Melodic
Big data cloud

Title:
System Specification

Abstract:
This document presents an initial system specification of the Melodic Multi-Cloud middleware platform. The document covers two important areas necessary for the subsequent development of the Melodic platform: Requirements Analysis and Technology Evaluation. The requirement analysis is essential to establish a set of initial features needed to realise an efficient middleware platform for Cross-Cloud data-intensive computing. At the same time, a complete evaluation of the available open technologies ensures that the efforts in the Melodic project are directed towards providing technology and tools that are currently beyond state-of-the-art and state-of-the-practice, and not on re-engineering solutions already available from other European and international open source projects.

Based on the requirement analysis, considering both the perspective of the general data-intensive Cloud applications as well as the needs of the four Melodic use-cases, a set of functional and non-functional requirements are presented in this document. In addition, technical and business-level constraints imposed by private and public Cloud solutions are identified for the Melodic middleware platform. Moreover, a comprehensive technology evaluation comprising identification of reusable components from three identified EU projects (PaaSage, CACTOS, and PaaSword) and available big data technologies are presented in detail – together with the identified extensions needed for Melodic. The requirement analysis and the technology evaluation together resulted in a preliminary architecture of the Melodic middleware platform. This preliminary architecture, as presented in this document, will lay the foundation of the first Melodic release, the integration release, due by the end of the first year of the project.

Multi-cloud Execution-ware for Large-scale Optimised 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:
D2.1

Date:
14 June 2017

Responsible partner:
Simula Research Laboratory

Editor(s):
Feroz Zahid

Author(s):
Yiannis Verginadis, Wiktor Zolnieroiwcz, Paweł Skrzypek, Daniel Seybold, Kyriakos Kritikos, Somnath Mazumdar, Antonia Schwichtenberg, Feroz Zahid, Jörg Domaschka, Geir Horn, Ernst Gunnar Gran, Daniel Baur, Hynek Masata and Paweł Gora

Approved by:
Geir Horn

ISBN number:
N/A

Document URL:
http://www.melodic.cloud/deliverables/D2.1 System Specification.pdf

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

Document	
Period Covered	M1-6
Deliverable No.	D2.1
Deliverable Title	System Specification
Editor(s)	Feroz Zahid
Author(s)	Yiannis Verginadis, Wiktor Zolnieroiwcz, Paweł Skrzypek, Daniel Seybold, Kyriakos Kritikos, Somnath Mazumdar, Antonia Schwichtenberg, Feroz Zahid, Jörg Domaschka, Geir Horn, Ernst Gunnar Gran, Daniel Baur, Hynek Masata and Paweł Gora
Reviewer(s)	Kyriakos Kritikos, Keith Jeffery, Jörg Domaschka
Work Package No.	2
Work Package Title	Architecture and Data Management
Lead Beneficiary	Simula Research Laboratory
Distribution	PU
Version	1.0
Draft/Final	Final
Total No. of Pages	109 + One Appendix

Table of Contents

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

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

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

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

List of Figures

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

1 Introduction

Data-intensive computing, often simply referred to as *big data*, is one of the major current trends in information and communication technology. In areas as diverse as social media, business intelligence, information security, Internet-of-Things, and scientific research, a tremendous amount of, possibly unstructured, data is created or collected at a speed surpassing what we can handle using traditional techniques. Several Cloud providers offer services to support data-intensive Cloud applications, typically implemented by large-scale data processing frameworks, such as Apache Spark[1] and Apache Hadoop[2]. The lack of a data-aware Cloud federation, however, keeps current Cloud computing from realising the full potential of data-intensive applications in the Cloud. The vision of the Melodic project is to enable federated Cloud computing for data-intensive applications, and provide the user with an easy-to-use unified Cloud environment, hiding the complexity of a Multi-Cloud set-up. The Melodic platform enables big data and data-intensive applications by transparently taking advantage of distinct characteristics of available private and public clouds, dynamically optimising resource usage, considering data locality, conforming to the user's privacy needs and service requirements, and countering vendor lock-in. From the perspective of a user, the Melodic framework will appear as an infrastructure-agnostic middleware platform supporting the development, deployment, and execution of data-intensive applications in distributed and heterogeneous Multi-Cloud environments. In this way, the Melodic platform will enable data-intensive applications to run within defined security, cost, and performance boundaries seamlessly on geographically distributed and federated Cloud infrastructures.

1.1 Scope of the Document

This document provides the initial system specification of the Melodic middleware platform, including the requirement analysis and the technology evaluation. The document is intended for a general audience and sets up the foundation of the Melodic project. Still, general knowledge about Cloud methodologies and software technologies is beneficial to understand some of the more technical parts of the document.

[1] http://spark.apache.org/
[2] http://hadoop.apache.org/

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

1.2 Summary of Requirement Analysis

The requirement analysis for the Melodic middleware platform is done from two perspectives: First, we have identified generalised requirements concerning Multi-Cloud and Cross-Cloud deployments and execution of data-intensive applications. We identify that it is important for the Melodic platform to support transparent deployments over heterogeneous Cloud infrastructures, both private and public, as well as infrastructures that are geographically dispersed. In addition, as clouds are both unpredictable and dynamic, runtime adaptation is necessary to ensure that the user-defined requirements and constraints are kept satisfied throughout the application and data life-cycle. Moreover, as trust is an important issue hindering broader Cloud adaptation, Melodic needs to make sure that the privacy and confidentiality constraints defined by the users are honoured while calculating the best possible Cross-Cloud configurations for the applications. Second, a comprehensive requirement analysis is done for each of the Melodic use-case demonstrator applications, covering requirements related to the applications and data management in Multi-Cloud scenarios. On the basis of the use-case analyses, and in view of the current use-case architectures, desired architectural requirements from Melodic have been identified for each of the use-cases. Besides functional requirements, non-functional requirements have also been identified for the Melodic project, such as scalability, availability, extensibility, reusability, and fault tolerance, among others.

1.3 Summary of Technology Evaluation

Melodic will make use of technology, components, and research results from the three identified European projects PaaSage[3], CACTOS[4], and PaaSword[5]. We have carefully evaluated the components of these projects, and identified necessary extensions and modifications required for their re-use in the Melodic platform. For the PaaSage project, the basic requirements revolve around the data-awareness that is needed to make sure that PaaSage components can work for the data-intensive applications. This includes adding data modelling, complementing the current application modelling, and providing support for data-aware deployments, configurations, and adaptation of the applications during reasoning phases (PaaSage *upper ware*). Similarly, for the components from the CACTOS projects (and PaaSage *execution ware*), extension are required to support big data frameworks and resource management systems. The PaaSword project's

[3] https://www.paasage.eu/
[4] http://www.cactosfp7.eu/
[5] 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

sophisticated context-aware access control mechanism needs to be integrated and tailored to allow for a context-aware security model with fine-grained access control scheme supporting Multi-Cloud data handling and life-cycle management.

For the big data technologies, two resource management systems, Mesos[6] and YARN[7], are evaluated. In addition, the three big data processing frameworks Hadoop MapReduce, Apache Spark, and Apache Flink are examined for integration in Melodic. After careful evaluation, the Mesos resource management system is selected for integration with the Melodic middleware. The selection is primarily based on Mesos' feature richness, its scalability, and its accessible extensibility needed for the Melodic middleware. Hadoop MapReduce and Apache Spark both were preliminary selected as part of the initial Melodic work plan, thanks to their popularity and extensive support in the current big data ecosystem. This preliminary selection is confirmed after an integration assessment has been completed. Another big data processing framework, Apache Flink[8], particularly geared towards stream processing, is omitted for the first Melodic development phase due to Flink's current lack of stability and support, but will remain an integration candidate for the future. Flink will be re-evaluated as it becomes more mature, and after the support of the other two more popular big data processing frameworks has been completed in the Melodic platform.

1.4 Structure of the Document

The rest of this document is structured into two main parts. The first part, comprising Chapter 2 – Chapter 5, starts by identifying different stakeholders in a Multi-Cloud ecosystem, and presents how the Melodic middleware platform will be beneficial to the stakeholders with respect to meeting their requirements in the context of a dynamic Multi-Cloud and Cross-Cloud operating environment. In Chapter 3, we move on to analyse key functional requirements for efficient execution of Multi-Cloud applications from a general Cloud user application perspective. Later, Chapter 4 presents our four use-cases and outlines their requirements and needs from Melodic. While providing use-case requirements, we also summarise relevant deployment scenarios with Melodic as a middleware between user domains and Cloud providers. Early use-case planning and technology use modelling presented in this document is intended to facilitate efficient development and modelling of use-case applications in parallel with the development of the Melodic middleware platform. Finally, in Chapter 5, we provide the main non-functional requirements needed from a Cloud middleware platform, such as scalability, availability,

[6] http://mesos.apache.org/
[7] https://hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site/YARN.html
[8] https://flink.apache.org/

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

extensibility, and interoperability, among others, and summarise Melodic's goals and means for fulfilling those requirements.

The second part of the document, Chapter 6 – Chapter 9, provides a comprehensive technical evaluation of available open source technologies potentially reusable by the Melodic project. The Melodic platform is meant to be developed by integrating available open source technologies, while providing the required extensions for efficient Cross-Cloud data-intensive processing. In particular, results from the three EU projects PaaSage, CACTOS, and PaaSword were already identified in the work plan to be reused in the Melodic project. In Chapter 6, we provide an overview of the available components from these three projects that will be reused in Melodic, also sketching extensions and modifications needed to be applied on them for Melodic. A technology evaluation of available big data technologies is presented in Chapter 7 and Chapter 8. Specifically, resource management systems are presented in Chapter 7, while big data processing frameworks are discussed in Chapter 8. In Chapter 9, based on the requirements and evaluation of the available technologies, we present the initial Melodic architecture. We conclude this deliverable in Chapter 10.

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

2 Stakeholders for Melodic

The key Melodic stakeholders are coming from two, increasingly overlapping, communities; the Cloud community and the big data community. Within the Cloud community, there are four main groups of stakeholders. The *Cloud Application Users* and *Cloud Application Developers* in different ways consume and refine services provided over the network by the *Service Providers*, often in the form of the Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), or Software-as-a-Service (SaaS) service models. In addition, *Cloud Brokers* or *Aggregators*, act as middlemen between Cloud users and service providers to facilitate efficient procurement of (composite) Cloud services. The big data community spans stakeholders linked to the value chain developed around the growing demand for big data processing in enterprises over the last few years. In the context of Cloud computing, there are three important stakeholders in the big data value chain: *Data Consumers, Application Providers,* and *Service Orchestrators*. The data consumers consist of a diverse set of organisations that can benefit and create value from the analytics and insights provided by data-driven applications and services. The application providers, on the other hand, develop and deploy data analytics services on platforms administered by System Orchestrators, which on their side deal with data management issues such as unification of data from heterogeneous sources and provide data processing frameworks to the developers.

2.1 Cloud Stakeholders

Cloud Application Users

Today, Cloud based services are an integral part of our everyday lives. We use Dropbox, Google Drive and OneDrive to store files; Google Docs and Office 365 to write documents; and Gmail, Yahoo mail and Outlook.com to manage emails. Moreover, a large number of business applications, banking software, content management systems, retail web portals, and shopping websites, just to mention a few, rely on Cloud-based solutions; thus everyone that uses them is a *Cloud application user.* Cloud performance, application configuration in the Cloud, and data centre location, all directly affect response time and perceived application efficiency, and ultimately the resulting Cloud user experience. In addition, for predictable service delivery to the Cloud users, applications need to be scalable and exhibit suitable performance levels from the user's perspective.

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

Melodic, by improving performance, security, and scalability of the Cloud applications, and in particular the data intensive ones, will be directly beneficial to the Cloud application users. Melodic's multi- and Cross-Cloud application deployment features, will make it easier for the application developers to develop Cloud applications that can be tailored to the Cloud application user's preferences, be privacy-aware, and offer predictive services under dynamically changing context and user load. In addition, as Melodic promises to improve cost-effectiveness of the Cloud applications by leveraging the economically best available Cloud offers from a variety of Cloud providers, low-cost Cloud services will in turn be offered to the end users due to open competition. Finally, Melodic will also foster innovation among Cloud providers and application developers, as once the vendor lock-in is overcome, the competitive advantages of the big providers should diminish. In this way, only the most competitive services in the market will prevail, which will be most beneficial to the Cloud users.

Cloud Application Developers

To an increasing extent, modern enterprises rely on Cloud-based services to meet their computational or data storage demands, demands they are unable to satisfy as efficiently using their own private infrastructure due to high management and operational costs. Moreover, even in private infrastructures Cloud computing helps improving resource utilisation and fault-tolerance. Cloud applications differ from traditional applications in two important ways. First, to take full advantage of the Cloud computing paradigm's elasticity and the corresponding pay-as-you-go model, Cloud applications need to be inherently scalable. Second, the storage model in clouds may differ substantially from those of traditional applications, depending on the kind of database used.

For the Cloud application developers, Melodic will deliver several important benefits to help the developer create, deploy, maintain and monitor applications in an easier, more efficient and automated manner. Melodic will also counter vendor lock-in and encourage competition by taking advantage of federated Cloud computing, together with data-aware deployment, configuration, execution, and adaptation of applications on Multi-Cloud platforms. Melodic will provide a unified view of private and public Cloud services to the Cloud application developers, and will facilitate autonomic selection of Cloud resources at the granularity of application components and data objects. The Cloud application developers will no longer need to choose resources from a single or static set of Cloud providers and live with whatever pricing and Service Level Agreements (SLAs) these providers imposes, while still satisfying their performance and security constraints. At the same time, Melodic will ensure that data management, spanning data migration, storage, and replication in geographically distributed Cloud locations, is under

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

complete control of the Cloud application developers, albeit indirectly through defined data access control and data management policies.

Melodic will enable holistic management of the complete data life-cycle by delivering an autonomous model-driven framework for data-aware design and development of Multi-Cloud applications. The framework will help Cloud application developers to annotate data roles, requirements, and flows at design time for this purpose. Melodic will also investigate the effectiveness of automated data placement and inter-cloud migration strategies.

Melodic will deliver a framework for effective application monitoring, runtime adaptation, proactive reconfiguration, and autonomic elasticity of data-intensive applications on highly distributed and federated Cloud resources. Real-time performance metrics, combined with historical data and Cloud specific statistical models, will enable Melodic to optimise Multi-Cloud resource allocation for a particular workload. In addition, Melodic will be equipped with machine learning capabilities in order to improve its prediction accuracy over time. Furthermore, the adaptation engine will be driven by a proactive reconfiguration mechanism to warrant early detection of potential constraint violations, and seek resource reconfiguration or an alternative application deployment in order to satisfy the given constraints, taking any reconfiguration or redeployment costs into account. This reconfiguration mechanism will be coupled with an ECA (event-condition-action)-based approach which will focus on adapting locally the application in terms of a single Cloud on which it is partially or fully deployed. Such an approach will rely on adaptation rules that are supplied by the application provider in the CAMEL language.

Melodic will deliver an integrated framework for storage and processing of data in the clouds under defined security and privacy specifications. A secure and efficient metadata management system will be developed to aid in implementing fine-grained access control mechanisms. In addition, the use of contextual information will enhance the access control mechanisms and it will enable the use of appropriate flexible and intelligent policy enforcement tools, able to cope with both the characteristics of the data artefacts to be accessed but also with the contextual elements that an access request may carry (e.g., origin of the request, time, device type, requestor role, etc.). The data access framework will enhance trust on Multi-Cloud deployments by providing Cloud application developers, and hence the Cloud users, greater control on their data. Coupled with the functionality preserving cryptographic and non-cryptographic data protection schemes, the objective of data access framework will be to ensure that the Cloud user's sensitive data is properly protected before it is uploaded to the Cloud, while still being able to benefit from processing it in the Cloud. At the same time, Melodic platform will ensure that the low level data management details, such as caching, pre-fetching, and the partitioning for parallel distributed processing are mainly automated providing dynamic autonomous data management under the high-level control and access policies implemented by the user.

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

Cloud Providers

In the scope of the Melodic project, a Cloud provider is defined as an organisation offering Cloud services to the user. These Cloud services may include all kind of service models, like Infrastructure as a Service (IaaS), Platform as a Service (PaaS) or Software as a Service (SaaS).

According to the NIST definition of Cloud computing [1], one can differ between public clouds where the Cloud services are offered to the general public (e.g., Amazon AWS[9], Microsoft Azure[10]) and private clouds where the Cloud services are only available to users of a single organisation (e.g., privately hosted *OpenStack* [11]Cloud). While in both deployment models, the provider of the service is categorised as a Cloud provider, the features of Melodic will differ for those types, mainly due to the varied level of administrative control available at the particular domain. For public clouds, users have very limited control over workload deployments onto the actual hardware, and hence Melodic must rely on application specific monitoring to ensure optimised usage of the acquired resources. On the other hand, in private Cloud deployments, optimisation of the workload while taking advantage of the higher level of control over the private resources can be advantageous. Yet, in hybrid Cloud setups, when jobs are simultaneously deployed on segregated resources from both private and public clouds, appropriate mapping of workload on private Cloud resources may still be very useful for achieving overall workload optimisation for the deployed applications.

Public Cloud providers are mainly an indirect stakeholder for Melodic. As Melodic addresses Multi-Cloud interoperability, cost control, performance predictability and Cloud security, it helps removing common obstacles hindering Cloud adoption [2]. Therefore, Melodic will decrease the effort required by companies to move their services to the Cloud, increasing the potential user base of public Cloud providers. In addition, reducing the vendor lock-in by providing a common access interface, will allow new providers to enter the market more easily. In this connection, several small local Cloud providers, in Poland for instance, have already shown interest in the Melodic project. While the above mentioned advantages also apply to private Cloud providers (and user organisations), they will also benefit from the improved resource management that Melodic will offer for the private clouds. The integrated and extended data centre workload optimisation solutions from the CACTOS project will enable Melodic to addresses challenges related to increased complexity, heterogeneity, and energy efficiency in the private clouds, together with

[9] https://aws.amazon.com/
[10] https://azure.microsoft.com
[11] https://www.openstack.org/

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

taking a more sophisticated *white-box approach* of workload optimisation for the data-intensive applications.

Cloud Broker

Since the number of Cloud service providers significantly grows and the requirements of Cloud consumers become more sophisticated, the need for entities willing to assume the role of an intermediator is becoming stronger. It is now clear that there is a distinct market for organisations that are ready to undertake such responsibilities. Well-known companies such as Rightscale[12], Heroku[13], SnapLogic[14] and Cloudability[15] already offer valuable brokerage capabilities in the Cloud market. They are known as *Cloud Brokers*. Although, technology market analysts Gartner Inc., Forrester Research and the US standards body NIST disagree on the exact definition of the term "Cloud broker", they have all agreed on its significance for the Cloud-computing domain. According to Gartner, a Cloud broker (or a Cloud service broker) is any entity, or technology acting as an intermediary, in order to bring added value to a customer's use of a service[16], as long as such an entity has a direct contractual relationship with the consumers of Cloud services[17]. A broker performs brokerage, which is defined as the intermediation between consumers and providers. In [3], the authors have classified the kinds of brokerage services that future Cloud intermediaries may perform, noting that a given entity may choose to fulfil one, or more of these functions. These services include [3]:

- *Service Discovery* – a single point of access to multiple services offered by different providers
- *Service Integration* – the vertical connection of Cloud services from different providers across the layers of the Cloud stack
- *Service Aggregation* – the simple bundling, or more sophisticated composition, of Cloud services to provide attractive consumer-facing packages, with a single point of access, identity management and billing
- *Service Customisation* – the extension or adaptation of generic Cloud services to provide added value for bespoke customers, with mechanisms to allow and regulate the participation of independent software vendors (ISVs)

[12] http://rightscale.com/
[13] http://heroku.com/
[14] http://snaplogic.com/
[15] http://cloudability.com/
[16] https://www.gartner.com/doc/1857618/cloud-services-brokerage-dominated-primary
[17] https://www.gartner.com/doc/1448121/defining-cloud-services-brokerage-taking

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

- *Service Optimisation* – the monitoring of service cost and performance, to offer arbitrage to consumers, who may select from alternatives according to pre-declared preferences
- *Service Quality Assurance* – encompassing service lifecycle governance, service certification, service monitoring for failure prevention and recovery.

Cloud Brokers constitute important stakeholders for Melodic, since based on the above discussion, it becomes evident that several of the Melodic envisioned functionalities pertain to, or extend some of, these brokerage capabilities. From a technological point of view, Melodic can be considered as a broker enabler. It will provide an integrated middleware platform capable of transparent management of data and constraint-aware deployment, execution, monitoring, and adaptation of the data-intensive applications on resources segregated from multiple Cloud service providers. Thus, any Melodic adopter will be able to offer certain brokerage capabilities such as optimisation and quality assurance over a number of private or public clouds. Nevertheless, we note that these brokerage capabilities do not follow the typical broker paradigm that supports multi-tenancy with respect to the brokerage services consumers, since the adopter of Melodic will be able to support single or multiple applications coming from a single organisation.

2.2 Big Data Stake Holders

Big Data Service Orchestrators

Answering the growing demand of big data analytics, big companies like Amazon and Google offer a complete stack of products that can be selected and configured to construct a certain big data solution based on the respective customer requirements. Other companies, especially SMEs, provide either one or just different versions of a certain product which understandably is not able to provide all the functionality covered by the aforementioned stacks. In general, such big data orchestrators, in particular SMEs, would benefit from the Melodic middleware platform by either enhancing their existing services, or developing and offering novel innovative solutions using the Melodic middleware platform. Moreover, as the Melodic project integrates state-of-the-art big data solutions, SMEs will be able to save substantial efforts in providing support of the existing open source technologies, while concentrating on the value-added services to strengthen their market positioning.

Apart from providing integration to the existing open big data technologies and supporting them in the context of the Multi-Clouds, Melodic also aims to deliver a unique set of innovate features which are not currently offered by others big data orchestration products in the market. To further elaborate, Melodic will provide a sophisticated automatic cluster management solution providing

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

adaptation of the big data application across multiple administrative domains via a model-driven approach according to the user requirements. In addition, Melodic will have the capability to define scalability / adaptation rules which could span both the data and computation aspects – that, in conjunction with the global adaptation approach, would enable the user application to adapt in a holistic manner. Furthermore, Melodic targets transparent and abstract big data process framework layer enabling easy switch from one processing framework to another. Finally, user-intuitive and transparent way of mixing different kinds of workloads within the same application irrespectively of the implementation technology for the processing, and the capability to satisfy hybrid deployment scenarios where private infrastructures are enriched with public resources leashed from public clouds when the respective need arises, will enhance and strengthen a range of big data services. These features would allow small and medium sized big data orchestrators to attract more customers and possibly increase their share in the market, currently dominated by the big players.

