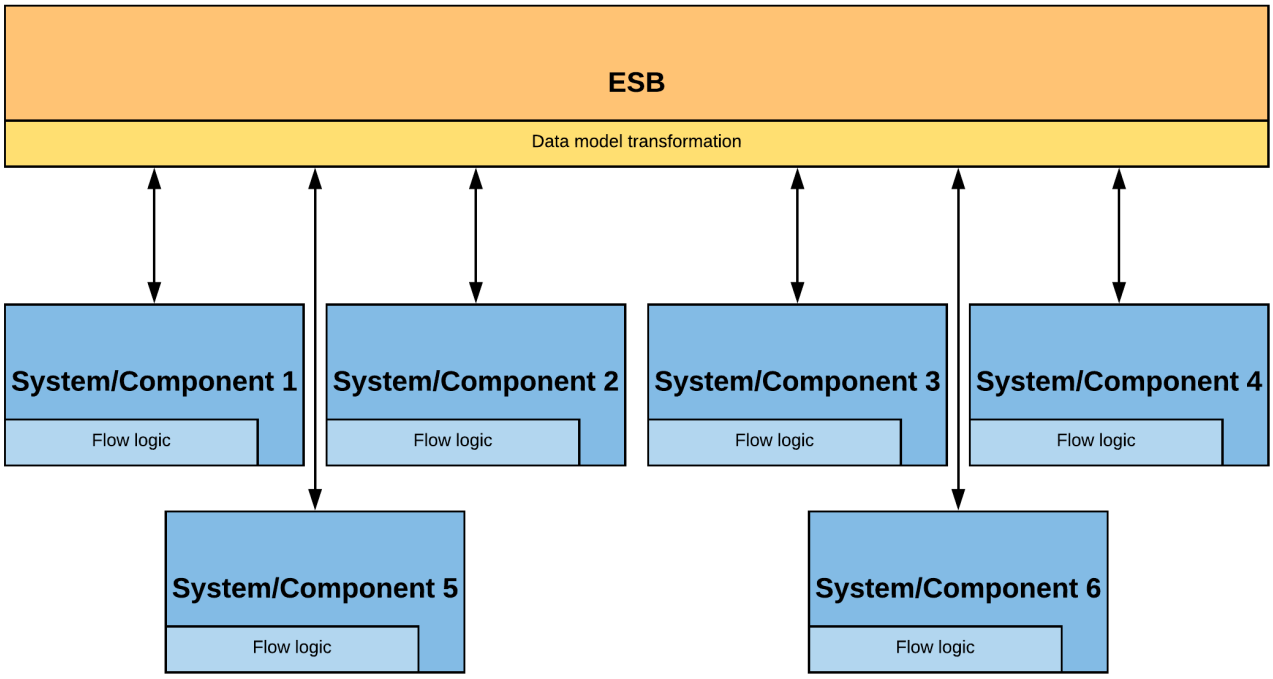

Figure 5: ESB based integration architecture

In Table 6 the evaluation of the ESB integration method is presented.

[15] http://searchdatacenter.techtarget.com/definition/high-availability
[16] https://www.techopedia.com/definition/1506/enterprise-application-integration-eai

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 6: Fulfilment of integration requirements by the ESB based integration method

Req. Id	Requirement	Fulfilment by given integration method	Comments
Req1	Reliability	Fulfilled	This type of integration is designed to be highly reliable due to the ability to set up a multiple node installation.
Req2	Performance	Fulfilled	Performance depends on the complexity of integration logic, but this requirement is fulfilled, since there is a possibility to build scalable solution.
Req3	Scalability	Fulfilled	Most of the ESB implementations have the ability to scale both horizontally and vertically.
Req4	High availability	Fulfilled	Most of the ESB implementations have the ability to be set up in HA configuration, where there is support for both active-passive and active-active modes.
Req5	Flexible orchestration	Not fulfilled	It is not possible to use flexible orchestration with ESB only. It requires external tools; this is described as a separate integration method (ESB with BPM orchestration, see next section).
Req6	Support for both synchronous and asynchronous communication	Fulfilled	Most of the ESB implementations have support for both methods of communication.
Req7	Security	Fulfilled	Most of the ESB implementations have support for centralized security management.
Req8	Monitoring	Fulfilled	Most of the ESB implementations have support for centralized monitoring.
Req9	Logging	Fulfilled	Most of the ESB implementations have support for centralized logging.
Req10	Support for different integration protocols	Fulfilled	Support for different integration protocols is a fundamental assumption for each ESB solution.
Req11	Data model transformation	Partially fulfilled	Supported with some limitations [4].
Req12	Exception handling and	Partially fulfilled	Most of the ESB implementations have support for exception handling and retrying. Nevertheless, it is not

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Req. Id	Requirement	Fulfilment by given integration method	Comments
	support for retrying		possible to handle exceptions and retrying at the business logic level.
Req13	Low resource usage	Partially fulfilled	The resource usage depends on the complexity of the integration logic, but it is usually higher than simpler solutions.
Req14	Easy to use	Partially fulfilled	The integration of a new system/component with ESB is very easy. The configuration and administration of an ESB requires more effort, but usually is supported by dedicated tools built in the platform.

