MORPHEMIC

Selection, design and implementation of the integration layer

Modelling and Orchestrating Heterogeneous Resources and Polymorphic Applications for Holistic Execution and Adaptation of Models in the Cloud

H2020-ICT-2018-2020
Leadership in Enabling and Industrial Technologies: Information and Communication Technologies

Grant Agreement Number
871643

Duration
1 January 2020 –
31 December 2022

www.morphemic.cloud

Deliverable reference
D4.3

Date
30 June 2021

Responsible partner
7bulls

Editor(s)
Paweł Skrzypek

Reviewers
Alexandros Raikos, Nebil Ben Mabrouk

Distribution
Public

Availability
morphemic.cloud

Executive summary

One main factor for the successful design and implementation of MORPHEMIC is to provide a proper integration and adaptation strategy that integrates the platforms on which MORPHEMIC is built, such as MELODIC and Activeeon's ProActive Scheduler. This includes not only the integration of components within the above frameworks, but also the development of new components and mechanisms in MORPHEMIC to handle the polymorphic and proactive adaptation feature. 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, Message Oriented Middleware (MOM) 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.

Author(s)
Katarzyna Materka, Michał Semczuk, Paweł Skrzypek

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871643

Table of Contents

List of Tables

List of Figures

1 Introduction

The right choice of integration strategy is crucial for the successful implementation of a given project, as there are plenty of integration methods available, each having certain advantages and disadvantages. Applying an appropriate methodology in order to select the best integration method for the given requirements of a system is quite a complex task. As stated in [1], a properly chosen integration strategy could provide significant benefits for the usability of the platform, in terms of stability, reliability, performance, as well as reduced development and maintenance costs.
The purpose of this deliverable is to evaluate different strategies for integration and adaptation (modify components of underpinning frameworks)**,** and to select the most efficient according to the objectives of the MORPHEMIC project. The selected strategy will also be analysed in order to highlight its main benefits and advantages.
As baseline integration - for the further enhancements in MORPHEMIC - project is focused around the integration of the underlying MELODIC[1] (as stated in Description of Action) and Activeeon's ProActive Scheduler[2] frameworks (based on the Consortium decision described and justified in deliverable D4.1[17]), a proper integration and adaptation strategy is crucial for the success of the project.

- MELODIC is an open-source integrated platform to support both the design, optimization, and deployment of cross-cloud applications. Together with an accompanying methodology, MELODIC supports model-based configuration, optimization and adaptive deployment of these applications. MELODIC allows for deploying existing and new applications independently of the existing underlying Cloud infrastructures.
- The Activeeon's ProActive Scheduler is a Cloud service orchestration framework which allows for deployment infrastructure and applications to the selected Cloud providers.

For the purpose of this document and 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 MORPHEMIC 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 a high-level plan for modifying components of underlying projects, in order to be usable in the MORPHEMIC platform. In this deliverable, the general adaptation strategy is presented, while details of the adaptation of integrated components (presented as a list of changes to the underlying frameworks) are described in the D4.1 "Architecture of pre-processor and proactive adaptation" deliverable.

1.1 Structure of the 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 the MELODIC platform and Activeeon's ProActive Scheduler, are analysed. The second part of the document is dedicated to analysing the methodology for the selection of the right integration and adaptation strategies for MORPHEMIC. The third part of the document explains how the aforementioned methodology has been applied, and what are the results of the application of the methodology. It also explains the rationale for selecting the respective integration and adaptation strategies for MORPHEMIC. Finally, the last document part elaborates more on MORPHEMIC's selected integration and (framework component) adaptation strategies.
The detailed structure of the document is as follows:

- Introduction – this chapter describes the main objectives and structure of this document.
- Integration methods in the MELODIC platform and Activeeon's ProActive Scheduler (Chapter 2) – the section contains a description of current integration methods used within the MELODIC and Activeeon's ProActive Scheduler projects.
- Selecting an integration and component adaptation strategy (Chapter 3) – description of the devised methodology for deciding on the integration and adaptation strategies for MORPHEMIC.
- Methodology Application (Chapter 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 MORPHEMIC.
- Integration and adaptation method for MORPHEMIC (Chapter 5) – description of integration and adaptation strategies for the MORPHEMIC project, selected based on the methodology application results, for both the Control and the Monitor Planes, as presented in deliverable D4.1 "Architecture of pre-processor and proactive adaptation".
- Summary (Chapter 6) – conclusions and next related steps.

[1] https://melodic.cloud/
[2] https://www.activeeon.com/products/workflows-scheduling/

The intended audience of this deliverable are primary the technical persons in the MORPHEMIC project responsible for development and integration of the components. Also, the persons outside of the Consortium interested in technical details of integration within advanced multi-cloud optimization platform are considered as audience for the deliverable.

1.2 Intended Audience

The intended audience of this deliverable are primary the technical persons in the MORPHEMIC project responsible for the development and integration of the project's components. Also, persons outside of the Consortium interested in technical details of integration within advanced multi-cloud optimization platform are considered as audience for the deliverable.

1.3 Glossary

Table 1 Specific terms used in the deliverable

Terms used in deliverable	Explanation of the term
High Availability (HA)	High level of availability of an IT system or application. This usually means that the system is installed in more than one instance.
Active – passive mode	Mode of High Availability (HA) / multi-instance configuration where one component's instance is active and handles requests while a second instance is up and will start handling request in case of failure of the first instance
Active – active mode	Mode of HA / multi-instance configuration where all component's instances are up and running as well as handling requests
Business Process Management (BPM)	A standard process for the management of business processes that is enabled through the use of Workflow / Process Engines
Strategy	A general plan to achieve one or more long-term or overall goals under conditions of uncertainty.
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 frameworks to the Morphemic platform
Integration	Alternative: process of linking together different components or systems in order to act as a coherent, coordinated whole
Adaptation	Adjustment and changes of a given component or technology needed to fit it to a particular IT system
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 an enterprise
Simple Object Access Protocol (SOAP)	A protocol for the integration of IT systems
Representational State Transfer (REST)	A protocol for the integration of IT systems
Control Plane	Integration layer responsible for handling control and data flow in the system
Monitoring Plane	Integration layer responsible for handling all monitoring related events and operations
MOM communication	Communication between IT systems based on a queue of messages, usually asynchronous
Synchronous communication	Direct method of communication between IT systems, where the invoker is blocked until it receives a corresponding response
Asynchronous communication	Indirect (usually through a queue message broker) method of communication between IT systems, where the invoker is not blocked until it receives the respective response

2 Integration methods in the MELODIC platform and Activeeon's ProActive Scheduler

In this section, the current (as–is) state of the integration layer in the MELODIC and Activeeon's ProActive Scheduler projects is described.