Big Data Application Providers

The big data application providers provide a vast array of big data applications and solutions offering distinct set of functionalities to the users. Broadly speaking, offered solutions lie in two categories: solutions providing *general-purpose* big data technologies, and the solutions targeting application of the big data technologies in particular domain areas [4]. The technological solutions deal with addressing key technical issues such as large-scale data acquisition, efficient data storage, resource management, scalable data processing, real-time data analytics, and data visualisation. In this connection, Melodic provides an *enabler middleware* for transparently integrating the use of big data technologies on Multi-Cloud and Cross-Cloud installations. Using Melodic, big data technology providers will be able to take the big data ecosystem across geographically distributed data centres spanning over different administrative domains and various Cloud providers. To begin with, Melodic targets to support integration with today's most popular big data processing frameworks, Apache Hadoop MapReduce and Apache Spark, for enabling both batch and stream processing in Multi-Clouds.

On the other hand, domain-specific applications, such as those used in the health, finance, Internet-of-Things (IoT) or energy sector, provide solutions offering application of the big data technologies in improving decision making, risk analysis, and to attain competitive intelligence in different verticals. Melodic, being a general purpose Multi-Cloud middleware platform for data-intensive applications, do not specifically target any particular application area. However, through the use of four distinct use-cases, Melodic aims to demonstrate sufficient features addressing the needs of different application areas, such as real-time processing, strict data confidentiality, cost-effectiveness, performance optimisation, and on-demand processing with automated elasticity.

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

Data Consumers

Today is the age of big data. In areas as diverse as manufacturing, financial services, digital media, business intelligence, information security, scientific research, and IoT, organisations create or collect very large datasets, often at speeds surpassing what we can handle using traditional data management techniques. Within the huge amount of data produced lies a great potential in the form of undiscovered structures and relations. To realise this potential, gain new knowledge, and create business value, the data produced needs to be accessed, processed, analysed, and visualised – in the most efficient manner. This need of big data processing has led organisations to invest heavily in big data related applications and services over the last few years. Data consumers are such organisations that count on extracting business value from the vast amount of data. From the perspective of the consumption of data offered through big data processing applications and services, data consumers are an indirect stakeholders and beneficiary of the Melodic middleware platform. Melodic will enable application providers to take advantage of the Multi-Cloud and Cross-Cloud resources, while keeping with the privacy and performance requirements of the users, which in turn will be beneficial to the data consumers. At the same time, holistic data management features of the Melodic project, covering complete data life-cycle in heterogeneous infrastructures, will be directly beneficial for the data consumers that, for instance, consume data at different geographical locations and manage it by themselves.

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

3 Generalised Multi-Cloud Requirements

In this chapter, we discuss generalised requirements for efficient execution of data-intensive applications in Multi-Cloud environments. The requirements presented here are in line with the project work plan and, in general, stem from four fundamental principles. First, the Cloud users need to have high-level control over their applications, resources, and data managed by the Melodic platform, while the platform itself will optimise Cloud applications as per user-defined constraints and policies concerning both the data and applications. Second, given a set of application-specific requirements, constraints, and specifications, the Melodic should be able to function autonomously with minimum need of user interaction during the operation. Third, Melodic should support simultaneous segregation of resources from different administrative domains (also referred as *Cross-Cloud*); heterogeneous and geographically distributed, private and public, and following different service models, such as IaaS and PaaS. Finally, the Melodic platform should support the complete life cycle of data and applications. In this way, the Melodic platform will provide support for DevOps core tasks in distributed and heterogeneous Multi-Cloud environments.

3.1 Transparent Deployment and Execution

The users expect that the platform offers an automated and transparent way of managing their data-intensive Cloud applications. This means that the users just need to appropriately model their applications and then interact with the platform in a limited and restricted way. As described in the introduction of this chapter, the main principle is that the application model comprises all appropriate information that is required by the platform to find the best possible Cloud service combination that satisfies the user requirements. Moreover, limited user interaction with the platform in decision-making helps achieve rapid application deployments. On the other hand, in case that a user desires to have more control over the application deployments, the platform should allow the user to inspect the deployment solutions and then either resume the execution of the deployment workflow or discard the offered deployment solution and instruct the platform to find a new one, by adjusting the input requirements, for instance. Once the user decides to continue with the application deployment, the platform performs the deployment transparently with respect to the user, as it has in its possession both the deployment solution and the user credentials for acting on behalf of the user across those clouds in which respective services are involved and exploited in the current solution.

Even when all application components are successfully deployed in the Cloud, a user application may not behave as intended. The can be, for instance, due to some components not being able to

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

communicate correctly or a certain component has exhibited an error. In such cases, there could be two different ways to check that the user applications are properly deployed: (a) the user provides some test scripts which can be applied over the application to check its operation - once all scripts are successful, then the application would be now ready for execution; (b) the platform itself somehow validates the user application. Ideally, the user might prefer the second solution but it is not possible for a platform to detect high-level application errors. In this sense, the first solution might be more appropriate. Thus, once the user provides sanity check scripts, the platform executes them and quickly identifies any specific error that has occurred. The user could then interfere, if needed, by either attempting to manually correct the error or modify the application model and re-initiate the application deployment. In case any specific issue occurs at runtime, such an issue can be handled by the platform as indicated in Section 3.3. As such, the platform provides an appropriate automation level to the user crossing both the application design and runtime but it does include specific interaction points to allow user control in the overall application management process.

3.2 Data Management

Efficient data management is characterised by challenges spanning each of the distinct phases of the data life cycle including data acquisition, preparation, storage, analysis, integration, aggregation, and its final representation. For large-scale distributed data-intensive applications, optimisation of the complete data life cycle becomes both complex and multi-dimensional, while individually proposed solutions for different phases and different levels often yield contradictory management decisions. As an example, optimising data analysis in geographically-distributed environments is not efficiently possible unless coordinated with the previous data acquisition, storage, and preparation steps. Moreover, unlike computations, data placement decisions in clouds can result in a long-term effect on the application placements due to high data migration costs. This is particularly true for the configuration of streaming data applications where data is continuously being added for processing and analysis. For these reasons, efficient data management covering all data lifecycle phases is essential for an efficient execution of data-intensive applications in Multi-Clouds, as envisioned by the Melodic project. Furthermore, efficient data partitioning, distribution, and replication mechanisms are required to maintain availability, and data privacy.

Melodic aims to enable holistic management of the complete data life-cycle by delivering an autonomous model-driven framework for data-aware deployment modelling and execution of Multi-Cloud applications. To support such a holistic and autonomous data management system, first, a generic metadata structure is needed to support data management, access control, and

data-aware application design for highly distributed and loosely-coupled Multi-Cloud domains. This also includes provisioning support for data annotations, roles, requirements, and defining data flows at design time for the developers. Second, efficient data placement, migration, and post-migration methodologies and algorithms are required to build a high-performance autonomous data management platform. This also includes efficient partitioning, caching, prefetching, and replication mechanisms for the data across Multi-Clouds. Finally, extensive support for standard data sources is needed, together with the provision of easy extensibility to cover a large variety of structured, unstructured, and hybrid data sources, and usability in different heterogeneous environments.

3.3 Runtime Adaptation

Unlike traditional IT infrastructures, Cloud datacentres incorporate a highly dynamic environment owing to multitenancy, rapid elasticity and on-demand resource provisioning. The dynamic nature of the Cloud breeds an equally unpredictable execution environment, resulting in variable application performance. Melodic takes advantage of the Multi-Cloud computing, together with data-aware deployment, configuration, execution, and adaptation of applications on Multi-Cloud platforms. To maintain the efficiency of such a platform with respect to the dynamic changes in the system, Melodic needs to support run-time adaptation of the Multi-Cloud user applications. These changes include variable Cloud efficiency in real-time, changes in end-user data processing requirements, resource reconfigurations together with alternative application deployment in order to sustain the service level demanded by the application provider.

The Melodic framework will be based on the Monitor-Analyse-Plan-Execution (MAPE) adaptation loop. The framework will need to provide well-defined APIs for monitoring the deployed application, along with access to real-time performance metrics, historical data concerning both the application execution and data in the clouds. Based on the monitored metrics, analysis and planning of the required reconfigurations to the Cross-Cloud application will be triggered. The output of this phase will be expressed in the form of a deployment script which is then coordinated by the Adaption engine and executed by the Execution ware level of the Melodic solution in order to reconfigure the Multi-Cloud user application. Finally, the adaptation engine enacts the adaptation by modifying only the parts of the Cross-Cloud application necessary to account for the difference and the target model becomes the current model.

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

3.4 Privacy and Confidentiality

According to the 2016 report of the Cloud Security Alliance[18] on top threats in Cloud computing, the most severe security threats were identified. Among the top 12 threats are the data breaches, the weak access control management and the data loss. It is also true that sensitive information disclosure, data privacy and confidentiality are among the major concerns that enterprises face when migrating their applications to the Cloud[19]. *Privacy* as the ability of an entity to seclude itself in the Cloud domain can be translated into the ability of protecting sensitive data stored in the Cloud by controlling access and only permit access to specific people in specific circumstances. The concept of privacy may partially overlap with the one of *confidentiality,* both being aspects of the *security* concept, which may imply, besides from the need of an appropriate use of sensitive information, the protection of this information even in cases where data might be intercepted due to system's or network's misconfigurations or successful hacking attempts. Hence, the security concept refers to the ability of reverting any sensitive information disclosure either for data in-transit (e.g., relayed across different VMs) or in-situ (i.e. stored in any public Cloud infrastructure). The notion of confidentiality is usually materialised via the use of appropriate cryptographic schemes that can secure the sensitive data in a way that renders them useless to any malicious interceptor that usually cannot acquire the corresponding decryption keys.

Based on this, Melodic will offer the appropriate tools that guarantee the secure access and processing of all the involved sensitive data, handled by big-data intensive applications deployed and maintained in Multi-Clouds through the Melodic platform: Any user should be properly authenticated before accessing data that reside in Cloud infrastructures. In addition, the data owner should be able to indicate as a requirement the need for cryptographically protecting her data that may be transferred, processed, and stored over public or private clouds through a Melodic-enabled application.

Any access request of a user or a software component (including Melodic's components) should be restricted according to a number of access control policies that can be evaluated after considering contextual information from the involved entities (i.e., Requestor, Resource and their relevant environment).

[18] https://cloudsecurityalliance.org/download/the-treacherous-twelve-cloud-computing-top-threats-in-2016/
[19] https://www.rightscale.com/lp/state-of-the-cloud

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

3.5 Application Support

In order to efficiently support data-intensive applications in Multi-Cloud and Cross-Cloud scenarios, it is important that the application components are mapped correctly on the heterogeneous infrastructures according to the application requirements. Earlier efforts have targeted model-based approaches for the design, development, deployment, and self-adaptation of Multi-Cloud applications [5], [6]. In particular, several Cloud modelling frameworks [5], [7] are in active development to equip application developers with capabilities to define a rich set of design-time and runtime attributes like application requirements, Quality of Service (QoS) constraints, and security considerations for Multi-Cloud deployments. Even though these initiatives are both relevant and beneficial for modelling data-intensive applications on federated Cloud infrastructures, such applications additionally bring a unique set of challenges and design needs, hindering their effective migration on Multi-Cloud systems. For example, big data processing is often characterised by distinct communication and compute-intensive phases in a job life-cycle. In addition to the big data applications, it important to ensure Melodic usability in a variety of application scenarios by providing support for legacy applications, such as web applications or applications using non-standard data processing toolchains.

3.6 Private Cloud Resource Management

When looking at current Cloud management software, they are split into two domains: the *user domain* where they help the user to manage his or her Cloud resources, to deploy software on those managed resources, or to apply scalability rules; the *Cloud provider domain* (infrastructure) where they manage the mapping of virtual machines to physical nodes. This gap between user and Cloud provider domain can lead to improper use of resources, as the information from the user and application domain is not passed to the Cloud provider and is therefore not included in the physical resource selection problem. This holds especially true, if the application is I/O intensive, as I/O properties like network and or disk are not part of the virtual hardware flavour selection offered to the Cloud user.

Melodic aims to close this gap, by interconnecting the Cloud user domain with the Cloud provider domain for private clouds. By discovering the physical nodes, their meta-data, such as hardware configuration, and the network topology of the private Cloud, Melodic can include this knowledge into its reasoning process, allowing the concrete and more optimal mapping of applications to specific physical nodes in the private Cloud.

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

3.7 Application Scalability

A major non-functional requirement, which also correlates to one of the main reasons for moving to the Cloud, is the scalability. Scalability indicates that a Cloud-based application should be scalable in order to acquire more resources when needed in order to appropriately handle additional or unexpected workload. There are two main dimensions via which an application can scale in the context of this project and the kinds of applications it attempts to suit: (a) computational: Virtual Machines (VMs) should be scaled up or out in order to obtain more resources to support the application processing; (b) data: databases should be scaled in order to accommodate for additional data that might need to be stored by the user application. As such, both dimensions would need to be scaled, or scaled out with replicates to allow parallel distributed data processing.

To support this scaling, a user expects to see respective scaling or adaptation actions that should be supported by the platform. Apart from scaling up or down, scaling in and out should also be supported in order to enable the application to use only those resources that it really needs and thus reduce application cost. Other forms of adaptation could be the migration of computation and data on different clouds, in case, for instance, the offerings of one Cloud might not still be suitable for the current user application.

While acquiring more resources could be the first step for handled additional or unexpected workloads, the second steps should come with the suitable control over these resources. This means that a load balancing mechanism should be in place in order to balance the load over the application components. Such a mechanism could be realised either via a new application component or by exploiting PaaS services offered by a certain Cloud. In the first case, the platform should enable the user to model somehow both normal and scaling scenarios in order to enable it to possess the appropriate knowledge to incorporate a load balancer component in the application when it is running.

By considering the modelling side, the platform should provide a mechanism via which the required scaling can be performed. By considering the different alternatives, two complementary ways could be employed: (a) the user models scalability rules which indicate how a particular application component can scale in a specific Cloud; (b) the user models the complete application model and exploits smart utility functions that guide the system on which components from the application model should be instantiated and according to how many instances. The second alternative is more suitable for global or Cross-Cloud application re-configuration.

To summarise, scalability is a must for any kind of Cloud application and a platform should provide to the user suitable control mechanisms in order to manage the scalability of his/her application. Such mechanisms should be mainly on the modelling side. They should also include

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

suitable adaptation/scaling actions which focus on both computational and data aspects. Melodic does support both modelling types, exhibits both local and global adaptation mechanisms and it already includes particular scaling actions which will be further enhanced with additional adaptation capabilities. Thus, it can really cater for well supporting the scalability requirements of applications.

3.8 Application Availability

Availability is an important requirement that affects selection of Cloud infrastructure for the applications. Cloud outages have the potential to cause millions of Euros in loss to business organisations. Just to give an example, in a recent failure of Amazon Web Services in February 2017, Cyence Inc.[20] estimated that the outage cost companies in the S&P 500 index around 140 million Euros in just an hour. Load balancing, check-pointing, redundancy, and service migration are some of the features that are used to maintain high availability of the offered services by eliminating single point of failures in the system. In Cross-Cloud scenarios, when resources used for the application are orchestrated from different Cloud providers on geographically dispersed data centres, the failure of one site does not fully affect the availability of the applications, if the application and data components are intelligently allocated in the Cloud allowing redundancy. Melodic, by providing transparent Multi-Cloud and Cross-Cloud computing for the applications, enables applications to use resources at geographically dispersed locations, provided that this is modelled in the CAMEL application model, which in turn will increase application availability in case of Cloud failures.

[20] https://www.cyence.net/

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

3.9 Summary of Generalised Requirements

Table 1: Summary of generalised Multi-Cloud requirements

Area	Main Requirements
Transparent Deployment and Execution	• Automated and transparent deployment of data-intensive applications • Ability to allow platform control flow to wait for the user instructions (through a user interface), if required
Data Management	• A generic metadata structure complementing unified data management, access control, and data-aware application design • Efficient data placement, migration, and post-migration methodologies and algorithms for Multi-Clouds • Support for a large variety of structured, unstructured, and hybrid data sources in heterogeneous environments
Runtime Adaptation	• Capabilities to continuously monitor deployed applications • Efficient algorithms to analyse and predict application performance based on historical data and real-time performance metrics • Support for model@runtime for dynamic runtime adaptation of deployed applications
Privacy and Confidentiality	• Secure and context-aware data access control mechanism • Ability to allow for user-defined data security and confidentiality requirements
Application Support	• Application components correctly mapped on heterogeneous infrastructures according to the application requirements • Data-aware scheduling of big data applications
Private Cloud Resource Management	• Efficient use of resources on the private infrastructures • Intelligent mapping of application components on to the actual hardware
Application Scalability	• Support for computational and data scaling

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

Area	Main Requirements
	• Support for application-defined platform local scalability rules • Utility-based scalability
Application Availability	• Transparent exploitation of geo-graphically dispersed Cloud locations

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

4 Use Case Requirements

In Melodic we have carefully selected 4 use cases, each representing separate class of problems to be addressed by Melodic as well as demonstrating different models of commercial exploitation.

In the first use case, CAS will show how a large CRM software provider can integrate Melodic into an internal, dedicated application store and deployment platform as an innovative way to manage extensions and personalisations of the core system. Melodic will provide support for deployment of data-intensive extensions of the CAS Cloud-based *anything relationship management (xRM)* software.

The second use case, demonstrated by CET, consists of two applications. In the first application, it will be demonstrated how Melodic will assist in providing close to real time processing of geo-dispersed big data on vehicle mobility for advanced, on-demand modelling of traffic in cities. Melodic will not only make this innovation technically possible, but will also enable CET to provide the service to many cities for an acceptable price, both under permanent contract as well as on demand, e.g., when a large event is planned in the city, or for crisis management purposes. With its second application, CET will demonstrate how the sensitive data on people mobility, acquired from and stored in different locations, including private clouds of telecom operators, can be securely accessed for on-demand processing and the delivery of affordable data analysis services to SMEs from different sectors, such as e-commerce, retail or tourism, potentially unlocking innovative data-driven business models in these sectors.

Use case 3, managed by 7Bulls, has a specific focus on value added services for SMEs deployed on top of the Cloud infrastructure. The FCR Secure Document Management application which is a SaaS solution for secure document management will be able to use public Cloud resources, without falling in a vendor lock-in, in combination with available resources in the private Cloud.

In use case 4, also demonstrated by 7Bulls, the focus will be on demonstrating suitability of Melodic for processing highly sensitive data. 7bulls will cooperate with the bioinformatics research group at the University of Bialystok (PL) to verify and demonstrate how Melodic will enable the distributed processing of the genome data, that come in a large volumes and are usually extremely sensitive.

In the following, we provide a detailed requirement analysis for each of the use cases, covering requirements related to both the applications and data management in Multi-Clouds. We also provide an anticipated architecture of the applications with the Melodic as the middleware platform, in view of the current architecture and the presented requirement analysis.

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

4.1 Use Case 1 – Market place for Data-Intensive Applications

Cloud-based anything relationship management (xRM) software, SmartWe, is designed to support customers in their daily work by providing a tailored software tool with respect to the particular role of each user. Tailored business solutions support work flows according to the needs of the users and are much more efficient and usable than generic software systems. SmartWe supports different ways to tailor the basic xRM solution to user role specific tasks, e.g., by adding only needed *apps* or by using the provided software development kit (SDK) to develop own apps.

The idea of an app-based xRM Cloud software that can be adapted and extended to diverse branches and customer needs is highly promising from a marketing and business perspective. However, it is also challenging from a conceptual perspective and even more challenging from a developer perspective. As the basic ideas is that the customers should be able to adapt existing apps and forms by themselves, CAS developed the app designer making it possible to perform the changes directly on the UI using the declarative descriptions of the underlying SmartDesign Client technology. Customers can also create their own data types using the graphical DB designer and create new apps for the manipulations of these introduced data types and records. Fields on the form of such an app can even be calculated and manipulated with the script engine that as well is available on the UI of the system allowing context sensitive content assist. All these are SDK tools assisting the app creation and adaptation, which is relevant not only for the customers who want to perform the customisations on their own but also for the CAS partners being in charge of not only promoting the xRM software system, but also performing the customisations on behalf of the end users.

Melodic comes into play when not only small customisation and apps for special datatypes are created but when a new app is to be developed that is either data- or computational intensive and therefore would affect the live system's performance in a drastic way. For example, a CAS partner may want to develop a new app for either integrating and analysing existing data from a 3rd party system or for analytics based on existing data from the primary CRM system. In these cases, there is a need for a dedicated deployment infrastructure for the app to be developed as:

- The data that is to be analysed needs to be stored in a dedicated way (because the original MySQL data structure might not be designed for the special analysis cases).
- The computations on the data might be that complex and heavy that they would affect the original system in a way that is not desired.

From CAS perspective, the recommendation of a perfectly fitting deployment environment for the app at hand is to be done by an app store/marketplace. Basically, we see the app store as being composed of different parts for the following stakeholders:

- 3rd party app developer

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

- CAS as platform provider
- End users

Looking at a common app publishing workflow from a high-level perspective, the following steps are to be performed:

1. App developer uploads his app and app description
2. CAS reviews the app, deploys it on a test environment and tests it
3. App developer gets recommendation of deployment environment, deploys the app, and the app is published (existence of account and credentials at the Cloud provider is a pre-requisite)
4. End user buys the app and uses it
5. The profit margin is divided to the 3rd party developer and CAS
6. App developer and CAS gets monitoring and usage information
7. The app deployment infrastructure scales according to the computational resources needed and to the actual users (affecting, e.g., also the size of the data storage for pre-calculated analytic results)

Clearly, this is only a first overview of the basic steps that are to be supported in such a scenario. However, it should become clear how important data-aware deployment recommendation and adaptation, as done by the Melodic framework, is within the CAS use case.

Architecture as-is

The CAS Open server is designed for anything relationship management purposes supporting strict multi-tenant separation. A partitioned traditional three-layer software architecture including physical separation between the layers determines the physical architecture of CAS Open. In general, CAS Open is a network of connected servers, operating as a Federated Cloud, over which CAS Software AG has jurisdiction. One or more CAS Open Server instances may serve requests from different users belonging to different tenants, at the same time, as depicted in Figure 1.

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

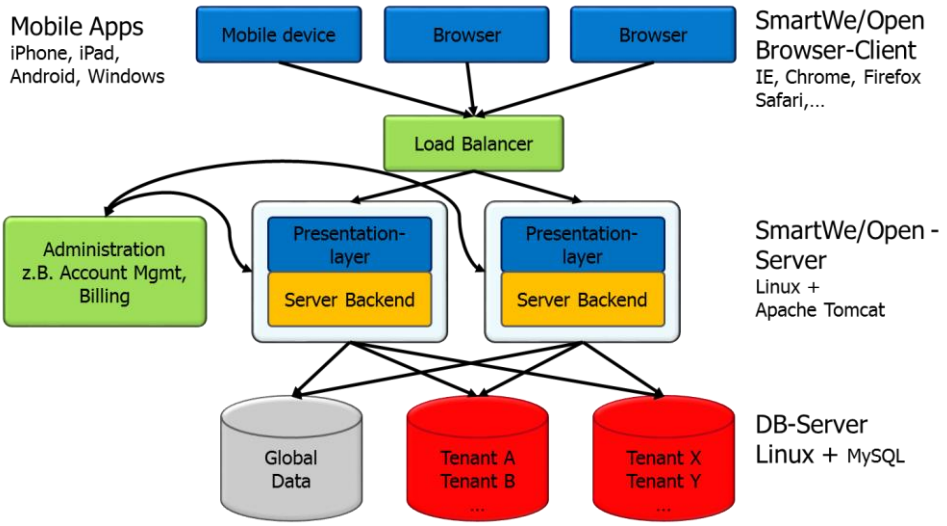

Figure 1: CAS SmartWe and OPEN Deployment

Furthermore, multi-tenant support leads to possible hosting of tenants on shared servers. To secure sufficiently sensitive data of a specific tenant's related customers, it may be allowed to operate private nodes in the Federated CAS Open Cloud while CAS Open architecture includes a strict separation between the data belonging to different tenants.