The estimated effort needed to implement this method for the Melodic project is 2. It is caused by the need to deploy the whole ESB as well as change the integration technologies/protocols currently used.

4.2.4 EAI/ESB integration with BPM orchestration

EAI/ESB integration with Business Process Management[17] (BPM) orchestration is the most flexible integration method currently used for systems integration, based on the features being provided. This type of integration is similar to EAI/ESB integration. The only difference is that business processes (BPs) are used for orchestrating method invocation, instead of coding this orchestration in each particular component. Based on this fact, it is much more flexible to change the flow of the process, and it is possible to use the same service exposed by a given component in various processes and features of the system. A more detailed elaboration and research for using BPM to orchestrate service invocation is provided in [1].

A typical EAI/ESB integration with BPM orchestration is presented in Figure 6.

[17] http://searchcio.techtarget.com/definition/business-process-management

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

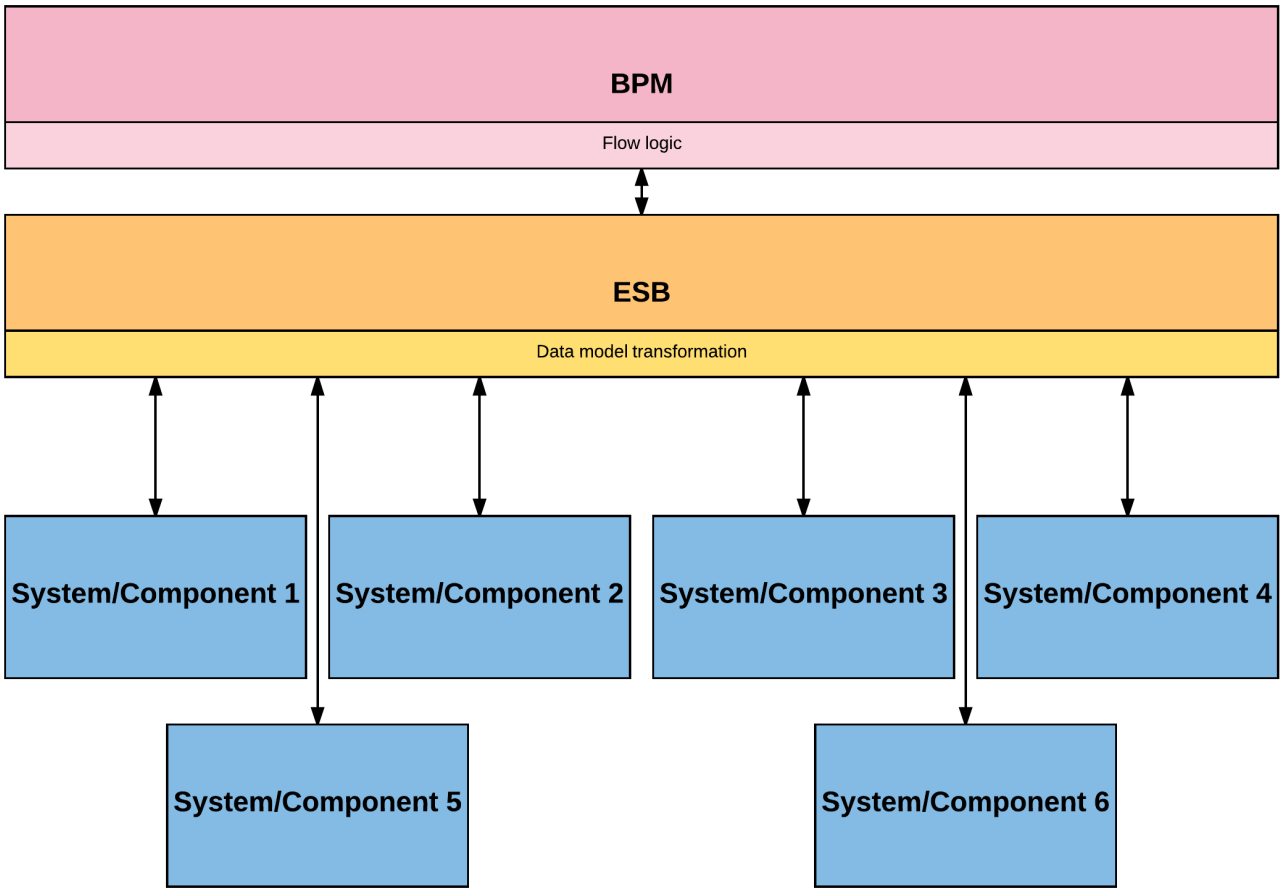

Figure 6: ESB based integration with BPM orchestration

In Table 7 the evaluation of the ESB with BPM integration method is presented.