2.1 Integrating methods in MELODIC

MELODIC integrates several underlying frameworks into one platform [14]. Different frameworks use different integration methods, both in tools used and in communication types – synchronous versus asynchronous. The proper selection of the integration architecture with MELODIC was a crucial point for the success of this project. An additional element to consider was the level of effort needed to implement the chosen integration method. An Enterprise Service Bus (ESB) with Business Process Management (BPM) orchestration was chosen as the most flexible and easy method of integration. ESB is a common integration method used to integrate enterprise grade systems. BPM is a standard for the description and execution of business processes. The integration layer of MELODIC contains two planes:

- Control Plane – for business logic integration and controlling.
- Monitoring Plane – for monitoring related activities.

The key benefits of this approach are:

- Flexible logic implementation in the BPM flow with no hard coding.
- Support for both synchronous and asynchronous communication.
- Support for most of the integration protocols.
- Reliability, configuration easiness, and high availability.
- Ability to integrate with other enterprise applications due to the use of the ESB integration method.

The architecture of the MELODIC software platform is shown in Figure 1.

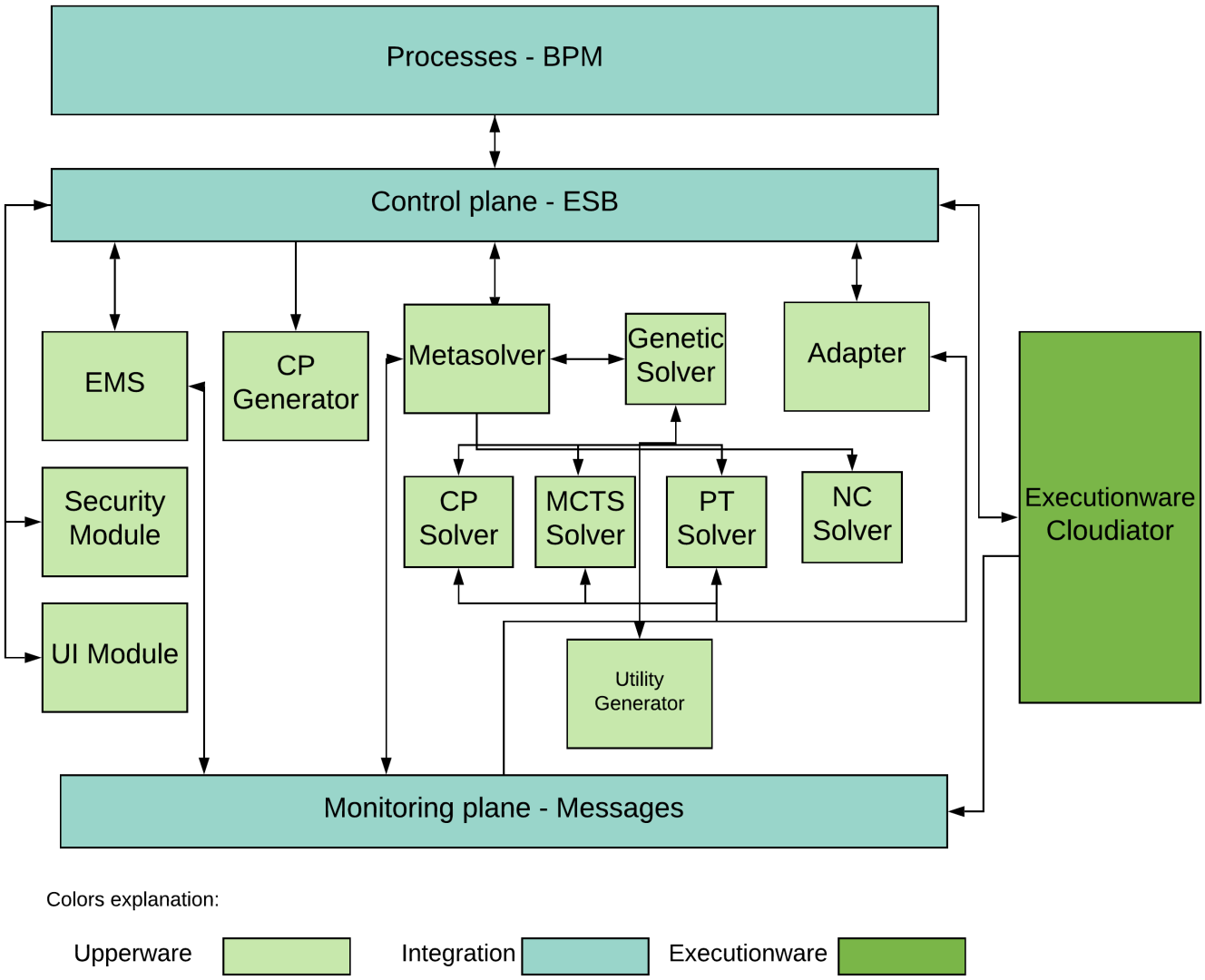

Figure 1 MELODIC high-level architecture

2.2 Integrating Activeeon's ProActive Scheduler

The Proactive Scheduler is intended to replace the Executionware module in MELODIC. Executionware is responsible for provision cloud infrastructure and deploying applications into the cloud. The rationale for that decision alongside with the Cloudiator's shortcomings are described in deliverable D4.1 [17].
The components of ProActive Scheduler are integrated into a microservice architecture. They interact through REST API calls. ProActive agents are located on remote infrastructures. They interact with the Resource Manager using the proprietary ProActive Network Protocol (PNP) and the ProActive Message Routing Protocol (PAMR). Both protocols are used internally by ProActive Scheduler.
Figure 2 exposes an overview of the Community version of ProActive Scheduler. The user interfaces of the Scheduler and the Resource Manager are included, but the workflow design interface (ProActive Studio) is omitted for the sake of clarity. The enterprise version comes with additional components not used in the MORPHEMIC project, and therefore not covered in this deliverable.