The data tier consists of one or more relational database management systems (RDBMS). Separation of tenant data is achieved by storing each in its own database. Apart from this, the data services offered are straightforward CRUD operations (create, retrieve, update, delete). Access to data is handled via an abstraction layer in the business logic tier, which is database-independent, as shown in Figure 2. Customised services usually require customised data types, for which the abstraction layer supports plugins for custom data access managers.

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

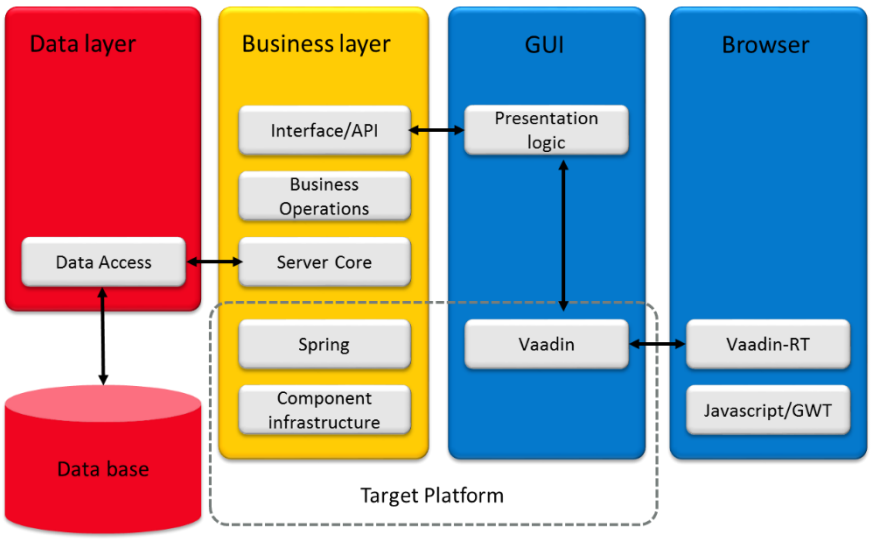

Figure 2: Overview of CAS OPEN architecture

The business logic tier is made up of multiple *CAS Open Server* instances. The CAS Open Server is one cornerstone of the CAS Open platform and serves as a central point to create, manipulate, store and retrieve xRM-specific data. The CAS Open Server is responsible for connecting to the DBMS and for encapsulating database-related functionality like transactions behind a high-level API. This API is the EIMInterface, the single gateway through which calls may come from external web services, external RMI calls or from direct administrative requests to manipulate data.

Apart from controlling data manipulation, the *CAS Open Server* provides a registry that hosts business logic operations supporting the xRM services. These operations allow addition of new tenants and users, the setting of user preferences and passwords, the management of authorisation rights to data, the logging of all changes to data, and the linking or tagging of data to obtain aggregated views.

An important property of the *CAS Open Server* is that it is stateless. No state or session handover has to be performed if one wants to switch to another server instance. This easily enables load balancing, i.e., multiple servers can share the workload. CAS Open Server is specifically designed for good scalability. A higher workload due to an increasing number of tenants and users can easily be addressed by adding new server instances and employing a load balancing mechanism. However, common administrative tasks, like cache synchronisation, are still necessary.

The presentation tier is the declaratively defined HTML5 SmartDesign based SmartWe. There are also native mobile solutions for iOS and Android developed separately.

The clients receive output from the *CAS Open Server* business logic layer, and maintain the state of visual widgets that are rendered on the client browser using JavaScript. Synchronisation is

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

handled by an AJAX connection, which supports a constant trickle of data in both directions, without needing to refresh the web page.

Architecture to-be with Melodic

As depicted in Figure 3, the backend of an app (in this case a data mining backend) should be hosted on another server infrastructure capable of scaling the needed hardware resources (CPU, RAM, Storage) according to the needs of the (data mining) application. These needs may vary within hours depending on the analysis to be performed, on the number of users using the analysis functionality (in form of a dedicated app), on the number of current changes of the data in the live system, and so on.

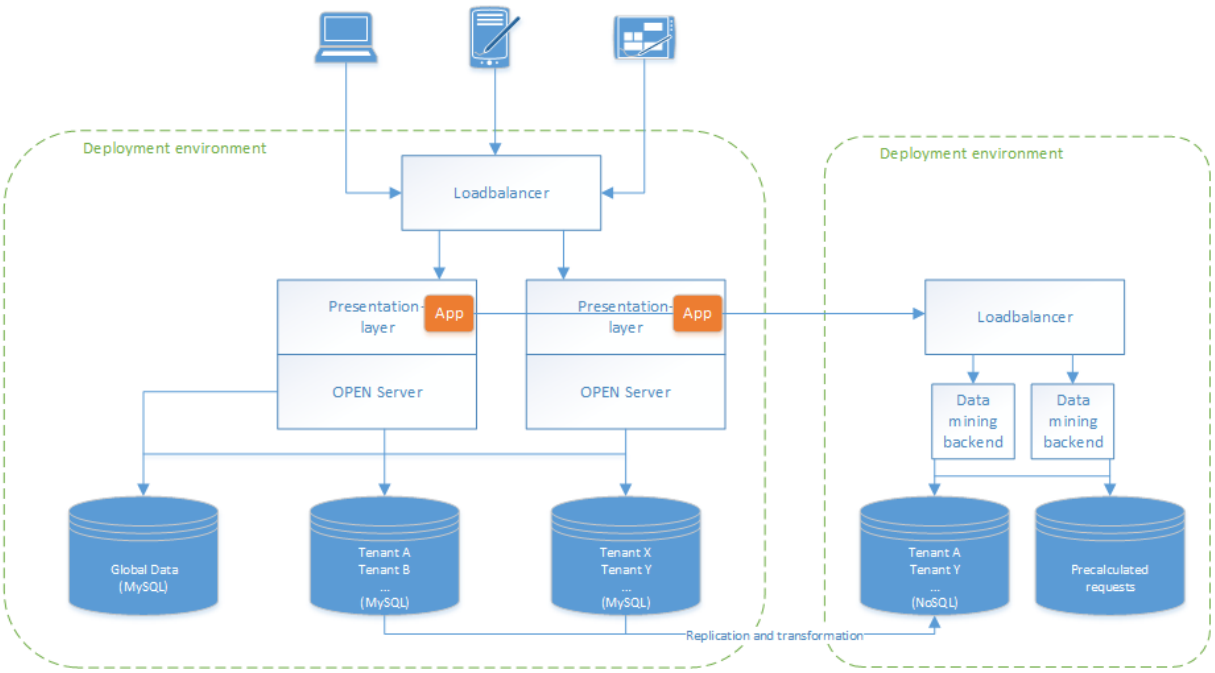

Figure 3: CAS deployment architecture with Melodic

As described earlier, within the CAS use case, the app store will be integrated with the Melodic framework, as shown in Figure 4, in order to let an app developer describe the app and get a deployment environment recommendation feedback. The following figure shows the high-level architecture of the CAS app store with the Melodic middleware.

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

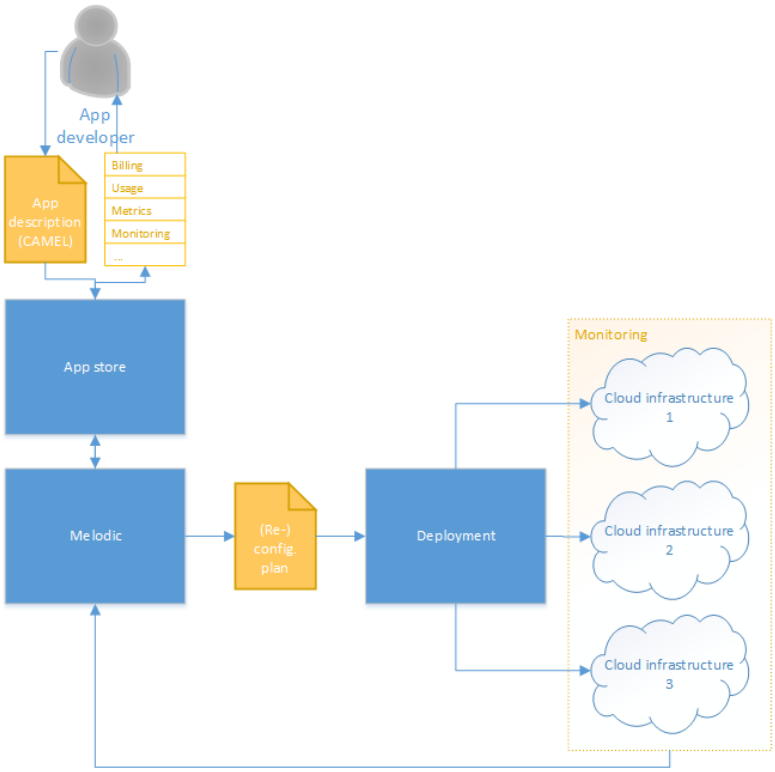

Figure 4: CAS app store architecture with Melodic

Concrete requirements for the Melodic framework are gathered in Jira[21]. An overview of high-level, abstract requirements is also presented in *Appendix A*.

Data related information

Table 2: Data related information for use case 1

Item	Description
Data	• Data is stored in relational database (MySQL) and documents in the file system
Workflow	• The workflow how the application is used is as follows:[22]

[21] https://jira.7bulls.eu/
[22] A tenant is a customer of CAS using the xRM/CRM system. A tenant has users being the employees of the tenant company.

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

Item	Description

	• Admin of the tenant registers and account is created
	• Admin creates user accounts
	• Users are using the xRM system and are thereby creating data
	• Admin/users can import data
	  • Addresses (Vcard)
	  • Appointments (iCal, Exchange)
	  • ERP data
	  • Any other customer specific data
	• Users are linking data records (e.g., linking an address with an appointment)
	• Admin is setting permissions on data types
	• Users are setting permissions on data records
	• Admin creates own data types, apps, views, forms, reports, etc.

Services	• Not relevant

Software Components	• Java application
	• Java Runtime Environment on which a Java web application is hosted
	• Inside this web application, the SmartDesign HTML5 Client is implemented using Vaadin as the basic web framework (multi-session and -user application)
	  • two different scenarios are supported:
	  • directly run the SmartDesign HTML5 Client binary using an embedded Jetty Servlet Container
	  • run the binary inside a standalone installation of a servlet container
	• Https load balancer
	• Server side component
	• Relational database
	• File system storage

Platforms	• Could be installed on any platform which supports Java.
Organisations	• CAS acts as platform provider and solution provider

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

Item	Description
	• CAS has a contract with a data centre for hosting the platform and solution[23] • CAS partners can adapt existing apps and develop new apps (software modules) • Customers can adapt existing apps and develop new apps • Customers can host their own platform and solution • With Melodic it will be possible for CAS, CAS partners and customers to get their own deployment environment for data- or computational intensive apps.
Persons	• Not Applicable
Size	• Depending on the customer • Current maximum: 300 GB in the data base + 16 GB in elastic search + 950 GB in the file system for document storage • Most of the CRM data in the data base is created in the journal[24] • Partitioned per tenant in the SaaS setting or per installation
Security and Privacy	• Not applicable
Storage	• File system • Relational database • Elastic search

4.2 Use Case 2 – Road Traffic and People Flow Monitoring

CET is a provider of traffic and mobility information services for private businesses and the public sector. Traffic services are based on Floating Car Data technology: anonymous GPS data of commercial fleets and navigation devices are processed to estimate behaviour of the traffic flows. People flow (mobility) services are based on the anonymous transactions describing movement of mobile phones between antennas of a mobile operator network. In both cases, provision and

[23] By *solution,* we mean a dedicated SaaS offering. SmartWe is such a solution being the standard xRM product CAS offers.
[24] The journal stores all changes of a data record over time. Therefore, the existing data is multiplied many times.

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

development of services are challenged by growing amount of the data, complex processing as well as time and cost constraints.

a) Data-intensive application for people flow (mobility) monitoring and analysis based on anonymised signalling data from mobile operator network

Data about people flows is essential for governmental agencies, cities, municipalities, as well as private business owners. Such data, to some extent, was traditionally gathered by conducting short term tourism surveys which are economically inefficient and limited in time and geographical scope.

CET is using anonymised signalling data from mobile operator network for monitoring and analysis of people flows (mobility), such as counting visitors of sites and advanced event tourism statistics, origin-destination analysis as well as concentration measurement of people in real-time

Current solution for people flow monitoring and analyses was sufficient for addressing the needs of customers in a small country like Czech Republic and on-demand projects. Addressing needs of a growing number of customers, larger markets and more countries requires a scalable and flexible solution allowing to process more data within shorter time. The amount of available data grows as well. In the Czech Republic, CET needs to process 10x more data for the same period with respect to the beginning when the first services were launched.

Architecture as-is

CET has built all its mobility tools in Python language for its ease of use and availability of various, powerful data analyses capabilities. On the other hand, in standard, Python cannot utilise multi-core tasks, parallel computing, and so on. Current tools work with data stored in CSV and HDF files (HDF stands for Hierarchical Data Format files and shall not be confused with Hadoop Distributed File System - HDFS). On-demand processing is deployed manually and performance is limited to the resources of a single machine. The overview of the current architecture is presented below in Figure 5.

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

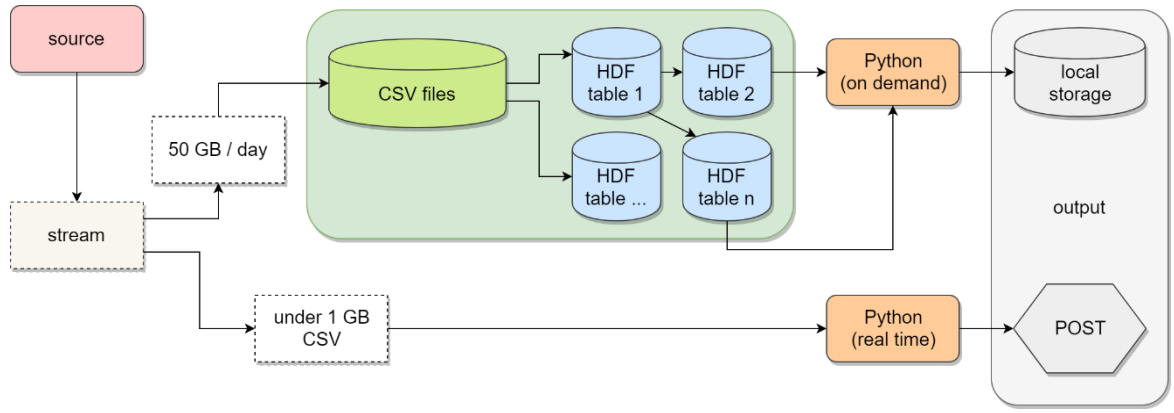

Figure 5: Current application architecture for people flow monitoring applications

Architecture to-be with Melodic

The new architecture will be based on big data platforms, such as Hadoop and Spark. CET will utilise all its experience and knowledge gained over the years of working with mobility data and implement its current and improved algorithms into new applications. These new applications will be able to take advantage of the big data platforms parallel processing capabilities as well as of the Melodic optimised deployment, scaling and resource management. Especially on-demand processing, depending on the project, may require to process vast amount of data within given time and cost restrictions. This could become a real challenge when there are more on-demand projects, potentially using the same data sets but different settings. To this end, Melodic will address this challenge by allowing the automatic or semi-automatic optimised application deployment based on provided meta-data and constraints. The overview of the architecture with Melodic as Cloud middleware is shown in Figure 6.

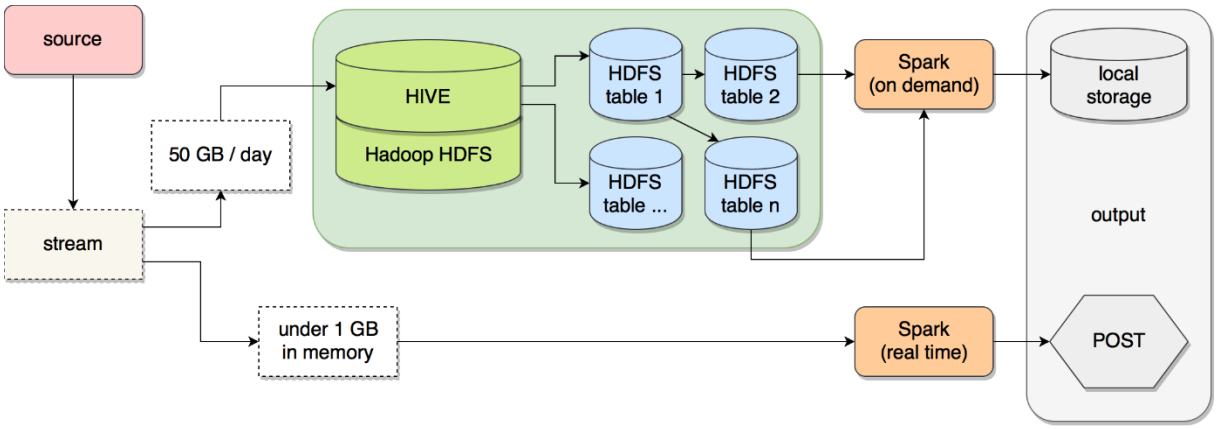

Figure 6: People flow monitoring application architecture with Melodic

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

Data related information

Table 3: Data related information for use case 2(a)

Item	Description
Data	• Real-time stream over TCP • HDFS tables with time series data and spatial data • Configuration files

Workflow	• Real-time processing workflow:
	  • Data stream received from the source is sent to two places: storage (Hive, for on-demand processing in the future) and real-time processing (Spark)
	  • Data is processed in memory: every 5 minutes data from last 1h (rolling window)
	  • Output data is posted to external private or public Cloud from where it can be consumed by external (customer) applications
	• On-demand processing workflow:
	  • User defines scope of the project (dates, locations, type of statistics, etc.) using set of configurations files and starts the application
	  • Data is accessed from Hive database processed by the application
	  • Results are stored locally as csv files

Services	• Offered by internal application platforms
	  • Storing raw and pre-processed data
	  • Real-time processing and posting results to external Cloud
	  • On-demand processing
	• Offered by external Cloud platforms
	  • TBD

Software Components	• Python applications • Spark • Hadoop • Hive • File system storage

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

Item	Description
Platforms	• Could be installed on any platform which supports above listed components.
Organisations	• Mobile Network Operator – owner of the source data, owner and administrator of the private Cloud • CE-Traffic
Persons	• Mobile Network Operator private Cloud administrator • CE-Traffic system administrator • CE-Traffic system user
Size	• Continuous real-time stream is incrementing HDFS warehouse • Daily amount of data is about 50 GB • In case of on-demand analysis, depending on the project, it is required to process data from single day up to many months so up to a few TB • The numbers are valid for a small country. In case of a bigger country and a larger number of subscribers, the amount of data will grow proportionately.
Security and Privacy	• Raw input data must stay in private Cloud of Mobile Network Operator • Connection is served via Cisco LAN2LAN IPsec tunnel
Storage	• Data are stored in Hadoop/Hive HDFS • Local configuration files outside of HDFS

b) Real-time traffic management based on the Floating Car Data and advanced traffic simulations

Vehicular traffic has an impact on the whole society, economy and environment, so it is crucial to analyse it, predict its evolution and manage it properly. However, it is not an easy task, as traffic is a complex phenomenon involving many heterogeneous agents (e.g., people, cars, public transport, and traffic signals) and depending on many factors (e.g., weather, mass events, road works etc.). To support these analysis, prediction and management tasks, it is important to have a sufficient amount of a real-world traffic data.

When traffic conditions are typical, it is sufficient to apply pre-built models and traffic management methods, which, if required, can be slightly modified based on real-time traffic data,

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

to give better accuracy and performance. However, quite often traffic conditions are atypical as a result of:

- Planned actions (e.g., road works, mass events),
- Events which can be predicted (e.g., bad weather),
- Events which are very hard to be predicted (e.g., car accidents).

All of the above cases may require performing actions which should change the typical way in which traffic is managed in order to prevent or resolve a crisis (e.g., occurrence of traffic jams). For example, possible actions may be related to traffic signal control, variable speed limits, traveller information systems (e.g., variable message signs, recommended routes). To run an appropriate action, a traffic management system must have accurate information about the current and predicted state of traffic, and typically it is required to have such information available as fast as possible. However, obtaining such information may require quite rapidly processing large amounts of traffic data (Big Data) in a cost-efficient way. Thus, it might constitute an important application of Melodic.

In particular, we are planning to use Melodic for finding typical and atypical profiles of traffic conditions and predicting traffic in typical and atypical conditions. This may require training and applying machine learning algorithms (e.g., neural networks) using real-world traffic data (e.g., FCD data) and running realistic traffic simulations. In some cases (e.g., finding typical traffic conditions, training machine learning algorithms) required actions may be performed offline (e.g., once per day / week /month), while in other cases (e.g., applying simulations and machine learning algorithms to traffic predictions) actions should be run in real-time, as fast as possible. Results of traffic data analysis and predictions may be applied in a real-time traffic management system. An overview of such a system is presented in Figure 7.