Table 7: Fulfilment of integration requirements by the ESB based with BPM orchestration integration method

Req. Id	Requirement	Fulfilment by given integration method	Comments
Req1	Reliability	Fulfilled	This type of integration is designed to be highly reliable.
Req2	Performance	Fulfilled	Performance depends on the complexity of the integration logic, but based on the ability to build a scalable solution, the performance requirement is fulfilled.
Req3	Scalability	Fulfilled	Most of the ESB implementations have the ability to scale both horizontally and vertically.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Req. Id	Requirement	Fulfilment by given integration method	Comments
Req4	High availability	Fulfilled	Most of the ESB implementations have the ability to be set up in HA configuration, where both active-passive and active-active modes are supported.
Req5	Flexible orchestration	Fulfilled	For this type of integration method, flexibility of orchestration is achieved by introducing BPM flows for orchestration.
Req6	Support for both synchronous and asynchronous communication	Fulfilled	Most of the ESB implementations have support for both methods of communication.
Req7	Security	Fulfilled	Most of the ESB implementations have support for centralized security management.
Req8	Monitoring	Fulfilled	Most of the ESB implementations have support for centralized monitoring.
Req9	Logging	Fulfilled	Most of the ESB implementations have support for centralized logging.
Req10	Support for different integration protocols	Fulfilled	Support for different integration protocols is a fundamental assumption for each ESB solution.
Req11	Data model transformation	Fulfilled	Fully supported ability to configure mapping between data models (domain and canonical) at the ESB level.
Req12	Exception handling and support for retrying	Fulfilled	Most of the ESB implementations have support for exception handling and retrying. It is also possible to handle exceptions and retrying at the business logic level.
Req13	Low resource usage	Partially fulfilled	The resource usage depends on the complexity of the integration logic and usually is higher than for simpler solutions.
Req14	Easy to use	Partially fulfilled	The integration of new systems/components with ESB is very easy. The configuration and administration of ESB requires more effort but is usually supported by dedicated tools built in the platform.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

The estimated effort needed to implement this method for the Melodic project is 1. It is caused by introducing an additional layer, changes in the components logic, the additional integration effort as well as the preparation of the BPs.

4.2.5 Overall Evaluation Results

Table 8 summarises the evaluation results for the integration methods examined across all integration requirements.

Table 8: Summary of requirement fulfilment for all integration methods considered

Req. Id	Requirement/Integration method	Point-to-point	Queue based	ESB	ESB with BPM
Req1	Reliability	Not fulfilled	Partially fulfilled	Fulfilled	Fulfilled
Req2	Performance	Partially fulfilled	Fulfilled	Fulfilled	Fulfilled
Req3	Scalability	Not fulfilled	Partially fulfilled	Fulfilled	Fulfilled
Req4	High availability	Not fulfilled	Partially fulfilled	Fulfilled	Fulfilled
Req5	Flexible orchestration	Not fulfilled	Not fulfilled	Not fulfilled	Fulfilled
Req6	Support for both synchronous and asynchronous communication	Not fulfilled	Not fulfilled	Fulfilled	Fulfilled
Req7	Security	Not fulfilled	Fulfilled	Fulfilled	Fulfilled
Req8	Monitoring	Partially fulfilled	Partially fulfilled	Fulfilled	Fulfilled
Req9	Logging	Partially fulfilled	Partially fulfilled	Fulfilled	Fulfilled
Req10	Support for different integration protocols	Not fulfilled	Not fulfilled	Fulfilled	Fulfilled
Req11	Data model transformation	Not fulfilled	Not fulfilled	Partially fulfilled	Fulfilled
Req12	Exception handling and support for retrying	Partially fulfilled	Partially fulfilled	Partially fulfilled	Fulfilled

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Req. Id	Requirement/Integration method	Point-to-point	Queue based	ESB	ESB with BPM
Req13	Low resource usage	Fulfilled	Fulfilled	Partially fulfilled	Partially fulfilled
Req14	Easy to use	Fulfilled	Fulfilled	Partially fulfilled	Partially fulfilled

4.2.6 Method Score Calculation

This is the second sub-step of the integration method research, review, and evaluation step, where the calculation of the overall score of each integration method per plane is provided. Before supplying an explanation about how the scores were calculated, we provide a summary in Table 9 which shows the mapping of the fulfilment level of each requirement per each method to the 0 … 5 range. In addition, an overall number of points per plane is calculated in the very last rows of the table (along with an indication about what should have been the ideal number of points per plane in parenthesis).