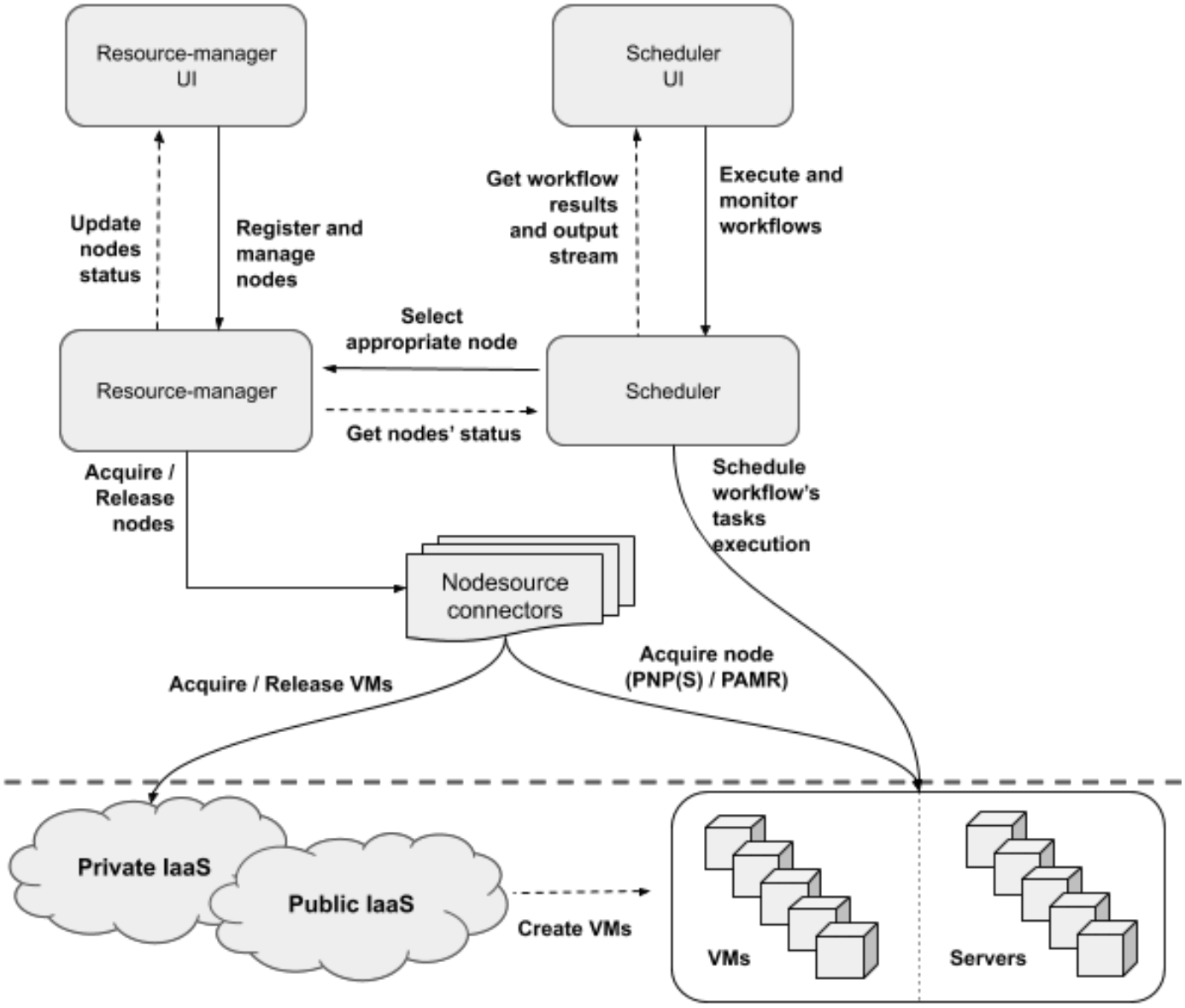

Figure 2 The architecture of the Activeeon's ProActive Scheduler

3 Selecting an integration and component adaptation strategy

This section contains a description of the methodology for choosing the integration and adaptation strategy for MORPHEMIC. 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 MORPHEMIC project, the following methodology has been used. This methodology has been devised according to our experience and the requirements (as presented in Section 4.1) 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 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 MORPHEMIC 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 MORPHEMIC 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 entire platform's operation.

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

       - *Partial_score_level_effort* = actual effort needed for method implementation divided by the maximum possible one (number 5).
       - *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 MORPHEMIC 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 MORPHEMIC is determined.
5. Based on the chosen integration strategy, the adaptation strategy will be determined, as elaborated later in this deliverable.

The final step is the selection of the right and most suitable tools to implement the selected integration method in the MORPHEMIC project.

The above steps are summarised in Figure 3.