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

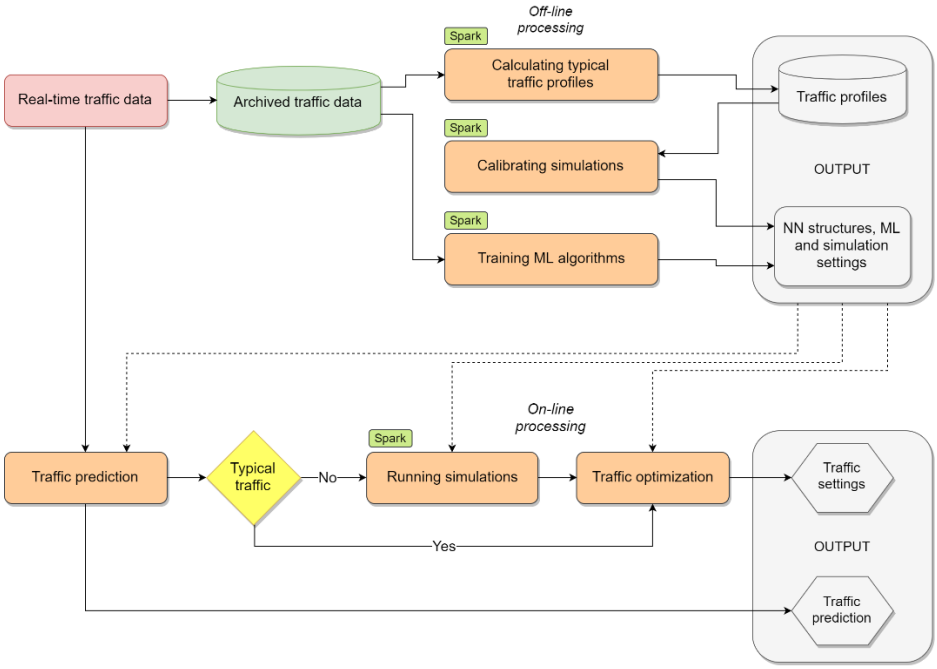

Figure 7: An architecture of the real-time traffic management system

Traffic data are archived and used for finding different profiles of traffic (e.g., typical traffic, free-flow traffic, traffic jam). For each profile, Traffic Simulation Framework (TSF) may be calibrated to simulate / predict traffic in a given conditions. Also, TSF may run simulations with different input settings and produce output, such as congestion, travel times, average speeds, total waiting times, so it may be used to evaluate a large number of traffic control settings, e.g., traffic signal settings, to find optimal settings for given traffic conditions. However, as the number of control settings is large, it is not easy to find the optimal one. To this end, the proposed approach takes advantage of AI / machine learning methods, for which TSF can be used to evaluate a large number (e.g., 100 000) of traffic control settings for each meaningful traffic conditions profile, in parallel, in a computational cluster (using Hadoop, Spark and, possibly, GPUs). These evaluations are later used to train (partially offline) machine learning algorithms (e.g., neural networks, using GPU, and, possibly Spark or other Big Data processing tools) approximating outcomes of traffic simulations very fast (a few orders of magnitude faster than by running simulations) and with a very good accuracy (up to 99%). Thus, it may be possible to evaluate very fast even larger sets of traffic control settings. To the find best settings, metaheuristics (e.g., genetic algorithms) may be applied (again, Spark / Hadoop may be useful to parallelise computations).

It is expected that, within the frame of the Melodic project, at least the following actions will be taken:

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

- Building different profiles of traffic conditions (by analysing archived traffic data using Apache Spark and data clustering methods, e.g., Self-Organising Maps): typical / crisis,
- Detecting current and predicted traffic states from data,
- Calibrating and running traffic simulations (TSF software),
- Training machine learning algorithms (e.g., neural networks) for traffic prediction,
- Training machine learning algorithms (e.g., neural networks) for evaluating traffic control strategies,
- Optimisation of traffic (applying metaheuristics, e.g., genetic algorithms, to find sub-optimal traffic signal settings)

.

Architecture as-is

New development will utilise the following existing applications and data resources:

- CET Floating Car Data system which provides real-time traffic information about speed, travel-time, delay and level-of-service information on the monitored road network.
- Archived traffic information (information published in real-time is being archived)
- Traffic Simulation Framework, as depicted in Figure 8, is a software for running traffic simulations in microscopic and mesoscopic models in a large scale. TSF was used in many research projects to run experiments related to traffic prediction, traffic analysis and traffic optimisation, but was never used in a production environment. Currently TSF can be run on a single machine using a single process, but it is planned to parallelise computations using GPU, Cloud and big data technologies.
- TensorTraffic is a tool for approximating outcomes of traffic simulations (e.g., the total waiting times on a red signal) using neural networks. The tool was developed in Python and is based on TensorFlow[25] library. It uses as an input data produced by TSF. TensorTraffic is available according to the MIT license.

[25] https://www.tensorflow.org/

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

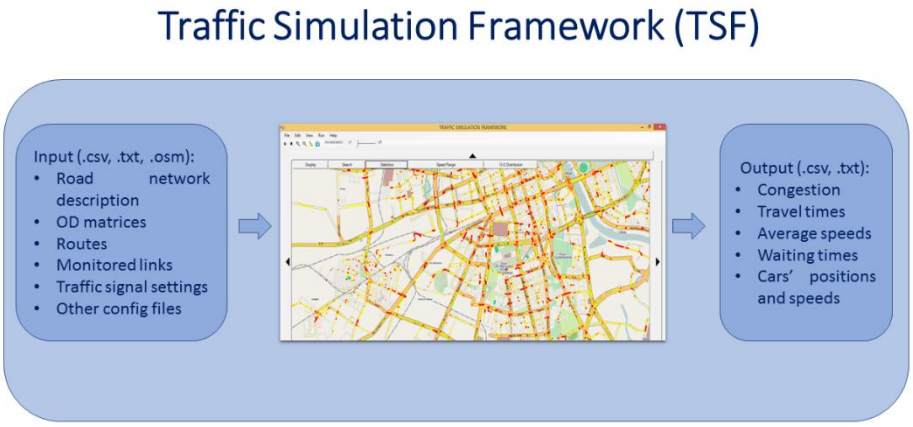

Figure 8: Traffic Simulation Framework Software

Architecture to-be with Melodic

New system will be based on the existing applications and data resources as described above and is intended to be a platform for new, on-demand and real-time traffic services. The basic architecture of the system will remain the same, as shown in Figure 7, with offline and online processing parts, but will be extended to utilise Cloud services through the Melodic middleware platform. A detailed application description scenarios are provided in Table 4.

Table 4: Scenarios for real time traffic management

Item	Description
Platform For	• On-demand analysis, e.g., training (and application) of ML algorithms, finding typical traffic conditions and other profiles of traffic • Real-time traffic prediction • Traffic simulations in real-world conditions
Data Sources	• Floating Car Data (FCD) and data from mobile network will be used for traffic analysis and for building offline and online traffic models. Offline traffic models will represent different traffic conditions (typical, free-flow, crisis), will be built based on historical data stored in databases and will be updated regularly (~ once per

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

Item	Description
	week). They may contain origin-destination (OD) matrices (i.e., matrices describing number of travels between each pair of communication areas in a given region) with respect to each meaningful time of a day/week (e.g., Monday 7:00-8:00am) and mode of transport, as well as traffic assignment (distribution of routes per each OD pair) and number of travels, travel times / travel speeds for each significant road segment. The presented scope of models and data is only preliminary. The precise scope of models will depend on available data and requirements for traffic predictions and traffic optimisation, so it has to be further elaborated. It is expected that offline models will correspond to different profiles of traffic (e.g., free flow, typical traffic, traffic jam), which can be inferred from data. Online traffic models will be built based on offline models and real-time FCD and mobile network data. They will contain similar information as offline models, but with respect to the current and predicted (short-term, i.e., 15-30 minutes ahead) traffic situation.
Use Scenarios	• Building profiles of traffic conditions by analysing available, archive data – Hadoop / Spark may be used to compute aggregated values (e.g., average travel times), clustering methods (e.g., Kohonen nets [8]) may be used to find crucial traffic profiles (e.g., free flow, typical traffic, traffic jam / crisis) – there may be different profiles for different cities / areas • Training machine learning algorithms for traffic prediction (CUDA / GPU support may be required to accelerate training, many machines / Spark may be required to train models for different areas and different traffic conditions) and for evaluating traffic control strategies • Running many traffic simulations with different parameters to generate training sets for neural networks and to estimate missing or uncertain data • Detecting current and predicted traffic states from data (many machines / Spark may be required to apply detection / prediction methods (e.g. Kohonen nets, recurrent neural networks) for different areas)

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

Item	Description
	• Traveller information module: sharing information about traffic conditions for users to make better trip decisions (e.g., what transport mode to choose, when to start travel, which route to take). Since type and format of data may influence travellers' choices, traveller information module may be considered as a traffic management system. • Applying metaheuristics (e.g., genetic algorithms) for finding sub-optimal traffic control strategies (possible parallelisation using Spark). It may be realised by making evaluations of short-term traffic conditions for different settings using TSF simulations or neural networks and applying metaheuristics (e.g., genetic algorithms) to find good settings. It can be initialised when system for a traffic monitoring / analysis and prediction will detect that an undesired traffic situation (e.g., car accident, traffic jam) has occurred or will likely occur.

Data related information

Table 5: Data related information for use case 2(b)

Item	Description
Data	• Real-time traffic: XML, CSV files updated every minute • Archived traffic: currently compressed XML files • Typical traffic / different traffic profiles: currently compressed CSV files • For efficient processing archived and typical traffic information may be downloaded and stored in a chosen database • Road network structure (.osm, .csv, .txt, .shp files) • Potentially other, external data sources related to the road traffic, e.g., weather data

Workflow	• Real-time processing workflow:
	  • New file with real-time traffic is downloaded every minute
	  • Traffic prediction and evaluation using ML algorithms run once per each minute. Data is processed in memory: every minute data from last x minutes (rolling window)

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

Item	Description
	• In case of typical traffic conditions traffic prediction is published • In case of detected crisis/untypical situation recalibrating traffic models and/or retraining ML algorithms. Many (1000-1000000) instances of simulations and ML algorithms (Hadoop/Spark). • Output data is posted to external private or public Cloud from where it can be consumed by external (customer) applications • Periodic / on-demand processing workflow (run once per day / week / month / …): • Typical and/or archived data is accessed and processed • Calibration of traffic models and training of ML algorithms • Results are stored locally as csv files
Services	• Offered by internal application platforms • Calibrating of traffic model • Training of ML algorithms • Finding typical traffic conditions • Real-time traffic prediction • Finding (sub)optimal control settings if crisis is detected, predicted or on demand
Software Components	• Python, Scala, C# applications • Spark • Hadoop • Mono.NET • CUDA (GPU support) • Python 2.7 ML / AI libraries (TensorFlow 1.0, numpy, scikit-learn) • relational database (e.g., Postgres, MySQL, SQL Server) • Big data store (e.g., HBase) may be required • Docker / Kubernetes may be required to scale computations in a (virtual) cluster.
Platforms	• Could be installed on any platform which supports above listed components.
Organisations	• CE-Traffic
Persons	• System administrator • System user

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

Item	Description
Size	• Real-time traffic: size of each file: 0.5MB – 2MB (zipped, ratio 10-15) updated once per minute • Typical traffic size: 100MB – 1GB updated once per day / week / month / quarter • Archived traffic: approximated size: 1TB / 1 year of data
Security and Privacy	• Advanced options not relevant
Storage	• Amazon S3 • HDFS • Some files outside of HDFS

4.3 Use Case 3 – Secure Document Management

The FCR Secure Document Management application is a SaaS solution for secure document management with full access/flow control and approving operations with digital signature. The FCR application is designed mainly for the banking sector and is compliant with the strictest security regulations. The system has been so far implemented in finance and medical sectors, where there are large security requirements and the amount of communications is massive. The companies like FCR are interested in improving their services, in terms of efficiency and security (the level of security suitable for banks and other financial institutions). For now, this solution relies on a private Cloud and it could actually benefit from the diversification of resources by using public Cloud with no vendor lock-in.

Architecture as-is

The architecture as-is of FCR application is a typical three-tier architecture, with the following tiers:

- Client side: Java applet running within web browser.
- Server side: Java, Spring, Spring Boot, Spring security.
- SQL database (Postgress, MS SQL, Oracle) and documents stored in file system in encrypted form.

Two types of data is being used in the application.

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

- Data stored in relational database: users, permissions, index and metadata for documents, structure, auditing and so on.
- Data stored in file system: encrypted documents.

The server side part of application could be scaled horizontally and there could be many instances of the server side part of the application. This element of application is compute intensive due to many cryptographic operations.

The current architecture of the FCR application is given in Figure 9.

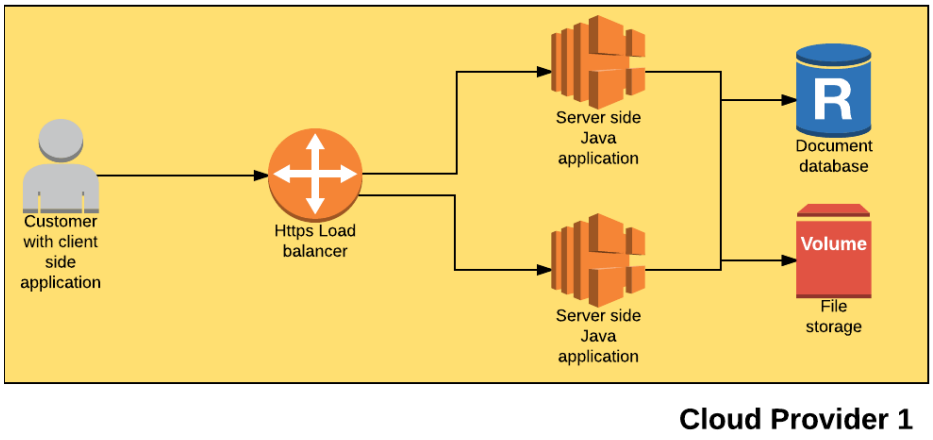

Figure 9: Architecture for FCR application

Architecture to-be with Melodic

Melodic will provide an easy to use Multi-Cloud environment for the benefits of both infrastructure providers and innovative SMEs deploying data-intensive applications in the Cloud. The document management solution offered by FCR requires Multi-Cloud to provide a highly secure processing of big data for mass communication of financial institution with its customers. It could actually benefit from diversifying the resources by using both private and public Cloud. With Melodic, it could be a semi-automatic operation. FCR will benefit by optimising costs (by scaling private clouds to typical, not expected maximal loads) and eliminating vendor lock-in (using a number of public clouds). This model will work for any organisations using an own private Cloud and has a large commercial potential.

For this use case we anticipate to use Melodic for scalability of the server side components using the Scalability Rule Language (SRL) part of CAMEL. We plan to optimise the cost of the

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

infrastructure using Melodic. Also the ability of Multi-Cloud application deployment will be an advantage for the FCR application.

Figure 10 provides anticipated architecture of FCR application deployed on the Melodic framework.

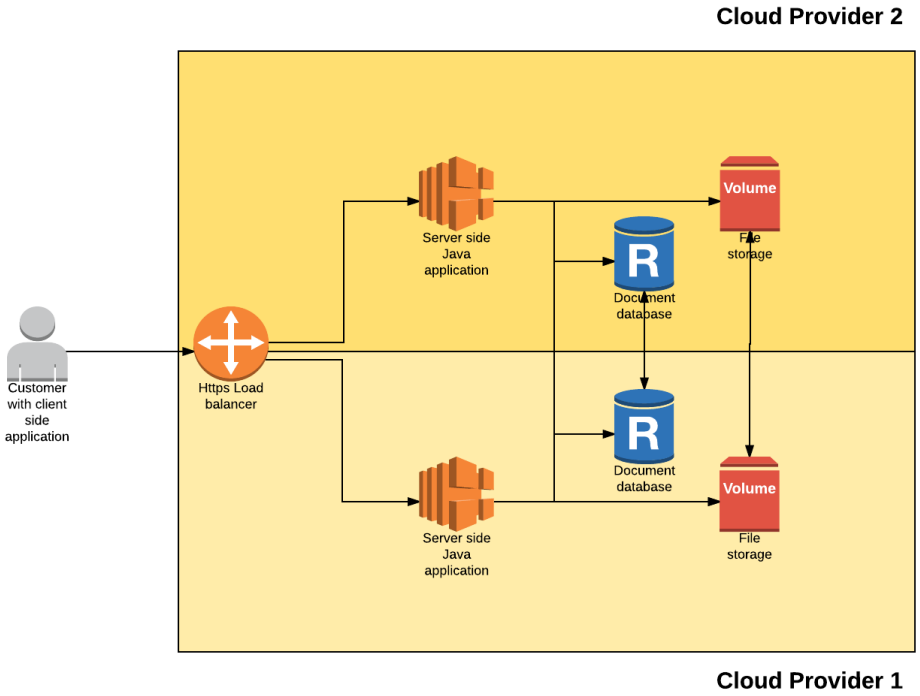

Figure 10: FCR application architecture with Melodic

Data related information

Table 6: Data related information for Use Case 3

Item	Description
Data	• Data are stored in relational database (Postgress SQL, could be any relational database) and files in file system.
Workflow	• The workflow how the application is used is as follows: • User uploads document which is encrypted by client side application

Item	Description
	• Encrypted document is transmitted to the server side component • The server side component checks if the document is properly encrypted and signed • The server side component inserts metadata of the document into the database and stores the encrypted document in the file system.

Services	• Offered by internal application platforms
	  • Secure document storing, with client side encryption
	• Offered by external Cloud platforms
	  • None

Software Components	• Client side application (Java applet) • Https load balancer • Server side component • Relational database • File system storage
Platforms	• Could be installed on any platform which supports Java.
Organisations	• The system could be used in a multi-tenancy model as a SaaS or on premises for one organisation.
Persons	• Not Applicable
Size	• Typical document size is 0.5 MB up to 5 MB, average size is 2 MB. There is currently around 10 000 documents, total size about 20 GB • Data could be partitioned because they are stored on file system, but it is not necessary (due to total size)
Security and Privacy	• The purpose of the application is to encrypt documents using client side encryption without sharing encryption keys, so the privacy is ensured by design.

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

Item	Description
Storage	• File system – could be partitioned • Relational database.

4.4 Use Case 4 – Biological data analysis

With the advent of the "Omics" era in the life sciences, researchers gained access to vast amounts of biological information, including data about genome, proteome, metabolome, transcriptome and molecular pathways just to name a few. The size of this data has now exceeded well beyond petabyte or even exabyte. As an example, the final results from 1000 Genome Project[26] consists of more than 200 terabytes of data. The forthcoming initiatives, like the 100,000 Genome Project[27], gives strong indications that the amount of data available for analysis will grow exponentially. To take full advantage of this data, scientists and developers will need to develop tools and platforms that will enable them to perform calculations on a scale well beyond a small cluster.

As a part of Melodic use-case, our research team will develop an application prototype that enables a robust approach for the discovery of synergistic variables in biological datasets, with a main focus on data from gene expression studies and genome-wide association study (GWAS). These datasets are often described with a large number of variables. Usually, only few of those variables are relevant for the phenomena under the researcher's investigation. Therefore, the *identification* of variables that are relevant for a given research is an important initial step of data analysis. The common way to identify the relevant variables is a univariate test for association between each explanatory variable and the response variable, but the univariate test ignores variables, which contribute information on the response only in synergy with others.

To successfully detect such relevant variables, it is necessary to examine all the k-tuples of variables. The multivariate exhaustive search requires huge amount of computations, which has made it impossible for a long time. shows number of tests required for 50000 variables, growing with the tuple size.

[26] http://www.internationalgenome.org/
[27] https://www.genomicsengland.co.uk/

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

Table 7: Number of tests required for 50 000 variables

k	Number of Tests
1	$5.0 * 10^4$
2	$1.2 * 10^9$
3	$2.0 * 10^{13}$

To overcome this stalemate situation we will be developing a prototype application that:

- can benefit from Cloud computing processing power and scalability (by complying to the Melodic application model)
- can make use of Graphics Processing Units (CUDA technology) for speeding up calculations
- applies a novel algorithm developed by the Faculty of Computer Science of Bialystok University, tailored for this problem

Architecture to-be with Melodic

Selection of the appropriate data for analysis is one of the important success factors of this use case. Initially, we will focus on open, publicly available datasets of genome sequence or genome expression data. One of the considered possibilities is the data from 1000 Genome project[28]. Data will be stored in the Apache Cassandra database. The workflow will follow the classical data bus as presented in Figure 11.

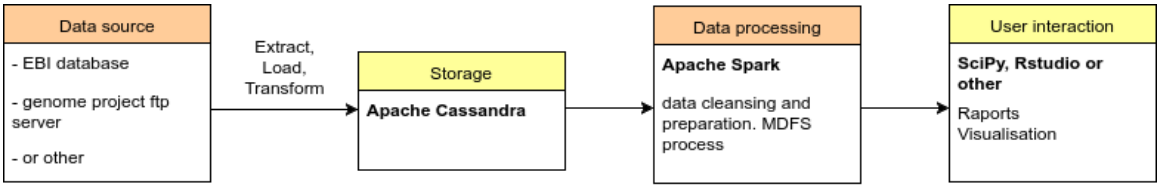

Figure 11: Workflow based on classical data bus

The architecture of the application will be based on the following frameworks:

[28] The data is freely availabile via ftp://ftp-trace.ncbi.nlm.nih.gov/1000genomes

- Spark - fast and general engine for distributed, large-scale data processing
- Mesos - cluster resource management system that provides efficient resource isolation and sharing across distributed applications
- Cassandra - distributed, highly available database designed to handle large amounts of data across multiple data centres
- Nvidia CUDA - technology for GPU parallel computing

A simplified application architecture is presented in Figure 12. Please note that this is the initial concept that might evolve in the later stages of development.

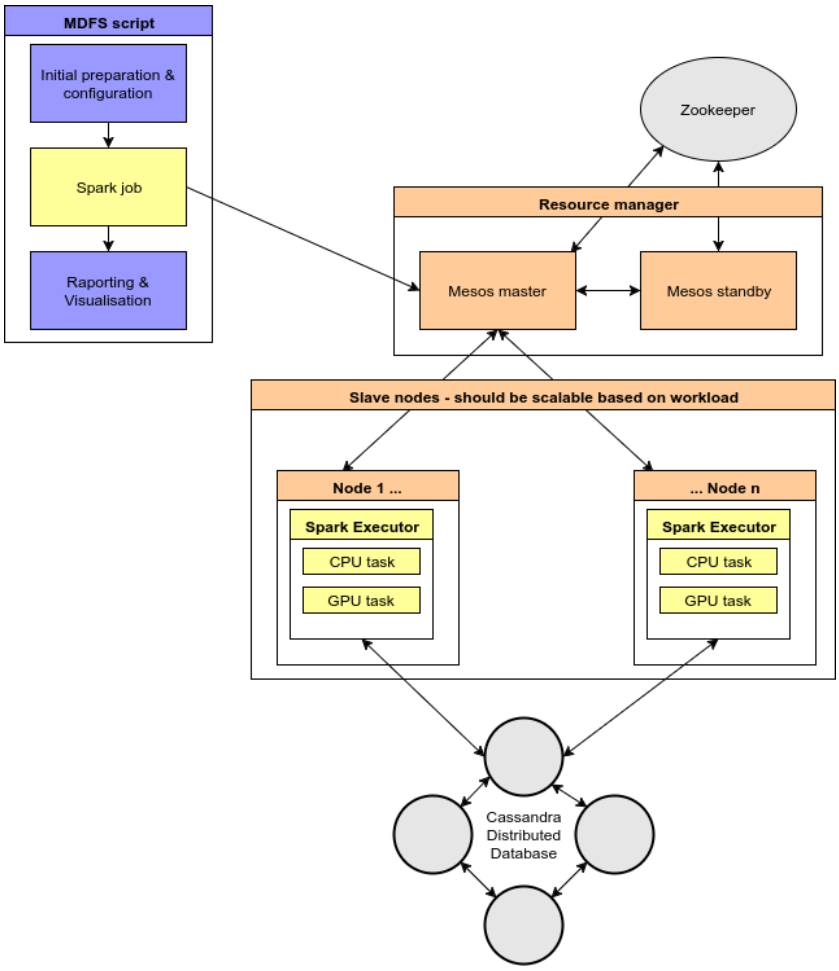

Figure 12: Preliminary application architecture for gnome analysis application

5 Non-Functional Requirements

In this chapter, we present non-functional requirements for the Melodic middleware platform. Non-functional requirements are gathered based on the best practices in software design and development. For a large development project like Melodic, it is important that the implementation follows a well-planned and established pathway related to both functional and non-functional requirements so the project goals can be achieved within the committed time and resources. Two important principles, on which the Melodic platform will be designed and developed, are extensibility and reusability. The Melodic plans for extensibility from the very start, and will re-use existing open technologies and tools to reach high ambitions set for the project. Moreover, it is required that the developed components and software artefacts are of high quality, well-documented, and easily maintainable. Furthermore, sustainability of the project beyond the commission funded period is taken as an important non-functional requirement regulating the project plan and implementation. In the following, we discuss non-functional requirements for the Melodic in detail.