Table 9: Summary of the integration method evaluation

Req. Id	Requirement name\ Integration method	Point-to-point	Queue based	ESB	ESB with BPM
Req1	Reliability	0	3	5	5
Req2	Performance	3	5	5	5
Req3	Scalability	0	3	5	5
Req4	High availability	0	3	5	5
Req5	Flexible orchestration	0	0	0	5
Req6	Support for both synchronous and asynchronous communication	0	0	5	5
Req7	Security	0	5	5	5
Req8	Monitoring	3	3	5	5
Req9	Logging	3	3	5	5
Req10	Support for different integration protocols	0	0	5	5
Req11	Data model transformation	0	0	3	5
Req12	Exception handling and support for retrying	3	3	3	5

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Req. Id	Requirement name\ Integration method	Point-to-point	Queue based	ESB	ESB with BPM
Req13	Low resource usage	5	5	3	3
Req14	Easy to use	5	5	3	3
	Estimated effort	3 (/5)	4 (/5)	2 (/5)	1 (/5)
	SUM OF POINTS for Control and Data Flow	17 (/65)	33 (/65)	54 (/65)	63 (/65)
	SUM OF POINTS for Monitoring Plane	13 (/15)	15 (/15)	13 (/15)	13 (/15)

In Table 10 we present the calculation of the overall scores per plane (covering the 2nd level). We assume that the level of fulfilment has a greater relative importance than the level of effort. This maps to assigning a weight of 0.75 to the level of fulfilment and 0.25 to the level of effort. In this respect, the overall score per plane and integration method is calculated according to the following formula: $score_{ij}=0.75*req_{ij}+0.25*effort_{ij}$, where $score_{ij}$ denotes the score of method i over plane j, while req_{ij} and $effort_{ij}$ denote the respective partial scores for the level of requirement fulfilment and effort, respectively, for the current method and plane pair. The respective results are imprinted in the table. Finally, the Overall Score for each method is calculated, based on the scores for the Control Flow Plane and the Monitoring Plane, with weights 0.6 and 0.4, respectively. A slightly higher weight is assigned to the Control Flow Plane, due to the more importance of that plane for the whole solution.

Table 10: Calculation of the overall scores per plane

Integration Method	Partial score for required effort.	Partial score for the requirement fulfilment level for the Control Plane	Overall Score for the Control Plane	Partial score for the requirement fulfilment level for Monitoring Plane	Overall Score for the Monitoring Plane	Overall Score for all the planes (0.6 weight for the Control Plane and 0.4 for the Monitoring Plane
Point-to-Point	3/5=0.6	17/65 = 0.26	0.34	13/15=0.86	0.8	0.52
Queue-based	4/5=0.8	33/65=0.50	0.58	15/15=1.0	0.95	0.73
ESB	2/5=0.4	54/65=0.86	0.72	13/15=0.86	0.75	0.73
ESB+BPM	1/5=0.2	63/65=0.96	0.78	13/15=0.86	0.7	0.75

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Based on the results in the above table, we nominate ESB+BPM as the best integration method for the combined Control Flow Plane and Monitoring Plane, as well as individually for the first plane. The Queue-based method is ranked as the best for the Monitoring Plane individually.

Using one integration method is the most preferred approach, due to less effort for implementation and maintainability of the system in the future. Also, it is a less complex and error prone approach. To achieve uniformity of integration method for each plane the ESB+BPM is then selected as an integration method for Melodic. To confirm this selection, two experts have been asked for verification of the choice made, as further detailed in the next section.

4.3 Integration Strategy selection verification

Based on the results of the previous methodology step, the selected integration solution is to be verified by experts. Thus, the goal of this methodology step is to confirm the chosen option. To this end, the professional opinion from two certified software architects, one being a Certified TOGAF Architect with specialization in enterprise grade solutions and the second being a AWS Certified Solution Architect with specialization in cloud solutions, was initially requested and then considered in order to reach the final verdict, i.e., to make the final choice over the ranked list of integration methods. In this respect, this section is separated into two subsections: the first indicates the opinion received from the two expects, while the second analyses the final decision taken.

4.3.1 Expert Recommendation

1. TOGAF Architect recommendation

Based on the requirements of the Melodic system, especially the focus on providing a highly customized solution which could be exploited by use case applications, the first expert recommends the usage of ESB as an integration method. Such a choice will make possible the creation of a highly scalable and reliable solution, which could be extended in the future, according to new user requirements and business needs. Using BPM for service orchestration allows to create a very flexible solution, which will minimize the cost of future changes and the integration effort for incorporating new components and systems as well as the overall total cost of ownership. The ESB/BPM combination is currently widely used for newly designed systems in the financial, insurance, telecom and other industries, as the most innovative and flexible way of system integration.