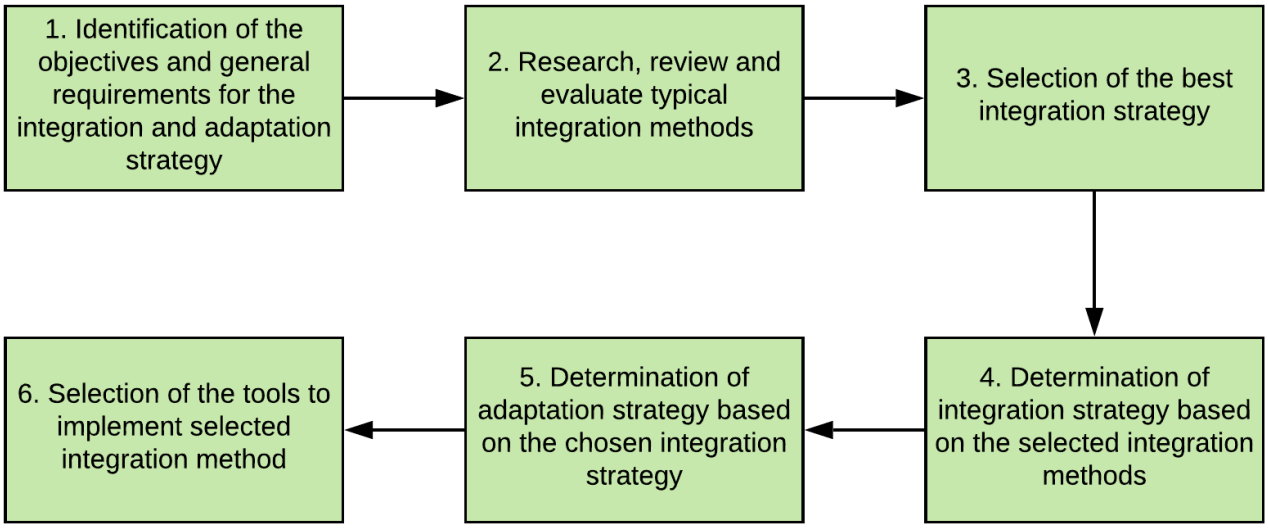

Figure 3 Diagram of methodology for choosing integration and adaptation strategy for the Morphemic project

4 Methodology Application

In the following sections, we elaborate on how the methodology analysed in the previous section is applied in the case of the MORPHEMIC project. The methodology used to evaluate and select the integration layer for the MORPHEMIC project is based on the approach described in [14][15]. The analysis is performed according to the structure of the methodology of the previous chapter in a step-wise manner, where each step is analysed in its own section.

4.1 Requirements Collection

One of the key activities of the work in the MORPHEMIC project is the integration and adaptation of the underlying frameworks MELODIC and Activeeon's ProActive Scheduler, followed by the introduction of the support for proactive and polymorphic adaptation. For this reason, the integration and adaptation strategy for MORPHEMIC should be carefully evaluated and precisely designed.
The fundamental objective of integration in MORPHEMIC is to achieve seamless cooperation between the components, independently 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 MELODIC platform consists of 16 components; these components are integrated using ESB (synchronous) and ActiveMQ (asynchronous).
- Activeeon's ProActive Scheduler, has also a certain component structure, but the features are exposed by one, unified API. The components of Activeeon's ProActive Scheduler are integrated via a REST API (synchronous).

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

- Asynchronous, MOM communication (e.g., ActiveMQ used for metrics delivery)

- Synchronous, via a REST API[3] (e.g., in case of the MELODIC Adapter's integration with Activeeon's ProActive Scheduler)

Furthermore, there are two separate layers of integration (both in MELODIC and MORPHEMIC platform), 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 integration and adaptation requirements for each plane are listed below. 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.
- *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 access control over offered operations driven by security/access policies.
- *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)[4].
- *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 for every single Morphemic 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 (with respect to the components of that plane).
- *Req14CMP – Easy to use:* the integration method should be relatively simple as it needs to be executed for every single Morphemic application.

In Table 2 The relation of integration requirements to the two planes, we provide a summary of the requirements collected along with their mapping to the respective planes of the MORPHEMIC platform.

[3] http://searchcloudstorage.techtarget.com/definition/RESTful-API
[4] https://www.techopedia.com/definition/4298/java-message-service-jms

Table 2 The relation of integration requirements to the two planes

Req. Id	Requirement	Which plane is affected by the requirement (Control Flow, Monitoring, Both)
Req1CP	Reliability	Control Flow
Req2CMP	Performance	Both
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	Both

4.2 Integration Method Research and Review

In this section, we analyse the application of the 2nd methodology step concerning the research, review and evaluation of integration methods. 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. They can either use [2] and it can be also done with [3] or by the most used [4]. For the purpose of this document, the following definition of integration will be used: *the interoperability between separate IT systems or components* [2]. 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.

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 a effort to be implemented, but it increases the complexity of the solution and it might have some negative impacts 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 MORPHEMIC 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 needed 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 as well as 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 4, we present a typical, generic example of point-to-point integration of IT systems.