5.1 Extensibility

Extensibility is an important design principle in software engineering and system design in which system implementation considers future extensions. It can also be seen as the systemic measure of the capability to expand the system and the level of effort required to perform this expansion. Logically speaking, a system can be extended either through the incorporation of new functionality or the modification / enhancement of existing ones. However, the extension of the system should be done in such a way that impact on existing system functions is minimised. This surely includes both the system internal structure and data flow. Extensibility also facilitates systematic reuse as enforces the development of components which can be added to different systems on demand in order to enhance their existing functionality.

Extensibility also affects software design. In particular, software should be designed with the principle that not everything can be covered from the very beginning. This leads to developing initial, lightweight software frameworks which allow for continuous development and enhancement by also preventing the occurrence of issues like low cohesion and high coupling. Such a design builds on the principle of applying change through addition by having each system part workable with any change. An extensible system design also caters for frequent re-prioritisation and enables the on demand development of functionality in small steps. In this way, extensibility leads to less and lower dependencies during development time, high cohesion and low coupling as well as to the determination of well-defined interfaces.

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

Based on the above analysis, extensibility is an important property for a system which relates to known software engineering issues like low cohesion and high coupling as well as to high development time for systems which are not planned to exhibit such a property. Such a property is also essential for systems which tend to evolve over time in order to, e.g., also cater for the satisfaction of evolving business requirements as well as the change of context (e.g., environmental changes, modifications to external components, etc.).

Based on the way Melodic will be built, it needs to cater and plan for extensibility from the very beginning. This is essentially true as Melodic will realise a system which will be initially built based on existing components mainly borrowed from a set of research prototypes and then expanded with the capability to support data-aware adaptive Multi-Cloud application provisioning. Moreover, data-awareness will be gradually realised which means that the system will still evolve over time until it will reach a stable optimum. By also considering that Melodic will rely on external technologies which evolve over time, it will also need to be able to also change to cater for such evolution by considering that new versions of technologies will be more improved and might also incorporate new functionality which could be interesting to be include in the Melodic system. The latter could also be true even for new technologies if we consider the current landscape in big data and Cloud computing and the current pace of its evolution and enhancement.

There exist different types of extensibility: white-box, black-box and grey-box. White-box extensibility relates to changes that can be performed in the system source code and is the least restrictive type of extensibility. Black-box extensibility relies on the definition of clear interfaces via which a system can be expanded and is the most limited form of extensibility. Such form of extensibility can then be applied by either system configuration applications or application-specific scripting languages. Grey-box extensibility is a compromise between black-box and white-box extensibility that does not rely on the full exposure of the system source code. In particular, developers are given the system specialisation interface that includes all abstractions for refinement as well as guidelines for how the respective extensions should be implemented.

Melodic will support white-box extensibility for most of the components involved in its prototype system. Black-box extensibility will be considered only in the case of external components which will be integrated via well-defined interfaces in the system. This type of extensibility will also cater for the external components evolution as the well-defined interfaces will enable the system to independently change and more flexibly adopt new external component versions on demand.

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

5.2 Reusability

Reusability refers to the capability to re-use existing assets in the software development lifecycle. Such assets can be products or by-product of this lifecycle and might include code, software components, designs and documentation.

Code re-use might imply the need to create a separate version of the asset being re-used. A software library is a nice example of code re-use where developers can create internal abstractions to enable the re-use of certain program parts or might create custom libraries for their own internal use. Reusability brings about several aspects of software development that need to be considered and which imply the explicit handling of build, packaging, distribution, installation, configuration, deployment, maintenance, and upgrade issues. If such issues are not handled, they may appear to be re-usable according to the design point of view, but not in practical terms. Software reusability can also be enabled only when certain design features of the corresponding software are realised such that it is made suitable for reuse. These features include at least the following: adaptability, consistency, correctness, extensibility, modularity, stability and flexibility.

There are different classifications of re-use which depend on the respective dimension concerned. With respect to motivation, reuse can be *opportunistic*, i.e., software / asset is reused on demand when the respective opportunity arises, or planned where the development team design components that can be reused across different projects. Concerning ownership, reuse can be *internal*, where the development team attempts to reuse internal components, possibly to also have the ability to better control them, or *external*, where external third-party components are licenced to be used. The latter type of reuse maps usually to 20 percent of the development time required to build the required component from scratch but the development team should also consider the extra time needed in order to discover, learn and integrate this external component. With respect to reuse structure, reuse can be categorised as *referenced*, where the developed code references the reused code such that each code is independently developed according to its own production lifecycle, or *forked* which means that a private copy of the reused code is exploited such that the developed software and reused code map to the same lifecycle and versions.

Reusability is important in system / software development as it leads to saving time and money as well as to the reduction of redundancy. As Melodic also relates to the production of a platform with a limited horizon of resources and time, it needs to also cater for re-usability as much as possible. This will enable it to not reinvent the wheel by building the platform from scratch but re-use existing state-of-the-art functionalities in order to more rapidly develop the respective platform which satisfies the functional and non-functional requirements posed.

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

Indeed, Melodic considers the re-use of components from the very start of its conception and existence. First, as it has been specified to exploit existing components that support the multi-Cloud management of Cloud applications, which have been developed in state-of-the-art European research projects. Second, as also indicated in this deliverable, by continuously observing and evaluating the developments in the area of big data processing and orchestration in order to re-use the most promising and prominent technologies according to the project main requirements. In both cases, components are or will be systematically reviewed in order to examine whether they can be used as they are or need to be modified which relates to the aforementioned reuse structure types (referenced or forked).

Finally, we should mention that Melodic will be developed and offered as a software which can be re-used by different applications and systems. Its reuse is guaranteed by following a service-oriented architecture which enables to provide the right abstractions for re-use. This means that while Melodic might evolve over time, its external interfaces will remain stable and this enables to independently interface any component or application to it. Such a feature has been deemed quite important in order to outreach the correct communities which could enhance in turn the further sustainability of the Melodic platform.

5.3 Documentation

Software documentation refers to documents or illustrations that accompany a developed software. It is an important part of software engineering and spans the following types: (a) *requirements* which provide statements relating to the identification of the attributes, capabilities, characteristics, and the quality of the system, mapping to the foundation of what should be developed; (b) *architecture / design* which provides an overview of the software which includes the relations to a particular environment and the software construction principles; (c) *technical* which maps to the documentation of code, algorithms, interfaces and APIs; (d) *user* which relates to manuals being developed for different roles in the system like administrators, developers and end-users; (e) marketing associated with the specification of how to market the software product as well as of the analysis of the market demand.

Software documentation is very important for the following reasons: (a) it enables to keep track of all aspects of software development; (b) it enables to improve software quality; (c) it makes information easily accessible; (d) it reduces the number of user entry points; (e) it allows new users to learn quickly; (f) it assists in cutting support costs. Based on all these benefits, Melodic plans to invest on the documentation of its platform in order to assist in the internal and external use of this platform as well as the software development tasks performed by the development team in the project. In this respect, all types of documentation will be touched mainly in the context of

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

project deliverables. This mainly indicates that Melodic from its initial conception has really focused on producing the right documents related to the software platform to be developed. In addition to deliverable documents, guidelines will be given to the development team about how the code has to be documented and which tools will be used for the production of the code documentation. This will ensure that a good and possibly uniform level of documentation is retained which assists in the production of good quality code and the more successful and rapid integration of the Melodic platform components. As such, the user documentation aspect will be covered perfectly with different types of documents being carefully produced for that reason.

5.4 Quality

Software quality in the context of software engineering maps to two distinct notions: software functional quality and software non-functional quality. Software functional quality prescribes how well the software complies with a certain design and the respective set of functional requirements and specifications. It is usually assessed dynamically while code reviews have been also exploited for static assessment. On the other hand, software non-functional quality prescribes how well the software satisfies the non-functional requirements posed. Some non-functional quality attributes can be assessed statically based on the analysis of the software inner structure and its source code at different levels, including the unit, technology and system level. Others can be only assessed dynamically, like the *usability* one.

A de facto standard pertaining to the above two types of quality is ISO 9126-3, which has been followed by ISO 25000:2005, which explicates the structure, classification and terminology of software quality attributes and metrics that are applicable to software quality management. By relying on these standards, the Consortium of IT Quality (CISQ) has defined 5 major characteristics that must be exhibited by software in order to have a business value: reliability, efficiency, accuracy, maintainability and accurate size.

The aforementioned latter characteristics thus map to attributes and metrics which can be measured by applying different types of techniques at design or runtime, which include: (a) analysis of the source code, the architecture, software framework and database schema; (b) checking of software engineering best practices; (c) checking of bad engineering (architectural and code) practices; (d) production of an instrumentation and measurement system which is able to assess the quality of the delivered software during runtime.

The quality of software has a great impact over other software properties like reusability and extensibility. A bad quality software would never be reused by application users or developers while it might also be very hard to extend leading to substantial re-engineering efforts. Bad software quality also leads to problems of instability and reliability as the produced software

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

might exhibit periodic or ad hoc errors. This leads to reduced software reputation and thus to reduction in gains and market share for the organisation that provides this software. As such, it is essential that appropriate methods, techniques and processes are involved in order to observe and assure the quality of the software produced which are coupled with respective guidelines on how such processes can be followed and applied.

As such, being aware of software quality criticality, the project has dedicated a certain task in its work plan called Quality Assurance, which has as main goals:

a) to produce a quality assurance guide which will comprise guidelines to developers on how quality software should be produced;
b) to identify the correct quality assurance processes that would enable to observe the quality of the Melodic software platform architecture, code and documentation.

Coupled with the task on Platform Integration and Testing, which focuses on how to integrate and test the Melodic platform components, the project will have all the appropriate mechanisms in order to react rapidly on identified quality issues in order to sustain an appropriate quality level for the software developed across the whole project lifetime.

5.5 Fault Tolerance

Fault tolerance maps to the capability of the system to continue operation even if one or more of its components have failed. A system can be fault-tolerant only when the following requirements are met:

a) there is no single point of failure;
b) there is fault isolation to the faulty component which also requires the capability to also include some dedicated fault-detection mechanisms;
c) the system is capable to impose fault containment to prevent fault propagation;
d) there is an availability of reversion modes.

Fault tolerance can be achieved in different ways based on the type of fault encountered. For non-catastrophic faults, the system can be designed to detect exceptional situations and address them accordingly by respective adaptation actions. If the fault is catastrophic or it is very expensive to keep a high level of system reliability, then duplication / replication might be considered. The system may also have to user reversion in order to backtrack to a safe mode. Two types of replication / redundancy apply: (a) *space* where additional components, functions or data items are supplied and made available – this redundancy type can be further classified into hardware, software and information redundancy; (b) *time* where the computation or data transmission is repeated and respective result is compared to a recorded copy of a previous results.

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

Replication comes with some penalties which span increased size, energy consumption, cost and time. To this end, it is not logical to make all system components designed to be fault-tolerant. This then creates the need to replicate only those components which are critical in terms of delivering the core functionality of the system. Other criteria which can play a role in the selection and replication of a system component can include the failure probability of that component and the cost of making that component fault tolerant.

Melodic intends to support the provisioning of data-intensive Multi-Cloud applications. In such kind of provisioning, the occurrence of faults is inevitable. This is well true if we consider the laws of probability over different parts of the system: (a) computation: it might well be the case that a single component, out of the multiple ones that Melodic platform comprises, might fail due to, e.g., one or more bugs that have not yet been identified for that component; (b) network: network errors or failures can suddenly occur leading to loss of data or making parts of the system unavailable / inaccessible; (c) out of the big data volume that has to be processed, one portion might become corrupted, lost during transit or even due to hardware failures. Apart from operational issues, due to the nature of the distributed environment we desire to deal with, problems can also occur during system / application deployment and re-configuration. For instance, a component may not be correctly installed due to errors in manually specified installation commands or due to temporary network problems that prevent the system from correctly downloading the component binaries.

To this end, by also considering the above analysis as well as the amount of resources and time available for the project, Melodic desires to realise some fault-tolerance features in its platform. In particular, concerning application deployment and reconfiguration, the approach that the project will follow would be to rely on adaptation / fault-tolerance policies. Such policies will attempt first to retry the failed deployment step. In case this fails, then another policy would be to retry a particular part of the deployment plan. Even if this fails, the deployment will fail and the system will revert to its previous normal state. The latter will be actually achieved by carefully applying only the changes needed by still keeping intact the original version of the deployed user application. In this sense, if the changes fail to be applied, then we just remove their products & by-products and we remain with the previous state of the application deployment.

Apart from the level of the deployment plan execution, Melodic will also apply fault tolerance on the level of communication/integration and process. Fault-tolerance at the former level will be achieved through the introduction of an ESB component through which the eventual delivery of the messages sent can be achieved. Such a component might also take care of switching endpoints, if one might become unavailable. Fault-tolerance at the process level will be achieved by the incorporation of the fault-handling logic inside the process specification in order to handle failures that span a certain process task or process fragment. Such fault-handling logic could take

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

the form of a well-defined sub-process which will take care of bringing the system to a normal state.

We should note here that Melodic will be able to handle faults that it can detect. If application-specific faults occur, then it is the responsibility of the application owner to design his/her application to become fault-tolerant (by, e.g., maintaining a full replica for some application components) or to provide the right requirements (e.g., scalability requirements for some application components) to the Melodic platform in order to take care of appropriately applying them. The application owner could also undertake the opportunity to make explicit to the system by providing suitable sensors, when and how application faults should be detected and how they can be confronted. In this way, the Melodic platform would have all the suitable information needed in order to appropriately handle this fault. In addition, apart from the aforementioned capabilities to overpass some Melodic system faults, it should be the responsibility of the application owner to raise the fault-tolerance level of the Melodic system by, e.g., maintaining multiple running instances for various Melodic components. However, in the latter case, the Melodic platform will provide some nice ways in order to appropriately configure its deployment.

Finally, please note that the Melodic system will attempt to offer some reconfiguration capabilities to the application owner which span mainly the infrastructural level. It will also attempt to cover the platform level, if this is considered appropriate by the use cases involved in the project. However, it is not the intention and the project does not have the resources in order to handle adaptations at the service level (i.e., replacing external SaaS exploited by the user application or recomposing a particular part of the user service-based application). Possibly this is a future work direction that could be realised by the community that Melodic will formulate or outreach during its lifetime.

5.6 Scalability

Platform scalability is an important non-functional requirement for successful production environments. A system is generally considered scalable when it is capable of handling increased load with a proportional increase in the available system resources. In the context of the Melodic project, the platform scalability can be defined as the capability of the Melodic middleware to handle a growing amount of work because of increased number of applications or complex resource optimisation requirements. Melodic, as a personal DevOps solution for a Cloud user or organisation, needs not to be scalable with respect to the number of users as it does not target multi-tenancy. However, the platform should be able to support possibly large number of jobs and data sources from a single Cloud user. On the other hand, including a scalable resource management layer can substantially reduce the load on the Melodic middleware and have a

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

potential to simplify reasoning for the optimisation, which in turn will increase platform scalability. Resource management is further discussed in Chapter 7 where we describe the criteria and selection of a scalable resource management system for the Melodic platform.

6 Technology Evaluation

The ambitious Melodic platform will be created as an integration of the available open source technologies, while providing the required extensions for efficient Cross-Cloud data-intensive computing. Melodic will utilise results from three European projects from the consortium partners: PaaSage platform for Cross-Cloud deployment and Cross-Cloud resource management, CACTOS platform for infrastructure resource management, and the PaaSword for transparent Cross-Cloud security and privacy. In this chapter, we briefly introduce the three projects, and provide assessment of the projects components and specify required extensions and modification for their reuse in Melodic.

6.1 PaaSage EU Project

The PaaSage project (FP7-ICT-317715) was a four year project running from October 2012 through September 2016 that pioneered the use of model driven run-time adaptive cross Cloud application deployment. The application was first modelled in a domain specific language, CAMEL, developed in PaaSage and describing the application components, their interconnections, the constraints on the deployment and the extra functional deployment goals and objectives.

The PaaSage "upper ware" was then responsible for finding the deployment *configuration* giving best possible application *utility* based on the deployment goals and objectives and the constraints of this model, and this configuration was then deployed to potentially multiple Cloud providers by the "execution ware" of PaaSage.

The CAMEL application model was first checked for validity involving a verification that the user had the necessary accounts for defined Cloud providers, and that there would be at least one solution possible given the various constraints. For instance, if there is a constraint that the application should be deployed in Europe, and the only Cloud provider given as alternative does not offer such a profile, the deployment problem has no solution. This was in PaaSage called the "profiling step" of the upper ware.

The validated model would still have many undecided parameters, called *variables*. This could typically involve the minimum and maximum number of instances to deploy for a given component type, and for each such range there should be fixed a Cloud provider and potentially also a virtual machine type. This is "reasoning" about the problem and its solutions, and was in

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

PaaSage implemented by using a mathematical solver trying to optimise the application *utility*. For this reason, the validated model was called the Constraint Programming (CP) model. The set of assigned variable *values* is called the deployment *configuration*.

The PaaSage "adapter" gets this configuration and turns it into a deployment script, i.e. a set of sequential actions needed to successfully deploy the application ensuring, for instance, that all instances are started before they will be connected. This task is trivial for first time deployment. A salient aspect of the Cloud is the elastic resources meaning that it will be easy to start another instance of a virtual machine, and the Cloud providers offers an interface to quickly start a copy of an already running machine. The *configuration* contains ranges of instances for this very reason so that an initial deployment can be minimal, i.e. containing only the least number of instances for each component type, and then scale using the Cloud provider's interface up to the maximum number of instances. This means that the adapter will need to maintain an updated view of the configuration actually running. At one point the number of instances might reach the upper limit of the range provided by the solver. Further scaling will *invalidate* the configuration found by the solver and requires that the optimisation problem is solved again. Hence, the solver will come up with a *new configuration*, and the difficult part of the adapter is to decide if it is worthwhile deploying this new configuration, or if the cost of reconfiguration offsets the additional utility of the new configuration for the current execution context. If the new configuration should be deployed, the adapter will create deployment scripts that will make the least possible changes to the running configuration to make it represent an enactment of the *new configuration*.

The execution ware receives the deployment script and translates this into Cloud provider specific interface calls. It installs a PaaSage "lifecycle agent" in each Cloud that is responsible for interacting with the PaaSage virtual machines instantiated in the same Cloud, and collect the monitoring information from these machines. CAMEL defines a S*calability Rule Language (SRL),* which is basically a set of *event-condition-action* rules where the event is monitored data to be compared with some threshold leading to a scalability action. Consider as an example the rule saying that a new web server machine should be started whenever there are more than 100 active users on all of the current web servers because 100 users per server is considered by the application owner as the maximum number of users for which the server is able to guarantee a good user experience. Whenever a monitored *metric* changes its value, the SLRs will be evaluated and the triggered actions done by the execution ware.

The *metrics* to monitor are defined in the CAMEL application description by the owner of the application, and connected to an available *sensor*. A sensor can either be a standard value available from the Cloud provider or the operating system of the virtual machines, like the memory consumption of a database, the CPU load, or the amount of input and output performed on a machine; or an application specific sensor, like the number of logged in users, or the

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

geographical location of the users or whatever is useful to decide on the application scaling and configuration. Furthermore, CAMEL offers the possibility to define a *composite metric* that is essentially a functional combination of atomic metric values. For instance, it does not make sense to trigger a scaling action whenever the CPU load exceeds 80%, but it can make sense to scale if the average CPU load has been more than 80% the last hour.

The metric values can be freely used in CAMEL, including in the scalability rules, the constraints, and in the definition of the application *utility*. Hence, a change in a metric that is used in the constraints or the utility definition actually changes the constraint programming problem solved to find the optimal *configuration* and therefore a different configuration can be better for the current context defined by the metric values. As an example, an application with a good load balancer may have a constraint saying that the global average number of application users per 15 minutes divided by the number of instances of the web server should be less than 100. The number of instances will here be the *variable* the solver can change, and this constraint may or may not be violated based on the measured number of users, i.e. the *composite metric*.

The consequence of this is that the constraint programming problem should be solved again whenever one of the metric values involved in the constraints or the utility definition changes. The *new configuration* found will then be passed on to the adapter, which will make the decision whether to reconfigure the running application, or leave it as it is because the cost of changing is too high compared with the added benefit brought by the *new configuration*.

The result of the mechanisms offered by PaaSage is that PaaSage becomes an automatic DevOps robot that constantly monitors a running Cloud application and try to adapt it to the best possible deployment configuration given the application's current execution context. Figure 13 gives an overview of the components of the PaaSage open source project, which will be reflected in the architecture of Melodic.