2. AWS Architect recommendation

The Melodic system, as a multi-cloud platform, should be aligned with the architecture of typical cloud computing applications, by relying on an as flexible as possible solution that can be easily adapted for the cloud. Point-to-point integration is the oldest method of integration, completely

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

not applicable to Cloud Computing due to its lack of flexibility. The chosen integration method should natively support the REST API over the HTTP protocol, as the mostly used solution for Cloud Computing applications. So, only ESB and ESB with BPM are the applicable methods of integration for that kind of solution. As such, the ESB with BPM is recommended as the most flexible method from the two.

4.3.2 Final selection of the integration strategy

Based on the results of the evaluation of each integration method against the integration requirements of the Melodic project, along with the professional recommendation of two certified architects, the ESB with BPM orchestration method of integration was selected as the integration strategy. This method achieved the highest ranking for fulfilment of requirements and enjoyed two positive professional recommendations.

4.4 Melodic platform adaptation strategy

The adaptation strategy is derived and closely linked with the integration strategy. Based on the selected integration strategy, the adaptation strategy for Melodic will be the following:

- All the components will be integrated based on the decided integration strategy and method.
- The prioritized list of changes that will be applied to the components of the underlying projects are described in deliverable D5.02 "Updates to OSS frameworks".
- The structure of the repositories will be aligned as described in the Confluence of the Melodic project[18] and will be reported in D5.03 "Security requirements & design" deliverable.
- The unit and integration tests for each of the components should be prepared as described in the D5.6 "Test Strategy" deliverable.
- The initial and final architecture of Melodic as described in D2.1 "System specification" and D2.2 "Architecture and initial feature definition" deliverables, respectively, will be respected by both the integration and adaptation strategies, and will be used as a baseline for any adaptation and modification performed.

4.5 ESB, BPM and Monitoring implementation

Based on the chosen integration strategy for Melodic, ESB integration with BPM, the following subsections focus on evaluating possible ESB and BPM implementations, which could be used in the Melodic project. Also, the new Monitoring implementation is initially covered.

[18] https://confluence.7bulls.eu/display/MEL/Melodic

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Due to the licensing model of the Melodic project, open-source licensing, only open-source solutions have been evaluated as possible implementations for both ESB and BPM.

4.5.1 ESB implementation

For the ESB implementation, three possible solutions have been evaluated:

- ServiceMix[19] – a high performance and available integration solution, being the most mature and stable one.
- MuleESB[20] – the most innovative solution, especially in the cloud computing area, with an easy to use GUI and possible, additionally paid, support from MuleSoft.
- WSO2[21] ESB – an open-source, dynamically developed integration solution, supported by the WSO2 technology provider.

The second and third solutions have also enterprise versions, which are not open-source. After carefully evaluating each option, summarised in Table 11, MuleESB has been chosen as the most suitable ESB implementation for the Melodic project for the following reasons:

- It is a stable and reliable solution, supported by MuleSoft, with plenty of documentation and online courses
- Supports the cloud computing model
- Rich and easy to use UI for configuration and management
- Implementation of many integration patterns

Table 11: Choosing ESB implementation

Criterium	ServiceMix	MuleESB	WSO2 ESB
Stable and reliable solution	Yes	Yes	Yes
Cloud computing support	No	Yes	Yes
Easy UI	No	Yes	No
Support of different integration patterns	No	Yes	Yes

4.5.2 BPM implementation

For the BPM implementation, there are four possible solutions that have been evaluated:

[19] http://servicemix.apache.org/docs/5.x/user/what-is-smx4.html
[20] https://www.mulesoft.com/resources/esb/what-mule-esb
[21] http://wso2.com/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

- Activiti[22] – one of the oldest and most mature open-source BPM implementations
- jBPM[23] – also a mature and stable BPM implementation, developed by Jboss[24], with integration support for the business rule server Drools[25]
- Camunda – a mature and more robust implementation of BPM, which does not require the whole Jboss stack to work.
- Flowable[26] – the newest solution, developed by a team of former Activiti developers.

On a first look, Activiti looked like the most promising solution. However, after evaluation and verification of the development roadmap and taking into account the fact that the Activity development team has split (the core of the development team migrated to the Flowable project), Camunda has been chosen as the BPM implementation for the Melodic project. The Flowable project is not fully mature for now, so it cannot accomplish the requirements of the Melodic Project. The jBPM from Jboss requires the whole stack of the Jboss technology, which complicates the implementation of the project. Key advantages of choosing Camunda are as follows:

- Lightweight implementation which is easy to deploy and maintain.
- Full support for the REST communication protocol.
- Easily available docker images, which allow for fast deployment.
- Low level of dependencies to other projects, which allows for easier upgrades and maintainability in the future.

Table 12 highlights the superiority of Camunda based on the 4 aforementioned criteria.