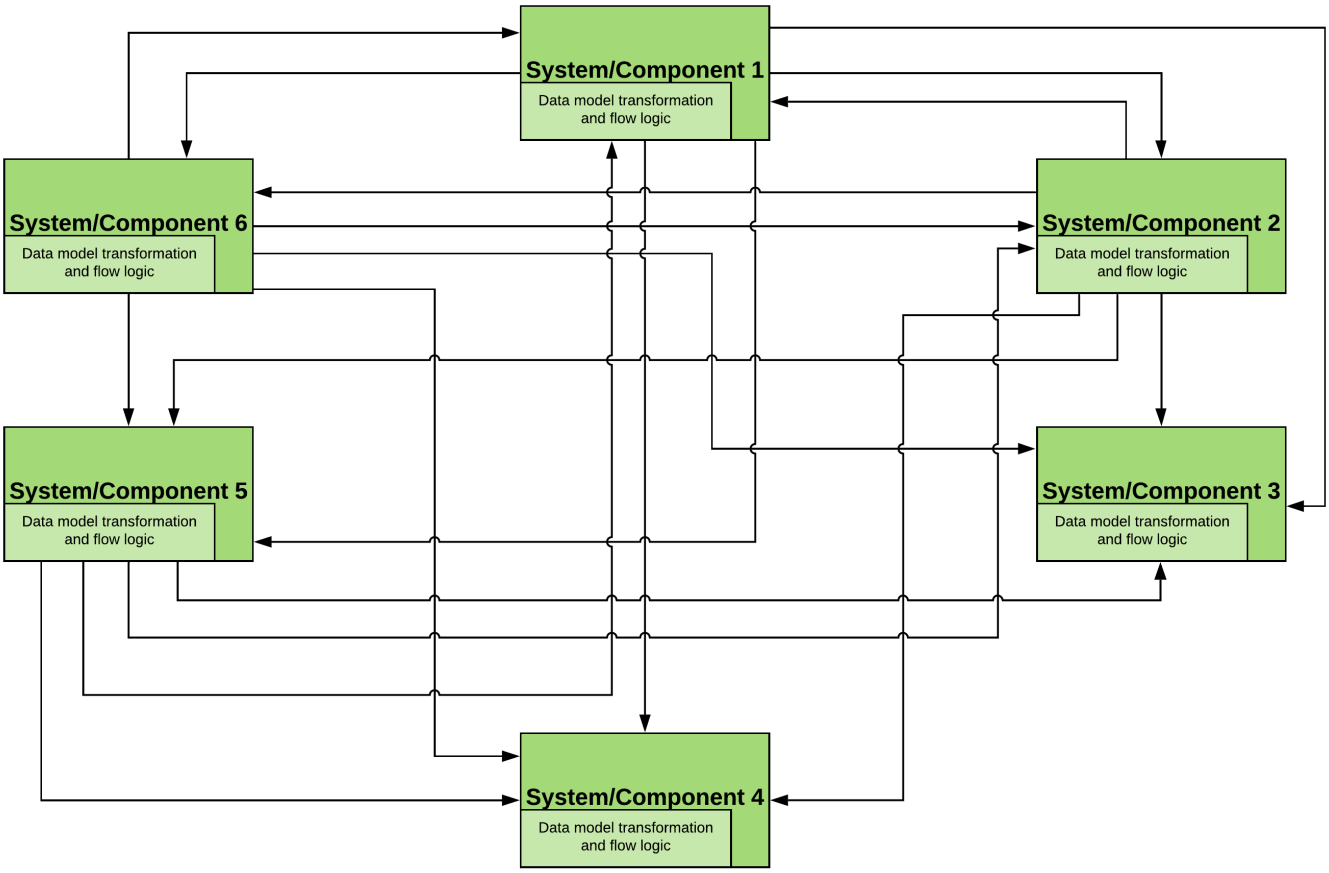

Figure 4 Point-to-point architecture

In Table 3, the evaluation of the point-to-point integration method is presented.

Table 3 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 is the reason for partial fulfilment. So, it is only limited by the performance of the respective sub-systems involved.
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; a huge effort will be required to implement it sufficiently and properly.
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.
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 MORPHEMIC 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 MELODIC and Activeeon's ProActive Scheduler frameworks based on related expertise.

4.2.2 Message Oriented Middleware integration

Message Oriented Middleware (MOM) use messages transported in a queue as a 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 partially reliable (with MOM broker as single point of failure), but the only supported method of communication is asynchronous communication. A more detailed analysis of MOM, also known as message-oriented middleware, is provided in [4].

In Figure 5, we present a typical MOM integration of IT systems.

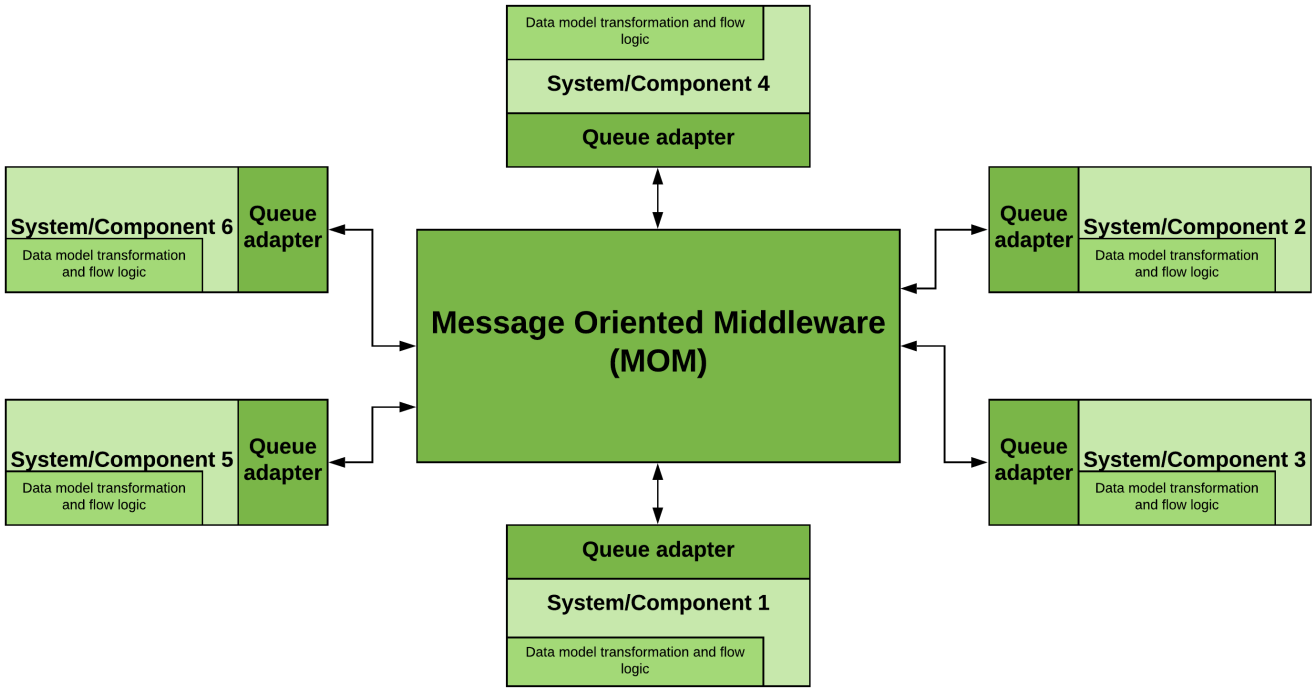

Figure 5 Message Oriented Middleware architecture

The estimated effort needed to implement this method for the MORPHEMIC project is 4, as it is already partially implemented for MELODIC.