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

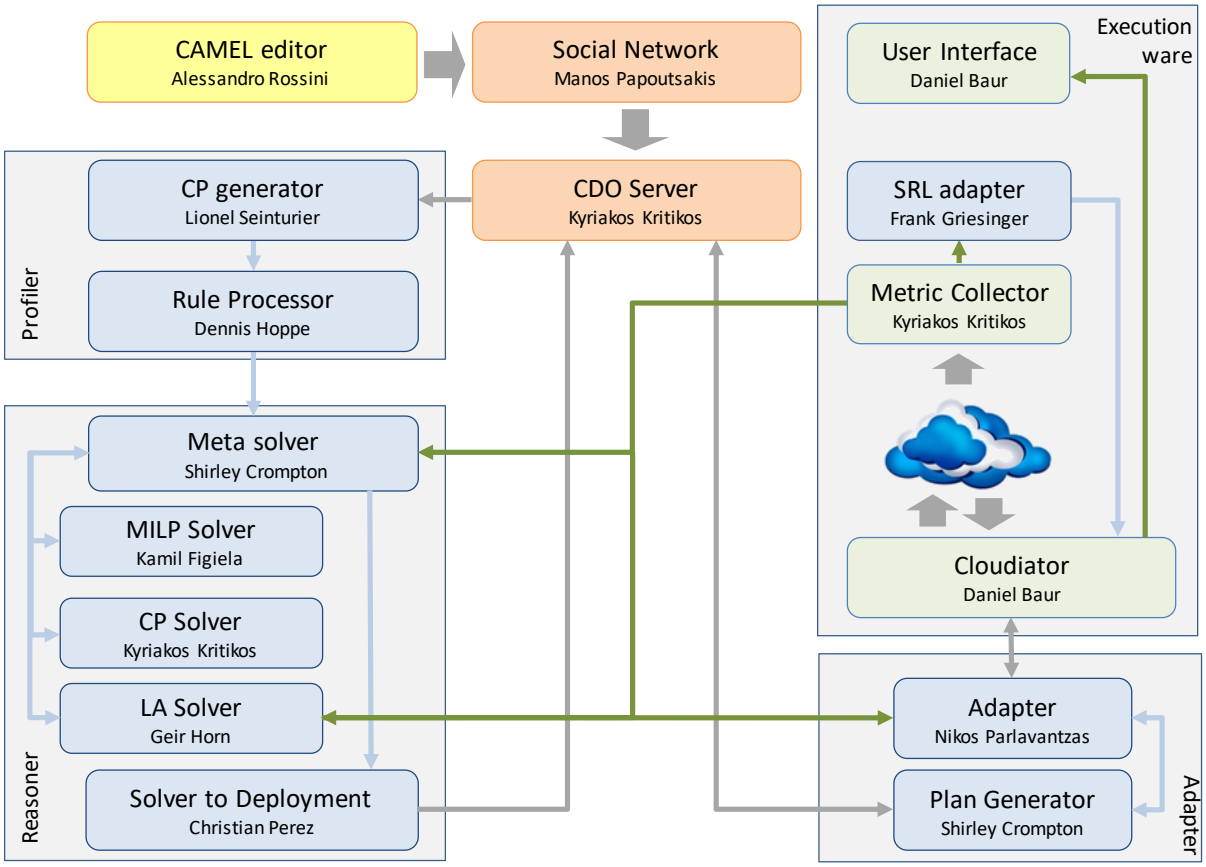

Figure 13: The PaaSage open source components and component owners (From PaaSage D8.2.2)

Software Component Assessment for Melodic

In Table 8, we summarise our assessment of the software components of the PaaSage project and enlist required extensions and modification for their reuse in the Melodic project. The assessment is done on the basis of both the requirements of the Melodic platform (mainly related to the support of data intensive applications) as well as any software refactoring needed to ensure modular and extendible Melodic architecture.

Readers are referred to PaaSage deliverables D3.1.2, D4.1.2, D5.1.2, and D8.2.2 for further details about the PaaSage components.

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

Table 8: PaaSage software component assessment for Melodic

Component	Required Extensions and Modifications	Remarks
CP Generator	Merge with Rule Processor component	CP Generator will be used to create a constraint problem for optimisation by solvers relying on a user-supplied application configuration model (CAMEL), and the available Cloud service offerings. It will be merged with the Rule Processor to create one integrated functional component responsible for preparing the suitable input for all the solvers.
CAMEL	Extensions are mainly needed in CAMEL meta-model to include data modelling features currently missing (see Section 6.4). Some modifications are also needed to increase CAMEL reusability.	The main meta-model extensions will be reflected in the domain code automatically produced as well as on the respective code of the textual editor (syntax update mapping to code update afterwards)
Metasolver	Add support for the big data frameworks and applications	Metasolver will be used to control the generation of (re-) deployment solutions as in PaaSage, but will be extended to initiate re-deployments of big data applications too, based on suitable metrics at the application and infrastructure levels.
MILP solver	Reusable without any change	There is no need to extend MILP solver to cover the encoding of all possible deployment solutions for data-intensive application as those should be independent from the CP model generation.
CP Solver	Reusable without any change	As above.
LA Solver	Adjust with respect to the data-awareness and optimisation	LA Solver needs to be adjusted to add support for optimisation of big data applications and frameworks.

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

Component	Required Extensions and Modifications	Remarks
Solver to deployment	Add support for big data application deployments	Solve to deployment logic will be extended with the capability to prepare a deployment plan for both the big data processing frameworks and the big data applications.
Metric collector	Add support for data-intensive applications	The PaaSage Metric collector needs to be extended with the ability to gather and publish metrics for the data-intensive applications, including framework level monitors.
SLR adapter	Add support for data processing frameworks. Consider possible integration with Adapter	SRL adapter needs to be extended with the capability to control the monitoring infrastructure also at the level of the data processing frameworks.
CDO Server	Changes according to change in CAMEL	See Section 6.4 for details.
CDO Client	Changes according to change in CAMEL	See Section 6.4 for details.
Plan Generator	Rewrite and integrate with Adapter	
Adapter	Rewrite and integrate with Plan Generator	Due to substantial changes related to the data-awareness, the PaaSage Adapter will be rewritten for Melodic, and will be integrated with the Plan generator.
Cloudiator	Extensions required for data awareness.	

6.2 CACTOS EU Project

CACTOS was an EC project that ran from 2013 until 2016. All of its software artefacts including documentation are currently available under http://cactos-cloud.eu. CACTOS had as a primary

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

goal to help the Cloud provider organise and orchestrate his data centre, to optimise the utilisation, and to provide a better service to his customers. Finally, it should also help planning data centre extensions. A particular focus during the project was put on supporting operators of data centres with heterogeneous hardware. This was deemed necessary in order to reflect the variety of technological options to build servers from e.g. different CPU architectures as well as specialised hardware such as GPGPUs.

In brief the results of CACTOS help to cope with the challenge of optimising the mapping of services (running in virtual machines) to a variety of different resources both hardware- and software-related topology-awareness is required. This mapping needs to consider placement of the services and demanded intelligent and cross-domain integration of actual and historical usage data. The key research outcomes addressed by CACTOS are as follows:

- Model for describing all aspects of the data centre and deployed software stack that are relevant to efficient resource management and application provisioning. This includes a sub-models for heterogeneous workloads, infrastructure-architecture models and topologies as well as facility management information and energy supplier information within a coherent information model.
- A scalable and open collection and analysis framework for historical usage data and how to derive from intelligent management strategies how to respond to different observed situations autonomously.
- Realisation of new infrastructure management methods that integrate different aspects into a unified multi-dimensional optimisation model. This includes among others dynamic workload placement, scheduling and migration by continuous optimisation across multiple partially orthogonal or correlated criteria (e.g. energy efficiency and costs).
- Development of a simulation framework for conducting costs and risk analysis to validate the intelligent Context-Aware Cloud Topology Optimisation Strategies for robustness on a large scale beyond the limits of prototypical installations and deployments.

CACTOS was designed to have a clear focus and aligns all its activities on the specific problem of optimisation within the Cloud operators' infrastructures.

Software-wise, CACTOS consists of the three tools CactoOpt, CactoScale, and CactoSim that had been assembled in two toolkits, the Runtime Toolkit consisting of CactoScale and CactoOpt for monitoring, analytics, and optimisation of data centres, and the Prediction Toolkit consisting of CactoSim and CactoOpt for simulating future utilisation and long term trends. Both Prediction and Runtime Toolkit are enhanced with integration components that link CACTOS to the (physical or simulated) data centre orchestration and monitoring layer. The architecture is shown in Figure 14. Besides the tools and toolkits, it details the infrastructure models that represent the current state of the physical infrastructure as well as the virtual machines running on it.

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

The Optimisation Plan model is a further model that represents a list of optimisation steps that have to be executed in order to improve the overall state of the data centre. These actions can include the migration of a virtual machine to another host, the start and deletion of a virtual machine on behalf of a user, switching on/off a physical server, scaling out/in an application, and the definition of prioritisation of two virtual machines running on the same host.

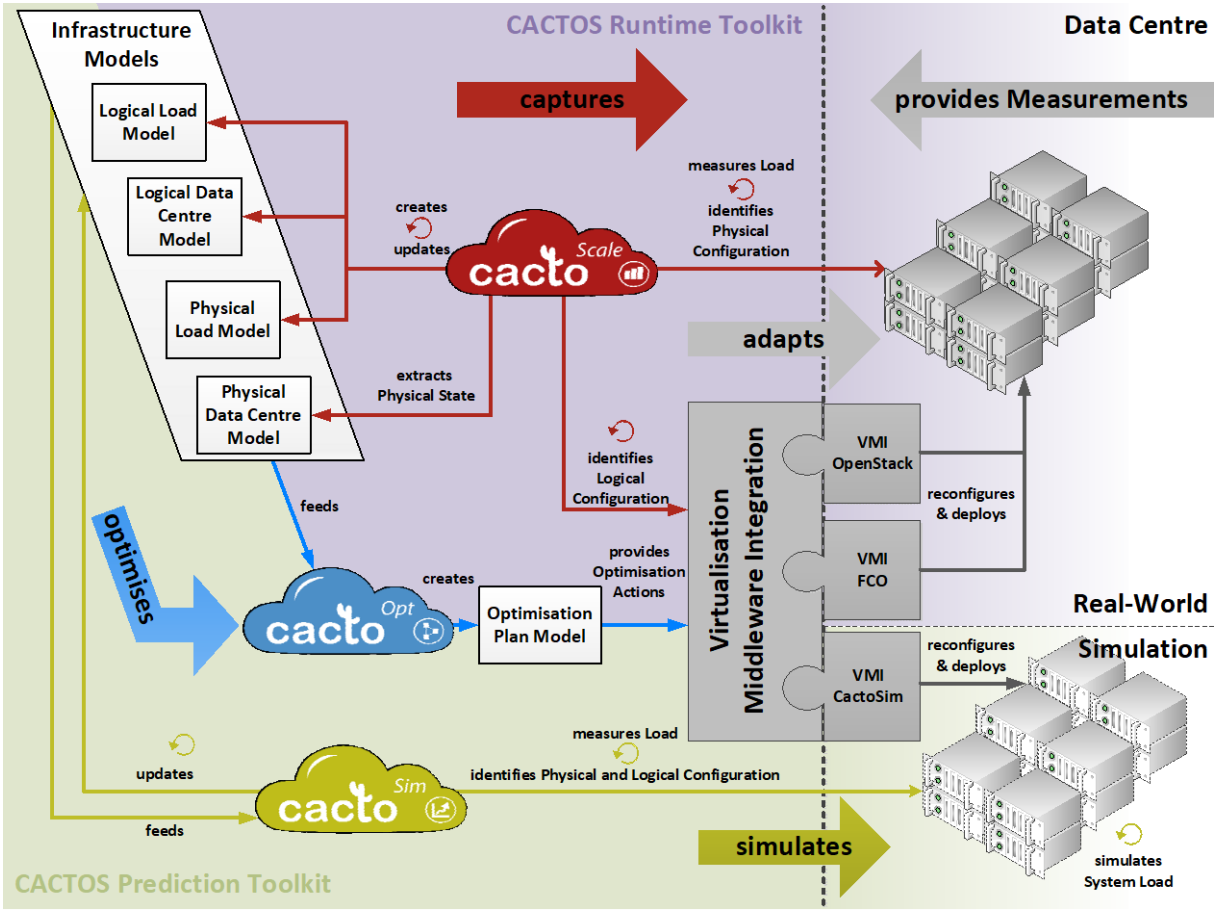

Figure 14: The CACTOS Architecture

The Virtual Middleware Integration component is crucial, as it is the component that interprets the actions of the optimisation plan and translates them to invocations to the Cloud middleware orchestrating the system. As shown in the figure, CACTOS comes with out of the box support for OpenStack and Flexiant FCO middleware.

What is not directly shown in the figure is that CACTOS not only provides support for individual virtual machines, but also for distributed applications consisting of more than one virtual machine. Here, CACTOS is capable of deciding when to scale out or in an application taking into

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

account the current workload on the application as well as the overall state of the data centre. In order to support this feature, CACTOS, in contrast to standard IaaS clouds, requires a notion of applications and a mechanism to deploy applications in on step. For that purpose, CACTOS adopted the Cloudiator toolkit from PaaSage, i.e. the Execution ware, and enhanced it with an application catalogue. This catalogue that not only includes the required descriptions of applications and their components, but also additional meta-data required by the CactoOpt optimisation engine.

Software Component Assessment for Melodic

Software component assessment for the CACTOS project with respect to the reuse in Melodic is provided in Table 9. Please refer to the CACTOS deliverables for more details about the software components and algorithms developed in the project.

Table 9: CACTOS Software assessment for Melodic

Component	Required Extensions and Modifications	Remarks
Enhanced Cloudiator Suite comprising Colloseum, Lance, and Sword	Add support for data-awareness and data-intensive application deployments	Cloudiator has proven its usability and flexibility in several research projects. Nevertheless, Cloudiator will need extensions in order to be able to cope with the intended data-awareness and support for deploying data-intensive applications targeted by Melodic.
Auto-scaling Algorithms	Consider at a later stage	The CACTOS auto-scaling have proven superior over user-provided auto-scaling algorithms as their control theoretical foundation avoids typical unintended and undesired patterns such as thrashing. On the downside, they are applicable only in a very narrow usage scenario (request/response based interactions). Also, the code base does not sufficiently hardened for wide-spread use.

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

Component	Required Extensions and Modifications	Remarks
Application Catalogue	Add support for data intensive technologies, and multi- and Cross-Cloud deployments	The application catalogue is stable and ready to be used. For being applicable to Melodic, it will have to be fed with data on the intended baseline technology. Further, it is necessary to enhance it with multi- and Cross-Cloud support, functionality which was not required by CACTOS.

6.3 PaaSword EU Project

PaaSword is a H2020 project that provides a holistic data privacy and security by design framework for Cloud applications. This security-by-design framework is provided as a platform-as-a-service (PaaS) solution, including several capabilities for assisting developers to define appropriate access control policies that can safeguard their sensitive data artefacts. The project offers technological solutions that aspire to address the semi-honest adversarial model [9] whereby a malicious Cloud provider that may host critical parts of a Cloud application, is assumed to properly follow the specification of a security protocol while at the same time, intercepts messages in order to acquire sensitive data [9]. Based on this objective, PaaSword offers, as a service, the following security-related features:

- Code annotation capabilities at the level of data access objects (DAOs) that allow developers to articulate access control policies that are required for protecting sensitive data in the Cloud
- Interpretation of the code level annotations into policy enforcement rules
- Governance and quality control during the policies formulation
- Implementation of a dedicated policy enforcement business logic that can control the access to sensitive data
- Transparent key management for encrypting sensitive data
- Fragmentation and distribution of data artefacts across remote databases for enforcing privacy

In Figure 15, the main aspects of the PaaSword conceptual architecture are presented. PaaSword considers applications that adopt and respect the Model-View-Controller (MVC) development pattern [10]. The main aspects of this approach include (a detailed description is provided in [11]):

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

I. the PaaSword integrated development environment (IDE) for performing security related code level annotations;
II. the Policy Enforcement Mechanisms that implement PaaSword's access control business logic and
III. the Physical Distribution, Encryption and Query Middleware Mechanisms that involve a pilot-specific key management mechanism for encrypting/decrypting sensitive data and a trusted database proxy for handling queries on the fragmented and physically distributed data.

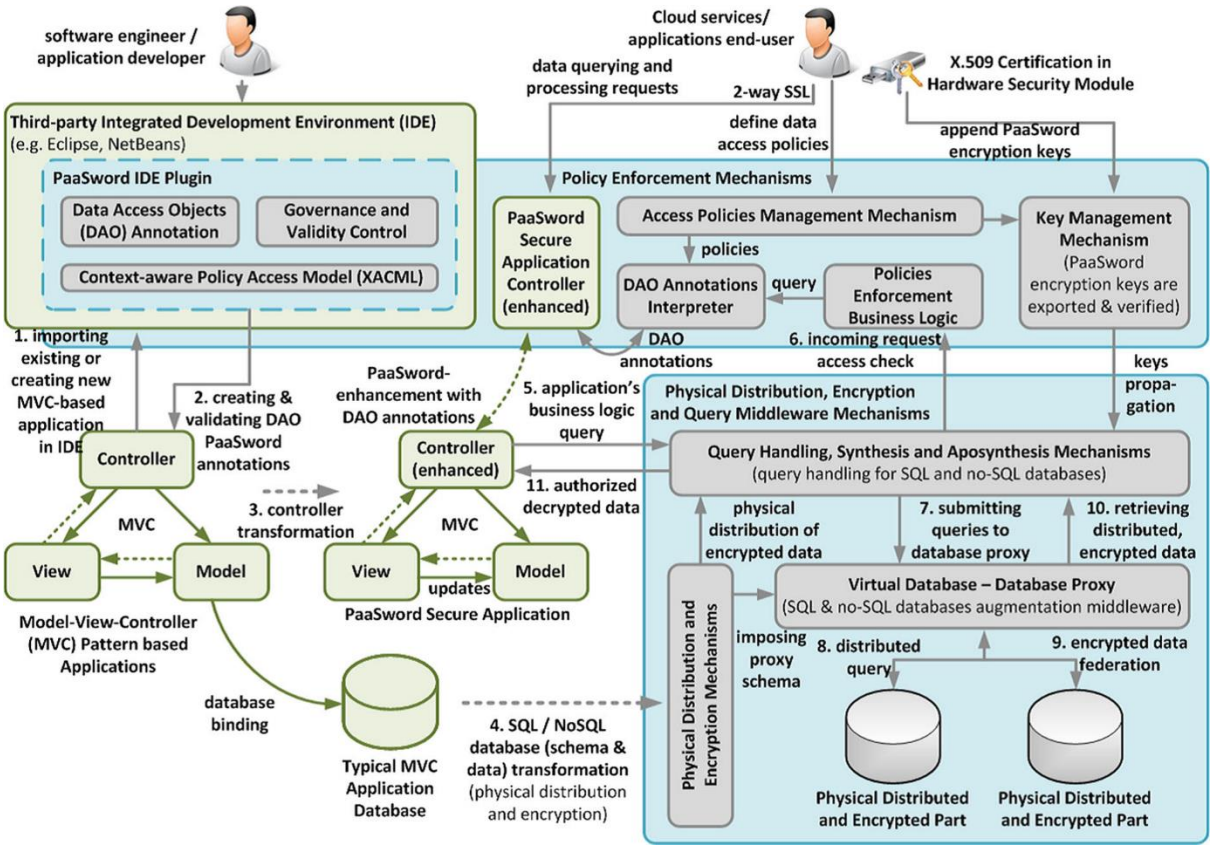

Figure 15: PaaSword framework conceptual architecture [11]

The most relevant to the Melodic needs, out of the PaaSword's platform-as-a-service offerings, is the sophisticated context-aware access control mechanism. This mechanism was based on a context-aware security model, which serves as the background knowledge of a fine-grained access control scheme, one that allows the per-user management of access rights. The notable achievement here is that this per-user management of permissions is done even if the user is not known in advance (i.e., at design-time), by using several contextual information that characterise each unique access request. Thus, the access permission to any incoming request is performed in real-time based on the current situation observed by the system. This is achieved by introducing

a XACML-based[29] context-aware access model, which can be used by the developers in order to create access policies on data handled by their applications.

The OASIS eXtensible Access Control Markup Language (XACML) constitutes a well-known implementation of the attributed-based access control paradigm (ABAC) with a worldwide industrial adoption by banks, insurance companies and health-care providers among others. ABAC [12] is a logical access control methodology where the authorisation to perform a set of operations, over sensitive data, is determined by evaluating a number of attributes related to the subject, object, requested operations, and the environment conditions against certain policy sets, policies and rules that describe the desired permissions. The advantage of XACML is that it can be used for describing declarative access control policies encouraging the separation of the access decision from the point of use.

PaaSword introduced a Context-aware Security Model for semantically describing such attributes that can be used in XACML-based access control policies. The metamodel and the ontological model for ABAC policies can be found in and , respectively. The main aspects of these models involve several classes for semantically capturing:

- The characteristics of incoming *Requests*
- The entity (i.e., *Subject*) seeking access to a particular object or the entity whose state should be considered for allowing a certain requestor to access sensitive data
- The details of the protected resources (i.e., *Object*)
- The *SecurityContextElement* which describes the various contextual attributes that may be associated with the subjects and/or the objects of a request, as well as with the request itself
- The allowed actions (i.e., *Permission*) that an individual of the class *Subject* is able to perform upon an individual of the class *Object*
- The recurring motives of data access (i.e., *ContextPattern*) in order for the future access requests to be permitted, or denied on the basis of historical data that might reveal malicious behaviour
- The characteristics of dedicated software components that are used for federating and processing raw data relevant to an access control decision and semantically uplifting them as instances of the Context Model (i.e., *Handler*).

[29] OASIS eXtensible Access Control Markup Language (XACML). https://www.oasis-open.org/

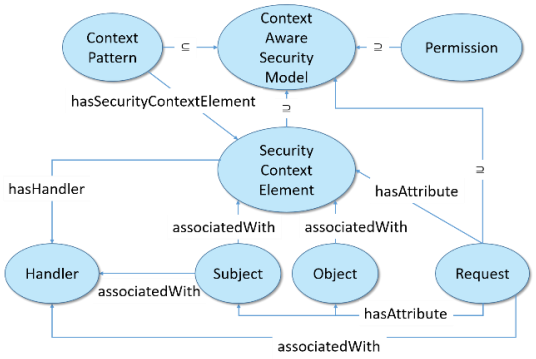

Figure 16 Context-aware security meta-model [13]

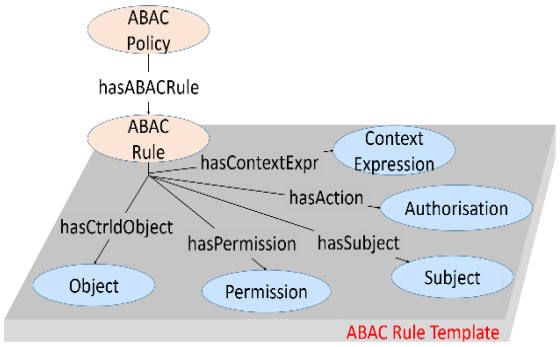

Figure 17 Ontological model for ABAC policies **[13]**

Software Component Assessment for Melodic

According to Melodic's description of work, it was stated that we intend to build on top of PaaSword's context-aware access control mechanism, extending it accordingly in order to be valuable for dynamic Multi-Cloud application deployments, where access to information may vary according to the workload and the user defined security requirements. As PaaSword consortium (ICCS and CAS are members of it) recently decided on licensing issues, it was announced that PaaSword platform will not be released under an open-source license. Therefore, PaaSword context-aware access control mechanism cannot be used and extended in Melodic, with the only exception being that of the PaaSword Security Context Model that ICCS solely developed and has already been released under Apache v2.0. Based on this, Melodic will re-use and extend the PaaSword Security Context Model by providing the appropriate additions of concepts and associations that are relevant to Multi-Cloud environments, while it will also implement a PaaSword inspired solution with respect to the access control enforcement mechanism. For this mechanism, open-source technologies like the Balana XACML engine[30] will be examined.