Table 12: Choosing BPM implementation

Criterium	Activity	jBPM	Camunda	Flowable
Easy maintenance and deployment	Yes	No	Yes	Yes
REST support	Yes	Yes	Yes	Yes
Docker images availability	No	Yes	Yes	No
Easy upgrade and maintainability	No	No	Yes	No

4.5.3 Monitoring component implementation

The monitoring capabilities for the modern, cross and multi cloud systems are considered critical elements to successfully deploy, reconfigure, and maintain them. As presented with examples in [7] the number of monitoring events and the ability to process them efficiently is a critical issue

[22] https://www.activiti.org/about
[23] https://bpm.com/what-is-bpm
[24] https://www.techopedia.com/definition/3525/jboss-application-server-jboss-as
[25] https://www.drools.org/
[26] http://www.flowable.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

for such systems. As estimated in the same work, the number of generated events that should be intercepted and processed for maintaining the appropriate service levels, could be bigger than 10.000 events per second for large scale, big data, distributed systems.

Consequently, the monitoring mechanism of the Melodic platform becomes crucial for the overall solution. It enables the continuous monitoring and metrics gathering generated by all the deployed application components. Continuous monitoring and optimization are the unique features of Melodic, comparing for example to other TOSCA-based solutions. The reason for this is that these Melodic features are based on the Models@run.time [8] approach. This approach dictates the initial creation and constant update of models that drive, in their turn, the continuous optimization of the application, using both current and historical monitoring data. Monitoring of cross and multi-cloud systems, allows for flexible and adaptive behaviour which is a necessity for large organizations which seek enterprise system grade quality when trying to exploit the clouds. As argued in [9], the use of a flexible monitoring mechanism is the most appropriate choice for driving the deployment and reconfiguration of the modern, distributed IT systems.

As shown by the discussion of the PaaSage approach in Section 2.2, the centralised Metric Collector component poses limitations with respect to scalability and performance for monitoring Big Data-intensive application components. The use of a central metric collection point may result in a high network bandwidth used just for monitoring purposes. In addition, message flooding scenarios cannot be avoided, exactly because of this centralised approach. With respect to internal implementation details of the PaaSage Metric Collector, we have identified several shortcomings leading to the design and implementation of a more flexible event processing solution (see the full discussion on the issues presented in Section 2.2).

On the other hand, the design and implementation of a Distributed Complex Event Processing (DCEP) approach presents concrete benefits against a centralized one. A centralized CEP must process a potentially huge number of events and thus may become a bottleneck [10]. Coping with an overwhelming amount of data can lead to message flooding and information overloading that can lead to queuing effects and into the delay of monitoring data processing [11]. The integration of the monitoring streams and the complex event processing jobs result in the ability to keep and process low-level high-frequency monitoring data (e.g. CPU/RAM usage) inside the VM that produces it while relaying processed data to higher levels only when needed; e.g. average CPU usage from all the VMs on a certain Cloud, or maximum number of users connected to a VM. This leads to the faster detection of application execution context situations that may necessitate reconfiguration, and at the same time reduce network bandwidth for monitoring data and the minimisation of message flooding occurrences. Last, but not least, the existence of only one event processing engine represents a single point of failure [12]. With the DCEP the impact of failures is reduced since the computational load is distributed among various CEP engines in an efficient way. Even though the top level CEP engine remains crucial for for the understanding of the *global* execution context, its failure will not bring down the monitoring system, and thus the DCEP is

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

recillient and offers graceful degradation in the presence of faults, even if it is the top level CEP that fails.

In terms of Melodic, we decided to address the limitations of the PaaSage monitoring system by introducing a standard and flexible DCEP, tailored to the needs of the multi-Clouds domain. The following criteria has been identified as important for the selection of the new Melodic monitoring system:

1. Ability to scale in order to cope with a potentially huge number of events generated by hundreds of sources;
2. Ability to apply complex formulas for the aggregation the metrics;
3. Ability to detect and publish complex events based on event algebra operators over the current global or local application execution context identified by the all or subsets of the metric values;
4. Design a monitoring system flexible enough to be able to efficiently handle metrics from a single component to huge deployments with hundreds of virtual machines.

Based on these criteria and the requirements for the Monitoring Plane, a monitoring mechanism was designed and implemented based on the Esper complex event processing engine. We have selected Esper[27] because of the following advantages:

- Esper offers a High Availability option in comparison to other CEP products and its basic version can even cope with 500.000 event/s [13];
- It is based on the Event Processing Language[28] for defining highly expressive complex event patterns that allow to perform nested queries over a monitor data stream. This expressivity is an important advantage of Esper against even more performance-oriented engines like Siddhi [13];
- It supports a rich set of configurable data windows while other engines provide a basic set of very simple rolling, sliding, or hopping windows. Esper data windows can be placed into intersection or union set-logic relationships [13];
- Esper supports all aspects of object-oriented design as well as dynamic typing, thus can handle schema evolution for adapting event processing rules[29].
- Esper is widely accepted in the event processing community and well-known for its commercial use[30] (e.g. Paypal, Accenture, Huawei, Oracle etc.).