Table 4 shows the level of fulfilment of the most desired requirements.

Table 4 Fulfilment of integration requirements by the MOM based integration method

Req. Id	Requirement	Fulfilment by given integration method	Comments
Req1	Reliability	Partially fulfilled	MOM, due to have one central point, is not fully reliable.
Req2	Performance	Fulfilled	Performance is high due to asynchronous communication and efficient method of communication.
Req3	Scalability	Partially fulfilled	Due to having one central point the scalability is limited.
Req4	High availability	Partially Fulfilled	Due to one central point the proper HA configuration set up is more difficult to be implemented.
Req5	Flexible orchestration	Not fulfilled	It is not possible to use external orchestration in this case due to asynchronous communication.
Req6	Support for both synchronous and asynchronous communication	Not fulfilled	This method of integration, by design, supports only asynchronous communication. There is no built-in 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 is usually limited only to messages flow monitoring. More advanced monitoring needs some custom implementation.
Req9	Logging	Partially fulfilled	Centralized logging needs some custom implementation.
Req10	Support for different integration	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

	protocols		covered.
Req11	Data model transformation	Not fulfilled	MOM 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	MOM has 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.

4.2.3 EAI/ESB based integration

The Enterprise Service Bus[5] 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 the format required by them [5]. A more detailed elaboration and research related to the integration approach using EAI/ESB could be found in[6] [6]. A typical ESB solution implements support for both synchronous and asynchronous communication. Asynchronous communication is usually implemented using a MOM (for example, MuleESB default broker uses ActiveMQ for asynchronous communication).

In Figure 6 we present a typical EAI/ESB integration of an IT system.

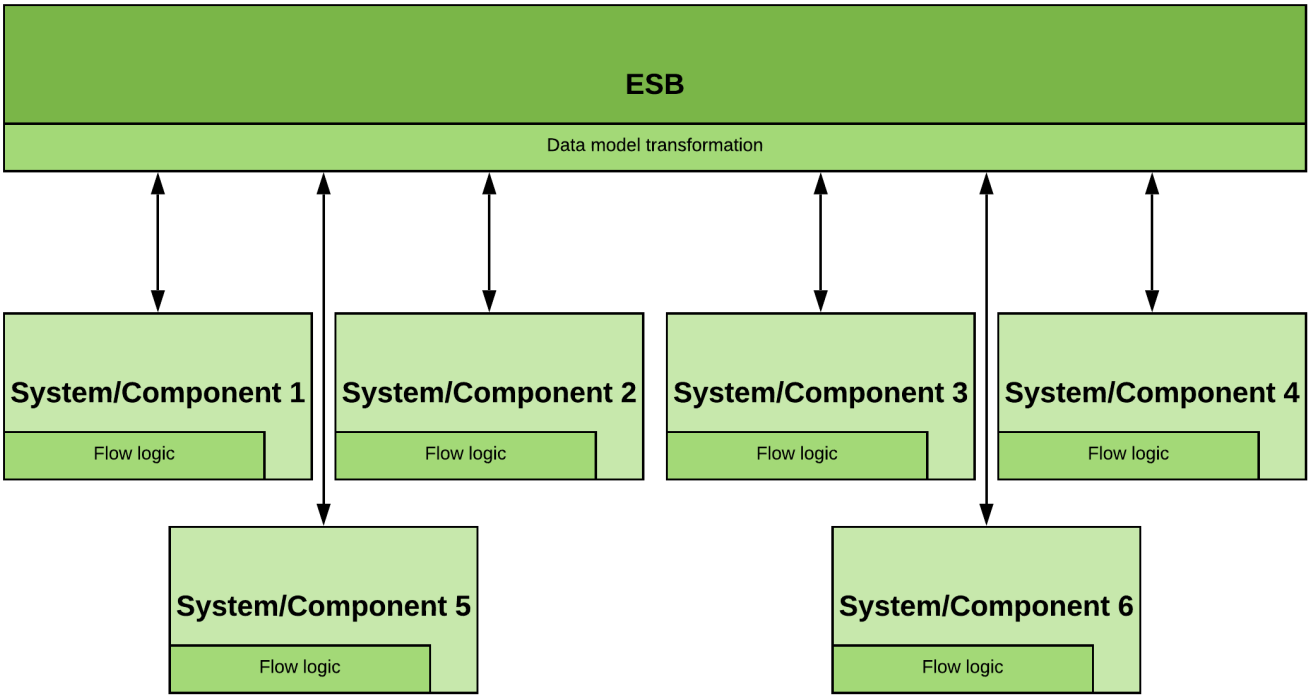

Figure 6 ESB based integration architecture

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

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

Table 5 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 a 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 covered 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 like lack of object type entities transformation [4].
Req12	Exception handling and support for retrying	Partially fulfilled	Most of the ESB implementations have support for exception handling and retrying. Nevertheless, it is not 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 MORPHEMIC project is 4, as it is partially implemented in the MELODIC platform.

4.2.4 EAI/ESB integration with BPM orchestration

EAI/ESB integration with Business Process Management[7] (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

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

EAI/ESB integration. The only difference is that business processes (BPs) are used for orchestrating method invocations, 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 invocations is provided in [1].

A typical EAI/ESB integration with BPM orchestration is presented in Figure 7.