[30] http://xacmlinfo.org/category/balana/

6.4 CAMEL – Application Modelling

In order to support the complete modelling of the user Cloud applications according to all the aspects needed plus the models@runtime approach [14], the PaaSage project has created a super-DSL called CAMEL. In this section, after introducing CAMEL, we shortly analyse this super-DSL and finally we provide an overview of those CAMEL extensions that are needed to properly support the data-awareness aspect needed for the Melodic middleware.

CAMEL is a super-DSL which includes multiple DSLs, each focusing on a particular aspect. CAMEL has been designed based on EMF Ecore[39] and OCL[40]. EMF Ecore enables the specification of UML-based meta-models, while OCL constraints accompany such meta-model specification with the coverage of additional domain semantics. CAMEL has been derived from pre-existing languages, before the PaaSage project was started, including CloudML [24] for the coverage of the deployment aspect, Saloon feature meta-model [25] for the coverage of provider modelling, and CERIF [26] for the organisation aspect coverage. Other sub-DSLs, like the Scalability Rule Language (SRL) [27], were developed during the PaaSage project to cover some missing aspects. All these sub-DSLs where integrated by moving them into the same technical space but also consolidating them to diminish their respective conceptual overlaps. Integration was also supported through the use of OCL rules focusing on cross-model validity (where cross-model here means between models from different sub-DSLs but still within the same overall CAMEL model).

Via the use of the Eclipse Environment, CAMEL is assorted with nice tools which ease the life of both developers and modellers. Developers are assisted through the production of a programmatic API (domain model) via which models conforming to CAMEL can be created and processed. Modellers are assisted via the offering of different editing tools. Such tools are either supplied by default by Eclipse, like the Eclipse tree-based editor, or have been created within the PaaSage project. The latter tools include: (a) the CAMEL textual editor which supports the concrete syntax of CAMEL (realised via the Eclipse Xtext[41] technology) and (b) the web-based editor which was developed based on the Eclipse RAP[42] technology.

A small analysis over all these editors can be seen in Table 10. The criteria of evaluation include the following: (a) *model validation*: if validation of models according to Ecore semantics and OCL constraints is supported; (b) *aspect*: which modelling aspects are covered by the editor; (c) *repository integration*: if the editor is able to support the live editing of models over a model repository; (d) *access control*: if the editor is able to enable a controlled access over the model repository contents; (e) *formats*: what are the CAMEL encodings supported (XMI & textual); (f) *roles*: which kinds of users can use the editor.

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

Table 10: Comparison table between the different editors available for CAMEL

Editor	Model Validation	Aspects	Repository Integration	Access Control	Format	Roles
Tree-based	✓	All	✓	~	XMI	any
Textual	✓	All	-	-	Both	DevOps
Web-based	✓	Requirements, Organisation	✓	✓	Both	any

As it can be seen from the above table, the editor which satisfies almost all criteria is the web-based editor but this editor does not cover all aspects in CAMEL. In fact, this editor supports only the modelling of requirements, metric and organisation models.

The second best editor is the tree-based one which supports only the XMI encoding of CAMEL and provides partial support to access control over the model repository.

The textual editor is more suitable for DevOps and not business users, based on their current practice of work. In addition, this editor can be only used in an offline mode, thus it is not rationale that it exhibits any kind of access control feature.

Current Status

Currently, CAMEL comprises many sub-DSLs which cover different modelling aspects, spanning both the design and runtime of the Cloud application. Table 11 provides an overview of all CAMEL DSLs shortly indicating their scope, the role involved in their modelling (user roles, like *DevOps*, *admin* and *business*, and *system* representing the platform) and whether they cover the design or runtime aspect. More details about CAMEL sub-details can be found in CAMEL documentation[31].

[31] https://gitlab.ow2.org/paasage/camel/raw/master/documents/CAMELDocumentation.pdf

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

Table 11: Overview of CAMEL

Name	Coverage	Role	Design / Runtime
Camel	Top model, container of other models, application	User[32], System	Both
Deployment	Application topology	DevOps, System	Both
Requirement	Hardware, security, location, OS, provider, QoS requirements	DevOps, Business	Design
Metric	Metric, Sensors, Attributes, Conditions	DevOps, System	Both
Scalability	Scalability rules, (composite) events, scaling actions	DevOps	Design
Security	Security controls, attributes and metrics	DevOps	Design
Location	Physical and Cloud-specific locations	DevOps	Design
Unit	Units of measurement	DevOps	Design
Type	Value types and values	DevOps	Design
Organisation	Organisations, users, roles, policies, Cloud/platform credentials	Admin	Pre-Design[33]
Execution	Execution contexts, measurements, SLO assessments, adaptation history	System	Runtime
Provider	Provider offerings	Admin	Both[34]

[32] *user* denotes any kind of user
[33] Pre-design has been entered as a value as the modelling of an organisation should be specified only once, even before any application is modelled in CAMEL. Thus, organisations should specify one organisation model and multiple, independent CAMEL models mapping to their use cases/applications.
[34] The value of 'Both' has been entered in order to highlight that in principle a provider model can be modified at both design and runtime by the system administrator. As runtime, the modification should be performed online in the system, as otherwise the platform instance needs to be stopped to apply the modifications and then restarted. The platform itself, if extended accordingly, might also apply modifications to the provider models online, as soon as it detects them.

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

Required Extensions for Melodic

In the context of the Melodic project, the coverage of the data aspect, which is not captured by CAMEL, is prominent. While the goal of this deliverable is not to provide an extension over CAMEL for this aspect, we just outline here some generic requirements that this extension should satisfy. A more detailed analysis over this extension is the subject of Melodic D2.4 deliverable.

The first extension that needs to be performed over CAMEL concerns the modelling of the data itself. This modelling could unveil details about particular features of the data, such as its format, size and number of objects/elements. It could also unveil details about how to access this data and where it is currently stored. All this information should enable the platform to be aware not only of the data size (or any other characteristic) but also how to access it and move it on demand (as part of the application processing).

The second extension concerns the fact that CAMEL should be independent of any data processing framework and specify its own tool-independent way of modelling data flows. The platform could then create specific adapters and transform the respective data flow to the format acceptable by a certain data processing framework. The data flow could be inspired by data flow languages already developed in the context of scientific computing. It could also be inspired by the language features of existing data processing frameworks. The modelling of the data flow itself can be considered as a linked data-based (I/O) connection between application components or data processing operators which defines then the flow of the respective information. On top of this flow, respective parallel programming paradigms could be enforced. This could depend on the way the flow is specified and by considering possibly specific user-provided parameters. Considering that operators or application components must be specified, we foresee an extensive enumeration of such operators in different categories, possibly depending on the type of processing that they can perform or the type of artefact that they represent (e.g., software component, web service). OCL constraints could be also coupled with these operators to restrict the number of input and output parameters that they can accept or the types of these parameters. This would enable a more sophisticated or semantic validation of the data flows specified.

To summarise, the design of the new CAMEL extensions relies on a set of requirements drawn from the project use cases. Then, a respective literature review over existing languages will be performed to determine suitable conceptualisations and adequacy of the respective languages with respect to satisfying these requirements. Finally, the findings from this literature review will be used as a guide for designing and developing the CAMEL extensions. We must stress here the role of standards which the project also is willing to promote. To this end, it can also be possible that an existing standard language is selected, extended, and possibly transformed to the same technical space as in CAMEL in order to be integrated.

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

7 Resource Management Systems

Effective management of available resources is an important challenge for any distributed system. In general, any physical or virtual component of limited availability within a computer system is a *resource*. In the context of Cloud computing, resources are broadly classified into two types: *infrastructure resources* and *platform resources*. Infrastructure resources are the actual data centre resources such as CPUs, storage, memory or network connections. Platform resources refer to the higher level abstraction of the infrastructure resources offering middleware, runtimes, databases, and other high-level services to the users. For the Melodic platform, resources are dynamic, acquired at runtime from clouds according to the requirements of the user application. Efficient management of the acquired Cloud resources can both improve the performance of the application, and reduce the costs for the Cloud users. In this chapter, we provide justification of the need of a resource management layer in Melodic, and present different generic requirements needed from the resource manager. In addition, we provide evaluation of two selected resource management systems, YARN and Mesos, based on the collected requirements.

7.1 Why a resource management layer is needed?

Melodic aims to transparently deploy and run a data-intensive application on segregated resources from different Cloud infrastructures including both private and public clouds. The execution of the data-intensive application typically comprises running application components of different types, such as traditional load-balancers, application servers, databases, together with so-called *big data processing frameworks* (Discussed in *Section 8*). The big data processing frameworks, such as Hadoop MapReduce and Spark, enables efficient parallel processing of large data sets in a highly scalable manner. Typically, the data processing frameworks are built upon a distributed architecture with multiple worker nodes, and perform data processing using a *job* based approach, where each job may have varying demands towards resources, runtime, and data locality. Generally, such frameworks focus only on the execution of the jobs based on a static pool of available worker nodes, and management of the available resources is considered a separate task. Having resource management as part of the big data processing limits scalability[35] as all resource management, job and task scheduling, and monitoring is to be done by the same framework.

[35] In old Hadoop MapReduce version 1.0, *JobTracker* was responsible for both the reource management and the execution of the data processing jobs, limiting its scalability [15].

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

As part of the collected use case requirements, it is established that running multiple applications developed using different big data processing frameworks (or customised data processing applications and toolchains) on the same infrastructure is a desired feature for the Melodic partners (See *Section 3.5, and Appendix A – MEL-12*). Creating infrastructure silos to accommodate individual user applications is inefficient and may result in low resource utilisation, and thus, high costs. Fine-grained sharing of resources among multiple big data processing frameworks (and applications) make it possible to use and share existing resources more efficiently among the user applications. This is, in particular, beneficial for the user applications that share data sources, another desired feature from the gathered use case requirements (See *Appendix A - MEL-7*). Moreover, having a separate resource management system allows for supporting legacy applications in the same cluster using customised scheduling policies, without changing data processing frameworks themselves to accommodate for a large number of possible customised configurations. In Figure 18, we present a decoupled resource management and data processing architecture depicting the approach taken by the Melodic.

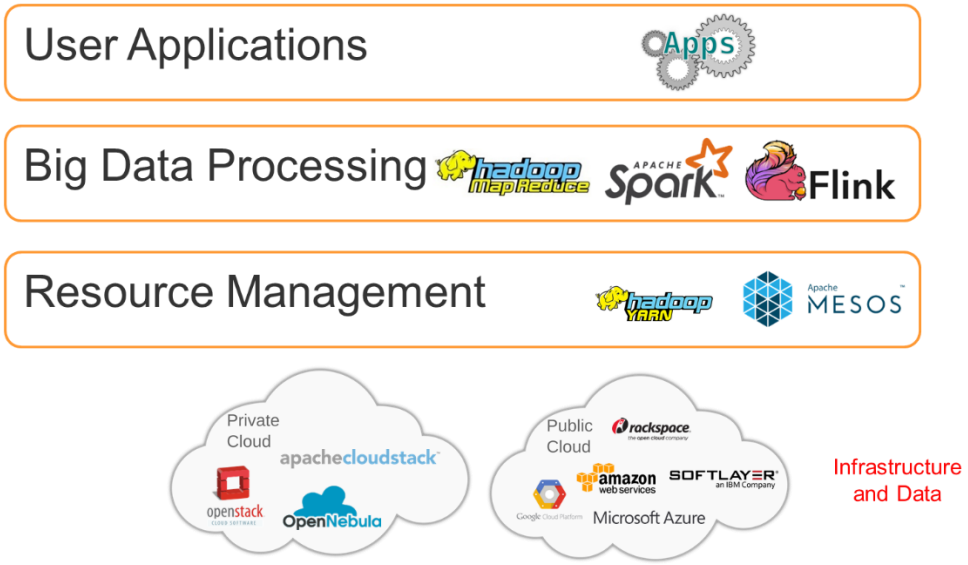

Figure 18: De-coupled resource management and data processing

7.2 Generic Requirements

In the following, we present generic requirements towards optimised resource management, gathered from careful consideration of use case requirements and generalised stakeholder and Multi-Cloud application analysis, presented in Chapter 2, 3, and 4.

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

Resource Abstraction: the data processing jobs typically impose basic requirements towards the resources, such as # of CPU cores, memory or disk. Hence, an abstraction layer between the data processing framework and available pool of resources is required, abstracting the access to the resources (e.g., physical or virtual) by providing resource units to the processing frameworks.

Scalable Architecture: As the pool of managed resources might grow when new resources are added to scale the processing of the data-intensive jobs, the resource management system itself needs to provide a scalable architecture to be able to manage arbitrary amounts of resource nodes.

Scheduling: The smart allocation of resources to data processing jobs is crucial for the performance and the resource utilisation. Especially as the number of data-intensive jobs to execute will vary over time and can exceed the capacity of available resources. Hence, the scheduling of jobs has to exploit sophisticated scheduling algorithms and provide for extensibility.

Data-awareness: Data-intensive jobs might have dependencies between each other and to external data sources. Hence, the resource management should be aware of data dependencies in order to include such factors into the scheduling process.

Technology Support: While Big Data processing frameworks are similar in terms of job submission, they differ with respect to the architecture and technology. Hence, a resource management system requires the generic integration of Big Data frameworks in order to support a variety of frameworks. In addition, data-intensive applications comprise additional application types besides Big Data processing frameworks. Therefore, resource management framework should be able support as well additional application components, such as databases or application servers to provide a holistic resource management for data-intensive applications.

High-Availability: By managing a pool of distributed resources, failures with respect to physical or virtual nodes as well as network partitioning might occur. Hence, the resource management system needs to be aware of such possibilities and provide appropriate mechanisms to deal with them.

Security: In order to restrict the access to the resources and respectively the executing jobs, the resource management system needs to support fine-grained authorisation and authentication mechanisms while the support for encrypted data transfer will be highly beneficial.

In the next sections the Hadoop YARN and Apache Mesos will be introduced and analysed according to the aforementioned requirements, describing their architecture, scheduling mechanisms, supported technologies, security features, and monitoring capabilities.

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

7.3 YARN

Hadoop YARN (Yet Another Resource Manager) evolved as a standalone component out of the first version of the Hadoop MapReduce framework[36]. The initial design of Apache Hadoop was tightly focused on running massively parallel data processing jobs. Due to the broad adoption of Hadoop, it has become a framework where data and computational resources are shared and accessed, stretching the initial design well beyond its originally intended. With Hadoop YARN, the two key shortcomings of the initial MapReduce framework are addressed: 1) tight coupling of a specific programming model with the resource management infrastructure, forcing developers to abuse the MapReduce programming model, and 2) centralised handling of jobs' control flow, which resulted in endless scalability concerns for the scheduler.

Architecture

The YARN architecture splits the resource management and job scheduling/monitoring into separate services as depicted in Figure 19. The resource management is handled by the *Resource Manager (RM)* and multiple *Node Managers (NM)*. The job scheduling and monitoring is handled by the Application Manager (*AM*) and *Application Masters (AppMstr).* Based on this architecture with the dedicated resource manager, a significantly increased scalability is achieved compared to the initial approach of the monolithic Hadoop MapReduce framework [15].

The *RM* and the *NM* form the data-computation framework, where the RM arbitrates resources among all the applications in the system. The *NM* is the per-node framework agent that is responsible for containers, monitoring their resource usage (CPU, memory, disk, network) and reporting the same to the *RM*.

The *AM*, which resides besides the *RM*, is responsible for accepting job-submissions, negotiating the resources for executing the initial *AppMstr*. The per-application *AppMstr* has the responsibility of negotiating appropriate resource containers from the Scheduler, tracking their status and monitoring for progress.

[36] http://hadoop.apache.org/

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

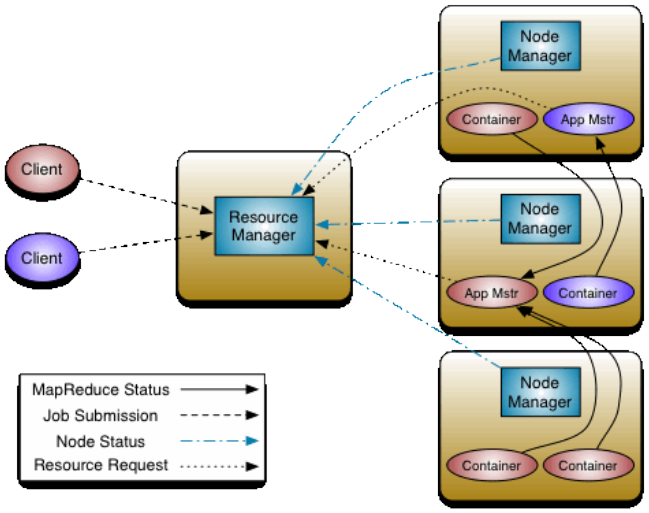

Figure 19: Hadoop YARN architecture

Resource Scheduling

The Scheduler has a pluggable policy which is responsible for partitioning the cluster resources among the *AppMstr*. Currently two schedulers are supported by YARN, the *CapacityScheduler*, useful in a cluster shared by more than one organisation, and the *FairScheduler*, which ensures all applications, on average, get an equal number of resources. Both schedulers assign jobs to the queues, and each queue gets resources that are shared equally between them. Within a queue, resources are shared between the applications.

YARN also supports resource reservations via the *ReservationSystem*, a component that allows users to specify a profile of resources over-time and temporal constraints (e.g., deadlines), and reserve resources to ensure the predictable execution of important jobs. The *ReservationSystem* tracks resources over-time, performs admission control for reservations, and dynamically instructs the underlying scheduler to ensure that the reservation is fulfilled.

Supported Technologies

As YARN's primary focus relies on the resource management for the Hadoop MapReduce framework, additional technologies can also be run on top of YARN, e.g., the Big Data frameworks

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

Apache Flink, Apache Storm[37], Apache Spark or the database Apache HBase[38] (based on the HOYA project[39]).

YARN provides extensibility to integrate custom applications but extensions require significant effort on the part of application implementers. Integrating a custom application on YARN includes building an own AM, performing client and container management, and handling aspects of fault tolerance, execution flow, coordination, as well as of other concerns.

Security

YARN provides authentication, service level authorisation, authentication for Web consoles and data confidentiality. The authentication uses Kerberos to verify that each user and service is authenticated by Kerberos. Service level authorisation ensures that clients using Hadoop services are authorised to use them. Access to the Hadoop services can be finely controlled via access control lists. Additionally, data and communication between clients and services can be encrypted using SSL and data transferred between the Web console and clients with HTTPS.

Monitoring

Hadoop YARN provides REST API and Web UI for the *RM* and the *NM*. The *RM* UI provides metrics on a cluster level while the *NM* provides information for each node and the applications and containers running on the node.

High Availability

YARN supports manual recovery using a command line utility and automatic recovery via *ActiveStandbyElector* based on Apache ZooKeeper[40], embedded in the RM. Hence, there is no need to run a separate ZooKeeper failover controller. ZooKeeper is only used to record the state of the *RMs*.

[37] http://storm.apache.org/
[38] https://hbase.apache.org/
[39] https://github.com/hortonworks/hoya
[40] https://zookeeper.apache.org/

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

7.4 Mesos

Apache Mesos represents a platform for sharing commodity clusters between multiple diverse cluster computing frameworks, such as Hadoop and applications based on the Message Passing Interface (MPI). Sharing improves cluster utilisation and avoids per-framework data replication. Apache Mesos shares resources in a fine-grained manner, allowing frameworks to achieve data locality by taking turns in reading data stored on each machine [16].

Architecture

The high-level architecture of Apache Mesos is depicted in Figure 20. Mesos builds upon two main components types, the *Mesos master (MM)* and the *Mesos agents*. The master manages the agents running on each resource.

The *MM* enables fine-grained sharing of resources (CPU, RAM, etc.) across the Mesos framework by supplying an offer-based resource distribution to the agent nodes. The master decides how many resources to offer to each framework according to a given organisational policy, such as fair sharing or strict priority. To support a diverse set of policies, the master employs a modular architecture that makes it easy to add new allocation modules via a plugin mechanism. Based on this architecture, Apache Mesos is able to scale up to 1000 nodes [16].

An application running on top of Mesos consists of two components: a scheduler that registers with the master to be offered resources, and an executor process that is launched on the agent nodes to run the local part of each framework. While the master determines how many resources are offered to each application, a framework's scheduler selects which of the offered resources to use. When a framework accepts offered resources, it passes to Mesos a description of the tasks it wants to run on them. In turn, Mesos launches the tasks on the corresponding agents.

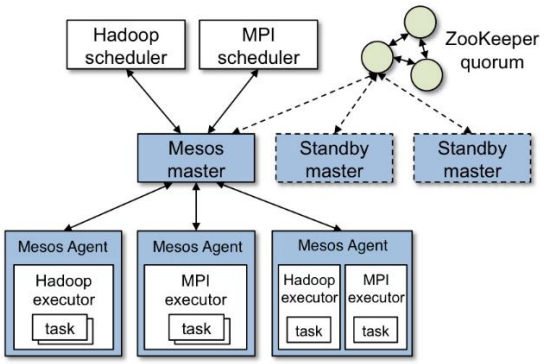

Figure 20: Apache Mesos architecture

Resource Scheduling

Mesos follows a two-level scheduling in an offer-based manner. In case an application scheduler, e.g., the Hadoop scheduler, wants to submit a new job, the Hadoop scheduler will receive an offer of available resources (e.g. free resources at node 1: 2 cores, 2GB ram, 10GB disk) by the master. Depending on the Hadoop scheduler policy the offer can be accepted (and the job will be executed on node 1) or reject the offer (and receives additional offers as soon as different resources are available). The two-level scheduling model of Mesos allows each application to decide which algorithms it wants to use for scheduling the jobs, e.g., by the Hadoop scheduler or MPI scheduler. Mesos acts as the arbiter, allocating resources across multiple schedulers, resolving conflicts, and making sure resources are fairly distributed based on business strategy.

Supported Technologies

From the beginning, Mesos was designed to support a wide range of technologies, comprising long running services, such as Aurora[41] or Marathon[42], Big Data Processing, such as Apache Spark, Hadoop MapReduce or Apache Storm, or Data Storage, such as Apache Cassandra[43] or MrRedis[44]. A complete list of the currently supported technologies is provided in the Mesos documentation[45]

The extension for custom technologies requires the implementation of a custom framework scheduler and custom framework executor.

Security

Mesos provides authentication for any entity interacting with the cluster. This includes the slaves registering with the master, jobs submitted to the cluster, and operators using endpoints, such as HTTP endpoints. Mesos' default authentication module, Cyrus SASL, can be replaced with a custom module. Access control lists are used to authorise access to services in Mesos. By default, communication between the modules in Mesos is unencrypted. SSL/TLS can be enabled to encrypt this communication. HTTPS is supported for the Mesos web-based user interface.