The metric values and event messages must be transmitted among the dynamic number of Esper CEP engine component instances, which necessitates a flexible publish-subscribe message distribution mechanism as outlined in Section 2.2. The ZeroMQ used in PaaSage would qualify as

[27] www.espertech.com/esper/
[28] https://ieeexplore.ieee.org/document/1419978/
[29] http://www.espertech.com/esper/esper-faq/#comparison
[30] https://www.slideshare.net/ChandraDivi1/rule-engine-evaluation-for-complex-event-processing?from_action=save

an efficient peer-to-peer distribution mechanism. However, as outlined in Section 4.2.1, configuring a peer-to-peer protocol for large and dynamic systems is a complex task owing to the fact that both end-points need to be configured when a new data provider joins the system. Typically, if a new VM is started, it will provide some metric values, and all other CEP engines processing metrics of the type provided by the new machine would need to subscribe to these metrics. A message broker simplifies this configuration. Given that the ActiveMQ[31] is installed for the control flow in each VM, it could ideally be reused for the metrics monitoring and the event messages.

However, this requires that the message broker is capable to cope with hundreds to thousands of events per second, while reducing the chances of message flooding in any complex big data application multi-cloud topology. For our solution, we have used multiple instances (one per VM) of the ActiveMQ server ('the broker'). It is well known as a lightweight, Java Messaging System (JMS)[32]-compliant solution that offers high availability, high performance, and fault tolerance[33]. Specifically, each Active MQ broker can efficiently cope with up to 22.000 messages/sec per topic[34]. Furthermore, Active MQ offers additional features like traceability of the messages, metrics volume statistics and so on. The federated ActiveMQ brokers will therefore be the starting point for the monitoring and event system in Melodic, and the performance and scalability of the implemented mechanism will be investigated, and results reported in Deliverable D3.4.

As a default approach, this mechanism will be deployed in a three-layer architecture:

- VM Level – gathers, filters and aggregates messages at the Virtual Machine level
- Cloud Provider Level – gathers, filters and aggregates messages at the Cloud Provider level
- Melodic Platform Level – gathers, filters and aggregates messages at the platform level

Each level will filter and aggregate messages, if possible, thus effectively limiting the number of the messages passed to higher layers. The number of layers could be configured according to the requirements of the particular system at hand, e.g. additional layers could be added per availability zone or region. Further details on this monitoring mechanism will be provided as part of the Deliverable D3.4.

[31] http://activemq.apache.org
[32] https://docs.oracle.com/javaee/6/tutorial/doc/bncdq.html
[33] https://www.slideshare.net/ceposta/activemq-performance-tuning
[34] http://activemq.apache.org/performance.html

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

5 Integration and adaptation method for Melodic

This section contains detailed information about the Control Flow, including the suggested construction of the processes and flows, rules for services invocation and ESB exposition and data model handling and transformation.

5.1 Discussion on the selected integration method for Melodic

Due to the architecture and the characteristics of the Melodic project, especially the two different types of flows and planes, and after the carefully evaluation presented in Section 4, an integration solution based on ESB/BPM has been chosen. The chosen implementation of the ESB/BPM, MuleESB, includes ActiveMQ, a queue-based integration solution which will be re-used for the Monitoring Plane.

The orchestration of the data and the action flows within the system will be modelled as processes in an appropriate BPM language, i.e. the one supported by the BPM solution selected as described in Section 4. The Integration layer based on ESB/BPM will allow reliable and monitorable method invocation. It will also support reusability of the methods exposed by underlying components and avoid any point-to-point communication.

The advantage of using Enterprise Service Bus with MuleESB, which is an enterprise grade solution, is the ability to achieve a high level of scalability and availability. The MuleESB could be installed in a multi-node configuration, supporting the active-active mode.

For example, a typical pattern and best practice is to use a control process which will handle the events that must trigger any action or sub-process on the system. Then, based on the event type and current state of the system, one or more dedicated processes will be executed. Examples of dedicated processes include:

- Deployment process – a process responsible for orchestrating the deployment of a new application, from uploading the user's CAMEL model until the final application deployment in the cloud.
- Un-deployment process – a process responsible for un-deploying the user application from the cloud.
- Reconfiguration of the application based on a new solution generated by the solvers – this process will handle all events generated by the system's components to address properly the application reconfiguration.

The above list is not exhaustive, and new processes could be implemented according to the user requirements and preferences. The services provided by underlying components will be exposed on the ESB and could be used (and re-used) from any process. Based on this, most of the changes in scope could be handled simply by reconfiguring the process flow (or implementing a new flow)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

instead of performing changes in the system code. This integration-oriented architecture part introduces an abstraction layer between business flow and domain systems.