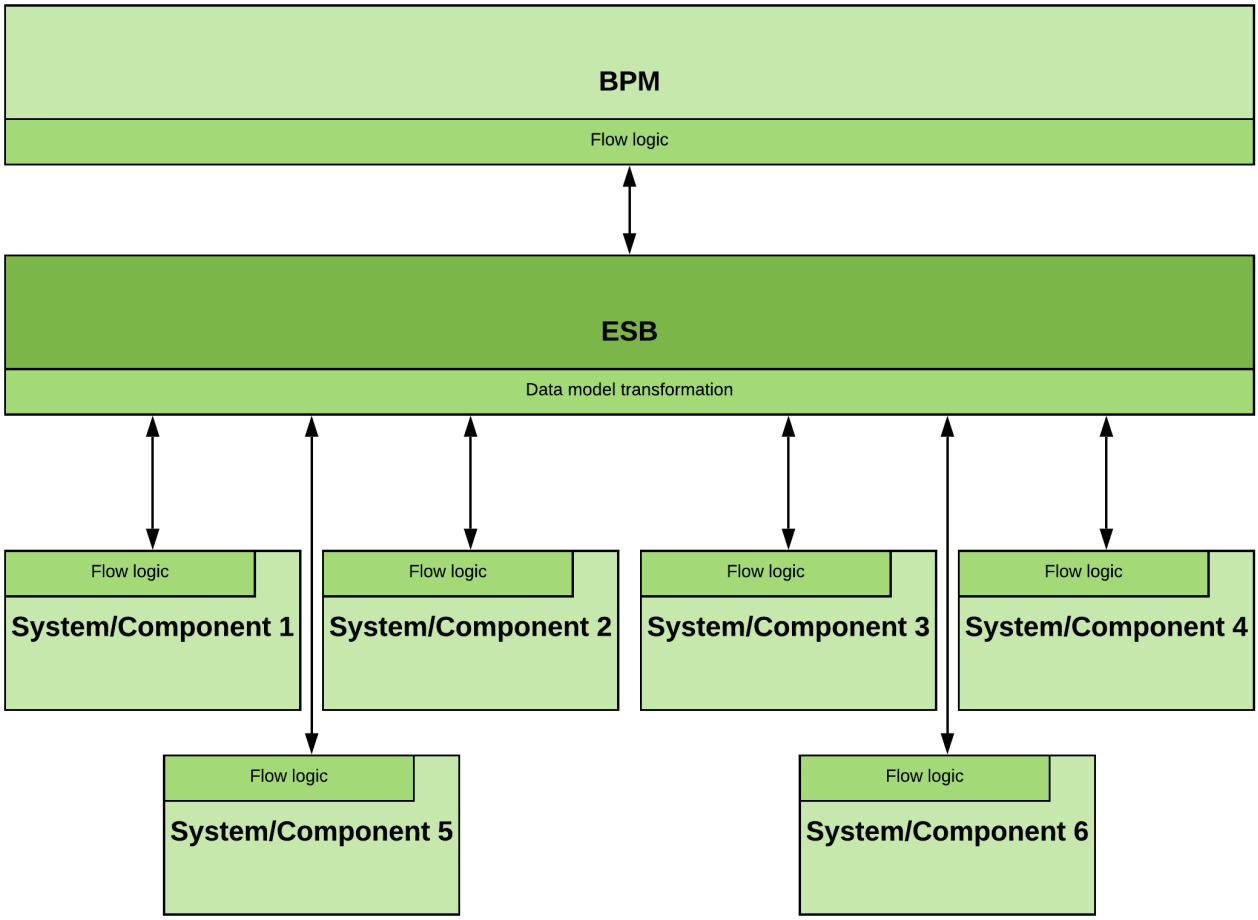

Figure 7 ESB based integration with BPM orchestration

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

Table 6 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.
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.

The estimated effort needed to implement this method for the MORPHEMIC project is 5, as this method is actually used by MELODIC project, as well as supported by Activeeon's ProActive Scheduler. The communication between MELODIC and Activeeon's ProActive Scheduler is done using REST protocol. End points can be defined in unified way in the ESB, including security, logging and monitoring of these end points.

4.2.5 Overall Evaluation Results

Table 7 summarises the evaluation results for the integration methods examined across all integration requirements. 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 the method i over the 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.

Table 7 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
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 8, 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 8 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
Req13	Low resource usage	5	5	3	3
Req14	Easy to use	5	5	3	3
	Estimated effort	3 (/5)	4 (/5)	4 (/5)	5 (/5)
	SUM OF POINTS for Control Plane	17 (/65)	33 (/65)	54 (/65)	63 (/65)
	SUM OF POINTS for Monitoring Plane	13 (/15)	15 (/15)	13 (/15)	13 (/15)

In Table 9, we present the calculation of the overall scores per plane (covering the 2nd level). The respective results are imprinted in the table. Finally, the Overall Score for each method is calculated, based on the scores for the Control Plane and the Monitoring Plane, with weights 0.6 and 0.4, respectively. A slightly higher weight is assigned to the Control Plane, due to the greater importance of this plane for the whole solution.

Table 9 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	4/5=0.8	54/65=0.86	0.85	13/15=0.86	0.85	0.86
ESB+BPM	5/5=1	63/65=0.96	0.97	13/15=0.86	0.9	0.94

Based on the results in the above table, we nominate ESB+BPM as the best integration method for the combined Control 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 MORPHEMIC. 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 determination

Based on the results of the previous methodology step, the selected integration strategy 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 an 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 experts, while the second analyses the final decision taken.

4.3.1 Expert Recommendation

4.3.1.1 TOGAF Architect recommendation

Based on the requirements of the MORPHEMIC 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.

4.3.1.2 AWS Architect recommendation

The MORPHEMIC 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 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 MORPHEMIC project, along with the professional recommendations by 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 MORPHEMIC 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 MORPHEMIC will be the following:

- All the components will be integrated based on the decided integration strategy and method.
- The structure of the repositories will be aligned and described in the Confluence (as technical documentation of the project) of the MORPHEMIC project[8].
- The unit and integration tests for each of the components should be prepared as described in the D4.4 "Test Strategy" deliverable.

The initial architecture of MORPHEMIC as described in D4.1 "Architecture of pre-processor and proactive adaptation" 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 and BPM implementation

Based on the chosen integration strategy for MORPHEMIC, ESB integration with BPM, the following subsections focus on evaluating possible ESB and BPM implementations, which could be used in the MORPHEMIC project.
Due to the licensing model of the MORPHEMIC project, i.e., 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 open-source solutions have been evaluated:

- ServiceMix[9] – based on review and previous experience, it is a high performance and available integration solution, being the most mature and stable one [16].
- MuleESB[10] – based on review and previous experience, as well as actual exploitation in the MELODIC platform, it is the most innovative solution, especially in the Cloud computing area, with an easy-to-use Graphical User Interface (GUI) and possible, additionally paid support from MuleSoft [16].
- WSO2[11] 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 10, MuleESB has been chosen as the most suitable ESB implementation for the MORPHEMIC project as:

- It is actually used in MELODIC framework.
- It is a stable and reliable solution, supported by MuleSoft, with plenty of documentation and online courses.
- It supports the Cloud computing model.
- It supplies a rich and easy User Interface (UI) for configuration and management.
- It makes available pre-implemented integration patterns.

Table 10 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:

- Activiti[12] – one of the oldest and most mature open-source BPM implementations
- jBPM[13] – also, a mature and stable BPM implementation, developed by JBoss[14], with integration support for the business rule server Drools[15]

[8] https://confluence.7bulls.eu/display/MEL/Morphemic
[9] http://servicemix.apache.org/docs/5.x/user/what-is-smx4.html
[10] https://www.mulesoft.com/resources/esb/what-mule-esb
[11] http://wso2.com/
[12] https://www.activiti.org/about

- Camunda – a mature and more robust implementation of BPM, which does not require the whole JBoss stack to work.
- Flowable[16] – 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 Activiti development team has been split (the core of the development team migrated to the Flowable project), Camunda has been chosen as the BPM implementation for the MORPHEMIC project. The Flowable project is not fully mature for now, so it cannot accomplish the requirements of the MORPHEMIC Project. The jBPM from JBoss requires the whole stack of the JBoss technology, which complicates the implementation of the project and increases the resource footprint of the platform. Key advantages of choosing Camunda are as follows:

- It is actually used in the MELODIC platform.
- 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 11 highlights the superiority of Camunda based on the 4 aforementioned criteria.

Table 11 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

5 Integration and adaptation method for MORPHEMIC

This section contains detailed information about the Control Plane. Included are:

- The suggested construction of the processes and flows,
- Rules for services invocation and ESB exposition.

The Monitoring Plane, due to simpler integration design do not require presenting additional detailed information.

5.1 Discussion on the selected integration method for MORPHEMIC

Due to the architecture and the characteristics of the MORPHEMIC project, especially the two different types of flows and planes, and after the careful 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 MOM 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 BPL (Business Process Language - BPL)[17], 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 invocations. 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 MuleESB, which is an enterprise grade solution, is the ability to achieve a high level of scalability and availability. MuleESB could be installed in a multi-node configuration, supporting the active-active HA mode (see Glossary in Section 1.3).

[13] https://bpm.com/what-is-bpm
[14] https://www.techopedia.com/definition/3525/jboss-application-server-jboss-as
[15] https://www.drools.org/
[16] http://www.flowable.org/
[17] https://www.btm-forum.org/boks/wikis/uam/UAM/guidances/supportingmaterials/uam_bpl_simple_858E0AEB.html

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 the 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 the upload of 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 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) 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 MELODIC and Activeeon's ProActive Scheduler frameworks to set up the MORPHEMIC platform. The Activeeon's ProActive Scheduler will replace MELODIC Exectionware and will not be a part of the global integrated solution. As for previous Executionware module the communication between MELODIC Upperware and Activeeon's ProActive Scheduler will be done through the REST protocol. The status of execution of the operations in Activeeon's ProActive Scheduler will be returned to MELODIC Upperware and available in BPM process. An appropriate integration and adaptation strategy is crucial for the success of the project, allowing end-to-end Cloud service automation. To this end, the MORPHEMIC project has to achieve the seamless cooperation between the various needed components of the two adopted frameworks. Detailed list of the reused and adapted components is presented in deliverable D4.1 "Architecture of pre-processor and proactive adaptation".

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 MORPHEMIC platform architecture.

From the very beginning, different integration methods were already available from existing frameworks: ESB with BPM with synchronous and asynchronous communication between MELODIC components, and synchronous REST API invocations for integration with Activeeon's ProActive Scheduler. The relevance of these integration methods for the MORPHEMIC project and the need for additional integration methods were discussed according to the two MORPHEMIC planes (Control Plane as well as Monitoring Plane) and to the MORPHEMIC specific integration requirements (including reliability, performance, and scalability). Four integration methods were reviewed (Point-to-point, MOM, 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 MORPHEMIC 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 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, the MORPHEMIC workflows 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. 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 the MORPHEMIC project, which most efficiently handles the carefully identified requirements.

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.

[14] Horn, G., Skrzypek, P., Prusiński, M., Materka, K., Stefanidis, V., & Verginadis, Y. (2019, October). MELODIC: selection and integration of open source to build an autonomic cross-cloud deployment platform. In International Conference on Objects, Components, Models and Patterns (pp. 364-377). Springer, Cham.

[15] Kritikos, K., Skrzypek, P., & Różańska, M. (2019, December). Towards an Integration Methodology for Multi-Cloud Application Management Platforms. In Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion (pp. 21-28).

[16] Shezi, Themba, et al. "Analysis of Open Source Enterprise Service Buses toward Supporting Integration in Dynamic Service Oriented Environments." International Conference on e-Infrastructure and e-Services for Developing Countries. Springer, Berlin, Heidelberg, 2012.

[17] Marta Różańska, et al. “D4.1: Architecture of pre-processor and proactive reconfiguration”, 2020.