6 Summary

This deliverable addresses the core issue of integration and adaptation of the underlying PaaSage and Cactos frameworks to set up the Melodic platform. An appropriate integration and adaptation strategy is crucial for the success of the project, allowing end-to-end Cloud service automation. To this aim, the Melodic project has to achieve the seamless cooperation of the various needed components from the two adopted frameworks: one component from the Cactos framework (so-called “Cloudiator”) and 11 components from the PaaSage framework.

Architecting such integrated solutions is a complex task. There are many conflicting drivers and many possible "right" solutions and “cookbooks” for such framework integration. Therefore, the goal of the task of framework integration was to make the best decisions on crucial points (like type of communication for a given plane), according to a carefully collected set of requirements, paving the way for a long-term flexible, supportable, maintainable, and cost-effective Melodic platform architecture.

From the very beginning, different integration methods were already available from existing frameworks: ZeroMQ as a messaging mechanism for asynchronous communication between PaaSage components, and REST API invocations for integration with Cloudiator. The relevance of these integration methods for the Melodic project and the need for additional integration methods were discussed according to the two Melodic planes (Control Plane as well as Monitoring Plane) and to the Melodic specific integration requirements (including reliability, performance, and scalability). Four integration methods were reviewed (Point-to-point, Queue-based, ESB and ESB with BPM) according to each specific plane (Control or Monitoring Plane) and the specific prioritized requirements that have been posed. An overall comparison of the integration methods was achieved according to the degree of fulfilment of the requirements for integration and implementation effort in the Melodic project.

Based on the results of the evaluation of each integration method and professional recommendations of certified architects, the ESB with BPM orchestration method of integration has been chosen as the integration strategy for the Control and Data Plane and the Monitoring Plane.

Out of existing open source ESB and BPM solutions, MuleESB has been chosen for the ESB implementation, while the Camunda execution engine was chosen for the BPM part. In this way, a Melodic workflow will be efficient and adaptable to new requirements as they come, as such processes like deployment, un-deployment and reconfiguration processes, or even others, can be flexibly modelled. As a Distributed Complex Event Processing solution, the ESPER module has

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 731664

been chosen. For the Monitor Plane, the ActiveMQ, as a part of MuleESB, will be used, which efficiently fulfils the requirements of this plane. By means of this combined solution, we can build a uniform and robust integration layer for Melodic project, which most efficiently handles the carefully identified requirements.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

7 References

[1] J. Sutherland and W.-J. van den Heuvel, "Enterprise Application Integration and Complex Adaptive Systems," *Communications of the ACM,* vol. 45, 2002.

[2] T. Gulledge, "What is integration?," *Industrial Management & Data Systems,* vol. 106, 2006.

[3] M. M. Lynne, "Paradigm Shifts - E-Business and Business/Systems Integration," *Communications of the Association for Information Systems,* vol. 4, 2000.

[4] F. Losavio, D. Ortega and M. Perez, "Modeling EAI [Enterprise Application Integration]," in *12th International Conference of the Chilean Computer Science Society*, Atacama, 2002.

[5] O. Dahl, "Enterprise Application Integration - Applying Patterns to the Process of Message Transformation," *Reports from MSI, Växjö University,* no. 02142, 2002.

[6] M. Themistocleous and Z. Irani, "Benchmarking the benefits and barriers of application integration," *Benchmarking: An International Journal,* vol. 8, 2001.

[7] T. Reidemeister, "Fault Diagnosis in Enterprise Software Systems Using Discrete Monitoring Data," *Electrical and Computer Engineering,* 2012.

[8] G. Blair, N. Bencomo and R. B. France, "Models@ run.time," *Computer,* vol. 42, 2009.

[9] M. A. Munawar and P. A. Ward, "Adaptive Monitoring In Enterprise Software Systems," Department of Electrical and Computer Engineering University of Waterloo, Ontario, 2006.

[10] N. P. Schultz-Møller, M. Migliavacca and P. Pietzuch, "Distributed complex event processing with query rewriting," in *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems - DEBS '09*, Nashville, 2009.

[11] F. Paraiso, G. Hermosillo , R. Rouvoy and L. Seinturier, "A Middleware Platform to Federate Complex Event," in *IEEE 16th International Enterprise Distributed Object Computing Conference*, Beijing, 2012.

[12] A. Mdhaffar , R. B. Halima , M. Jmaiel and B. Freisleben, "A Dynamic Complex Event Processing Architecture for Cloud Monitoring and Analysis," in *IEEE 5th International Conference on Cloud Computing Technology and Science*, Bristol, 2013.

[13] M. Dayarathna and S. Perera, "Recent Advancements in Event Processing," *ACM Computing Surveys (CSUR),* vol. 51, 2018.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664