---

41 http://aurora.apache.org/
42 https://github.com/mesosphere/marathon
43 https://github.com/mesosphere/dcos-cassandra-service
44 https://github.com/mesos/mr-redis
45 http://mesos.apache.org/documentation/latest/frameworks/

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

Monitoring

Apache Mesos provides numerous metrics for the master and agent nodes, accessible via a REST interface. These metrics include, for example, percentage and number of allocated CPUs, total memory used, percentage of available memory used, total disk space, allocated disk space, elected master, uptime of a master, slave registrations, connected slaves, and so on.

High Availability

The Mesos cluster manager supports automatic recovery of the master using Apache ZooKeeper to enable recovery of the master. Jobs which are currently executed, continue to run in the case of failover.

7.5 Comparison and Integration in Melodic

As can be deduced from the above discussion, even though both YARN and Mesos provide efficient resource scheduling with rich security and availability features, the Mesos resource management system has several advantages over YARN. Mesos, by design, provide easy extensibility for supporting a wide range of big data technologies, including custom data processing frameworks. On the other hand, adding support for custom applications require substantial efforts in YARN. For the Melodic, supporting legacy and custom data processing applications is an important requirement affecting the usability of the project in a variety of use-case scenarios. In addition, to support the future-looking vision of Melodic, a resource management system is needed that can be easily extendible to incorporate upcoming superior technologies in the future. In addition, thanks to the two-level scheduling approach taken by the Mesos, it scales linearly and proven to work efficiently up to tens of thousands of nodes, comparably better than YARN. This has led the consortium to decide that Mesos will be integrated in Melodic to provide resource management layer for the data-intensive applications. However, resource management in Melodic is planned to be developed using abstract interfaces. In this way, replacing Mesos with a new resource management system will be straight-forward as long as the required interfaces and features are provided by the resource manager.

In Table 12, we provide results for our assessment of the Mesos resource management system along with the integration requirements in different Melodic sub-systems.

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

Table 12: Mesos integration requirements for Melodic

Sub-system	Integration Requirements	Remarks
Upper ware	Provide a new Mesos allocation module	Allocator modules in Mesos contain logic about what resources to be offered to which framework. Several built-in allocator modules are available, however, for the Melodic, the allocation logic needs to be *controlled* by the Melodic upper ware.
Execution ware	Add support for the automated management of Mesos, i.e. orchestrating Mesos masters and agents across Cloud resources. Enabling dynamic resizing of the resource pool on which Mesos is operating	Mesos needs a master node that manages resource management though Mesos agents that need to be executed on each cluster node.
Applications	Add support for custom data processing frameworks when needed by the user applications Add support for *dummy* frameworks for legacy applications	A framework in Mesos has two main components: scheduler and executor. The scheduler is registered with the Mesos master and is offered resources. The frameworks' schedulers select which of the offered resources to use for the data processing tasks. An Executor process is launched on each agent node for running framework tasks.

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

8 Data Processing Frameworks

A common way to handle and process huge amounts of potentially unstructured data, is to implement and run a data-intensive application by means of a *big data processing framework*, or just *big data framework* for short. Such a framework provides a unified, abstracted and distributed processing solution for the development of data-intensive applications. That is, the framework makes it possible for the developer to focus on the functionality and the data analytics part of the application at hand, and less so on general distributed data processing issues like cluster management, distributed storage and fault tolerance.

To facilitate efficient data-intensive processing by means of Melodic, it is clear that the Melodic middleware should support the most prominent big data processing frameworks. For this reason, as already mentioned in the description of work, Melodic will support both Apache Hadoop MapReduce and Apache Spark. Furthermore, new promising big data frameworks, like Apache Flink, should also be considered as they mature. However, the requirement of Melodic to support more than one big data framework calls for the introduction of a separate resource management layer utilised by the frameworks, as already detailed in Chapter 7. The resource management layer will then make it possible for Melodic to concurrently handle applications utilising different big data frameworks in an efficient manner. At the same time, a resource management layer makes Melodic very flexible when it comes to adding support for new big data frameworks as they appear and become more mature at a later stage – in addition to supporting legacy applications using customised data processing tool chains.

In the following sections we will briefly introduce the three big data frameworks Apache Hadoop MapReduce, Apache Spark, and Apache Flink, and discuss components being candidates for reuse in Melodic.

8.1 Hadoop MapReduce

Apache Hadoop MapReduce is an open source software framework implementing the MapReduce programming model to support scalable processing of large datasets. The MapReduce programming model was originally developed by Google to handle their increasing day-to-day demand of data-intensive processing, in particular, related to the web page indexing. After the success of MapReduce at Google, Apache Software Foundation and Yahoo created an open source version of the MapReduce, now known as Hadoop MapReduce.

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

Salient Features

The basic idea behind MapReduce programming paradigm is to provide a clean abstraction of the underlying cluster infrastructure to the programmer, saving details of the management of the actual cluster, node communications, task monitoring, and failure management to the framework itself. Typically, data in a Hadoop cluster is fragmented into small independent chunks (or blocks), which are assigned to the *compute/worker nodes* in the cluster, and are processed by the *map* tasks in a completely parallel manner. Output from the individual map tasks is then combined by the *reduce* tasks to produce the final output. The MapReduce framework operates on <key, value> pairs. Both the input to the jobs as well as the output is stored as <key,value> pairs of different types. The programmer supplies *map* and *reduce* functions based on the application needs while the framework handles task scheduling, monitoring, and re-execution of the failed tasks. The data, both the input and the output of the jobs, is usually stored in a distributed file system. The Hadoop Distributed File System (HDFS) is commonly used due to its tight integration with the Hadoop MapReduce framework. Practically, the MapReduce compute nodes and the HDFS storage nodes can be the same (running both MapReduce and HDFS) allowing MapReduce framework to take advantage of the data locality by scheduling corresponding data processing tasks on the nodes where the data already resides. The exploitation of data locality saves substantial data transfer costs, and improves the efficiency of the Hadoop cluster.

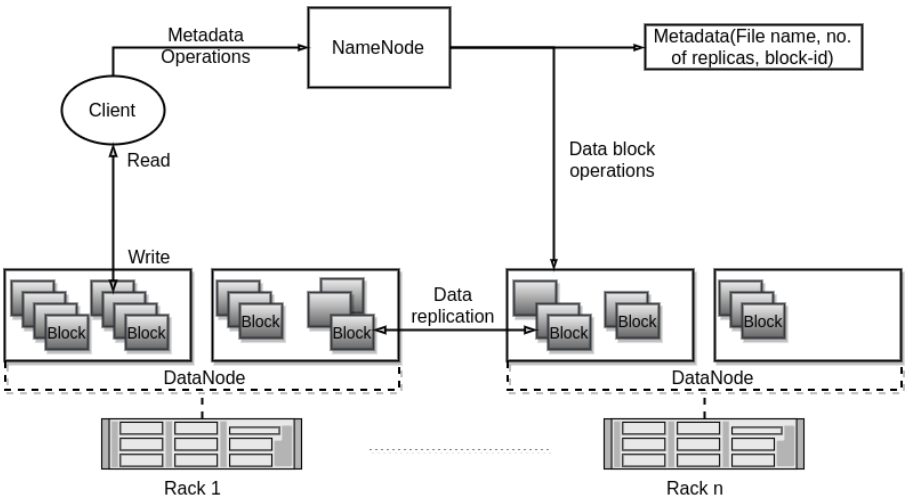

Figure 21: Canonical block diagram of HDFS

HDFS is based on single-master and multiple-slave architecture. As shown in Figure 21, the *NameNode* is the master node, and the *DataNodes* are the slave nodes. NameNode manages the file system namespace and controls the file access by clients. The user data is stored in files

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

which are internally split into fixed size blocks. These blocks are stored in a set of DataNodes, one per cluster node. The general function of a DataNode is block creation, deletion, and replication upon instruction from the NameNode as well as local storage management. The NameNode determines the mappings of blocks to DataNodes which are also responsible for serving read and write requests from the file system's clients. Notable features of HDFS include fault-tolerance, high throughput, and possibility of deployment on low-cost commodity hardware.
In initial MapReduce versions, a monolithic approach was taken in which resource management and job scheduling/monitoring was coupled together. This resulted in scalability issues on large clusters. As described in Section 7.3, starting from MapReduce 2.0, resource management and job scheduling has been split into separate services using YARN (*See* Figure 19). In YARN, the resource management is handled by the Resource Manager and Node Managers in coordination. Whereas, Application Manager and Applications Masters are responsible for the job scheduling and monitoring. Thanks to the decoupled architecture with a dedicated resource manager, a significant increase in scalability is achieved, compared to the initial approach. Hadoop MapReduce is also supported by the Mesos resource manager, taking a similar decoupled resource management approach as YARN, albeit with a more scalable and flexible solution.

Integration in Melodic

Hadoop MapReduce is a popular data processing framework holding extensive market share in data-intensive computing. Many Cloud providers offer specialised Hadoop-based services for analysing large volumes of data in their clouds such as Amazon EMR, IBM BigInsight, Hadoop on Google Cloud Platform, and Microsoft Azure HDInsight. In addition, the framework has been supported and complemented by a larger number of value-added big data technologies. Following the initial description of work, the Melodic platform will provide integrated support for applications developed using MapReduce programming framework. The selected resource management system for Melodic, Mesos, already supports Hadoop MapReduce. However, Melodic needs to provide solution for easy application modelling of the MapReduce framework-based applications.

8.2 Apache Spark

Apache Spark is a general-purpose open-source cluster computing framework with in-memory processing capabilities for high-speed big data analytics on diverse data sources. While being based on a programming model similar to MapReduce, Spark extends this model with an essential data-sharing abstraction, the *Resilient Distributed Datasets* (RDD). The RDD abstraction

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

represents resilient sets of objects partitioned across a cluster, allowing for parallel manipulation of the sets. RDDs are created as the user define new RDDs from stored data (e.g. from HDFS), or as new RDDs are derived through operations called "transformations" on already existing RDDs (e.g. using *filter* or *map* operations). Furthermore, RDD operations called "actions" are available to return results to a program (e.g. by counting the number of elements in an RDD). A significant advantage Spark holds over previous models like MapReduce, is the ability to allow users to mark these RDDs as persistent across computations. That is, by keeping specific RDDs in memory, a subsequent variety of queries from different computations against the same RDDs (i.e. data) can be performed without intermediate stores and loads from disk/storage. This data sharing can lead to significant speedups, particularly for applications based on iterative algorithms and interactive queries. The same functionality is key to Spark's generality [17], and by means of the flexible and efficient sharing provided by the RDDs and a unified API, Spark seeks to be the unifying engine for big data processing.

The Spark framework can run as a standalone deployment utilising HDFS or alternative storage systems such as HBase or Amazon's S3 directly, or it can be deployed together with YARN (Hadoop) or Mesos as a cluster manager, as depicted in Figure 22. Either way, Spark consists of a core engine providing distributed task dispatching and scheduling, exposed as an API based on the RDD abstraction briefly explained above. On top of the core, specialised processing libraries implement optimised solutions for different processing workloads. All together, the complete Spark stack efficiently supports a variety of workloads that previously required separate processing engines (e.g. SQL, streaming, machine learning, and graph processing).

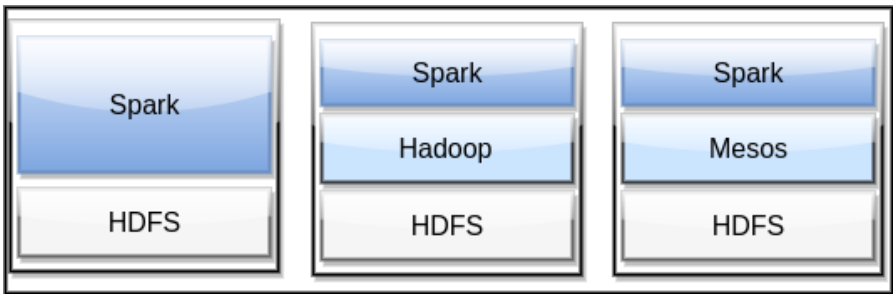

Figure 22: Apache Spark deployment scenarios

Salient Features

Apache Spark is a flexible and highly efficient big data processing framework. The RDD abstraction of the core engine provides Spark with resilient high-performance in-memory analytics capabilities. Teamed with a rich set of tailored libraries, including Spark SQL, MLlib for

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

machine learning, GraphX, and Spark streaming for processing data streams, the Spark framework covers a wide range of data-intensive application scenarios. Furthermore, the framework supports several languages, such as Java, Python, and Scala. For easy adaption, Spark is made flexible when it comes to coexisting with Hadoop; Spark can run standalone side-by-side with Hadoop in a cluster utilising HDFS, or integrated with Hadoop YARN as the cluster manager.

Integration in Melodic

All together, the salient features combined have made Spark into what is probably the most popular big data processing engine of today. Thus, integrating and supporting Spark in Melodic is likely to be instrumental for Melodic to gain broad acceptance and widespread use. By taking advantage of Spark's API and Spark's flexible capability to utilise different cluster managers, Melodic should integrate Spark with minimal, or preferably no, modifications to the Spark framework itself, thus making it possible for Spark application developers to take advantage of Melodic, and by that transparently making their Spark applications utilise Multi-Clouds, with minimal efforts.

8.3 Apache Flink

Apache Flink is an emerging open-source stream processing framework, originally developed as part of the European project Stratosphere [18]. Flink promises reliability and accuracy in stream processing by supporting stateful computations and event time semantics. Event time window support guarantees accurate results even when events are delayed and arrive out of order. In addition, Flink's data streaming engine achieves high-throughput and low latency without substantial parameter tuning and configuration. Flink's architectural stack is shown in Figure 23.

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

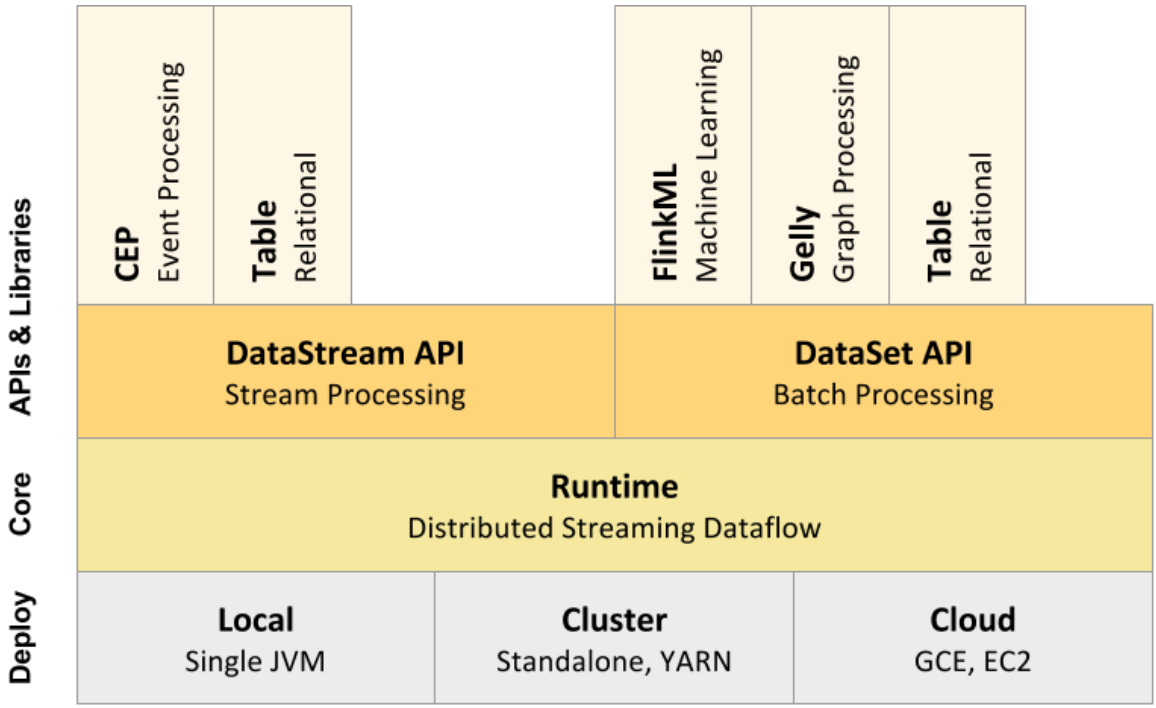

Figure 23: Apache Flink Stack [19]

In the following we outline the results of our evaluation of Apache Flink as a candidate stream processing framework for the integration in Melodic middleware.

Salient Features

Apache Flink supports both streaming and batch processing. For stream data processing, *event time semantics* enables accurate computation of results over streams where events do not follow a strict order, e.g. due to unpredictable communication latencies. The computations in Flink are *stateful.* Together with a light-weight check pointing mechanism, stateful computations enable Flink to be fault-tolerant and easily recoverable from failures without any loss of data. The windowing support in Flink is very flexible. Apart from windows based on time, count, or data, customised windowing mechanisms reflecting the application domain can also be defined. Apache Flink works well on very large clusters, both standalone and on top of a resource management system, such as Apache YARN or Apache Mesos.

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

Integration in Melodic

Apache Flink might very well have a prosperous future. However, at this stage Flink is found to be inferior to Apache Spark. First, Spark is a stable production-ready solution used by many data processing applications, while Flink is still under development and lacks production-level stability. Second, as per the current status, Flink has failed to attract major big data vendors, who on the contrary have invested heavily in Spark, such as IBM. With respect to Melodic, it is wise to wait to see Flink's reception in the market before investing in a potential integration. That said, due to Melodic's modular and extendable architecture with an abstract data framework layer, integrating Apache Flink or other third-party frameworks at a later stage should be workable.

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

9 Preliminary Architecture of Melodic

This chapter presents a preliminary architecture of the system that will be delivered by the Melodic project. The architecture is based on the gathered functional and non-functional requirements (Chapter 2-5), evaluated open technologies, and a detailed software component assessment in terms of reusability in the Melodic project (Chapter 6-8). The architecture described in this chapter will lay the foundation of first release of the Melodic platform (integration release due by the end of the first year). For the next releases, the architecture will be adjusted to further refine and enhance support of data-awareness and adaptive deployment and execution of big data frameworks and applications.

We first summarise fundamental generalised and use-case requirements leading to the proposed architecture of the system. Next, we describe existing (from partner EU projects) and anticipated future components of the software which will be delivered by the Melodic project. We also present an analysis of the approach on how to integrate different Melodic components. Finally, we define the high level preliminary architecture of the Melodic platform together with the description of each of its components, their interaction, and high-level work flow.

9.1 Requirements Summary

The purpose of the Melodic project is to deliver the ability to deploy and optimise big data applications on segregated resources acquired simultaneously from different Cloud providers and private infrastructures. The list of functional and non-functional requirements below is based on the expectations that the providers of such applications have from the system to be delivered. The expected requirements, detailed in previous chapters, can be technically summarised in the following four important aspect.

- Support a Cloud provider agnostic deployment for applications to avoid vendor lock-in.
- Ability to deploy data processing frameworks, applications, and other big data technologies in a scalable manner
- Ability to optimise the deployment of the Cloud infrastructure, the application itself and the big data framework based on utility functions reflecting user requirements and the current infrastructure capabilities
- Support for security, authentication and authorisation over the applications, their methods and the data manipulated by them

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

The software delivered by Melodic project is planned to be used in production environments. To this end, based on the experience of the Melodic consortium partners, the following key aspect are important to maintain high quality of the delivered software.

- Stability – system should work in a stable manner.
- Error handling – functional and non-functional errors should be properly handled.
- Monitoring and traceability – there should be possibility to monitor and trace all activities in the system.
- Ability to deploy applications based on a High Availability / Disaster recovery configuration
- Logging – unified approach for logging of all components with configurable logging levels.
- Backup – support for backing up system databases and most critical components.

9.2 Software Components

A large number of the Melodic software components, and corresponding architectural elements, will come from the PaaSage project. A summary of the assessment of the components and required extensions for the Melodic project are provided in Section 6.1. Most of the components will be used in a similar manner having the same role as in the PaaSage project. From the CACTOS project, the *Cloudiator* suite comprising *Colloseum*, Lance, and *Sword* sub-components, will be used to build the Melodic execution ware sub-system which will allow the deployment of legacy and data-intensive applications in Multi-Cloud environments (summarised in Section 6.2). The application modelling language used in PaaSage, CAMEL, will be extended to support data-modelling to enable Melodic to consider data aspects together with the applications requirements and constraints (Section 6.4).

After carefully evaluating Melodic-specific requirements and technological challenges arising from them as well as the current capabilities of the existing components from the partner projects, the following new components are needed to be developed for Melodic.

- *Adapter with Plan generator:* Due to the new requirements in the Melodic project, mainly related to big data applications and data life cycle management, a new Adapter component will be created. This new component will include the Plan generator functionalities from the PaaSage project.
- *Custom Allocation module for Mesos* - for efficient and suitable usage of the Mesos resource allocator, a custom Allocation module for Mesos will be realised. The module will exhibit two important features:

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

- Ability to allocate offers from agent nodes applying data locality aspect;
- Ability to compute reverse metrics, which could be used for generating events to trigger the scaling and descaling of the Mesos cluster. Such events could relate to having no offers of free resources in the case of Mesos cluster scaling up (add new agent nodes), or too many offers of free resources in the case of Mesos cluster scaling down (remove agent nodes).

- *Security* module: A new module will be created within the Melodic project, to handle application security issues. The XAMCL standard will be used to handle authorisation to application method invocation via a dedicated communication port. For authentication, the SAML2 standard will be used. This module will exploit the PaaSword Security Context Model as explained in Section 6.3.
- *Status and event service* – due to the orchestration based on business process management (See Section 9.3), a new service will be introduced which will be returning status and other information about the system operational services to UI and other systems. Also this service will be used to pass events to the Control Process. This service will not be handling events generated by Cloudiator and related to metrics.
- *UI Component:* Depending on the use case partner requirements, a UI component could be added to support application deployment and monitoring – to be decided in the future.

9.3 Component Integration

For the integration of the component and their interaction in Melodic, an Enterprise Service Bus (ESB) and a business processing management (BPM) based architecture will be used. The orchestration of the data and the actions flow in the system will be modelled as a process in an appropriate BPM language, which can be executed by business process manager. In this way Melodic workflow will be efficient and adaptable to the new requirements as they come. Several open source ESB and BPM solutions are currently being evaluated for the selection, such as MuleESB[46] and jBPM[47]. For the first release of Melodic, there will be no translation from domain models to the canonical model (and vice versa) as the current canonical model will be solely used. A canonical data model and respective (bi-directional) transformations from domain-to-canonical data models will be implemented for the subsequent Melodic releases. Similarly, metric collection and monitoring communication will be handled by ZeroMQ in the same manner as currently in PaaSage system in the first release.

[46] www.mulesoft.com/Mule-ESB
[47] https://www.jbpm.org/

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

A list of processes are to be implemented and executed via the selected BPM framework to support data and action orchestration. A control process will be realised which will handle the events that must trigger any action / process on the system. Then, based on the event type and current state of the system, one from the following dedicated processes will be executed:

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

Each process comprises steps which are either executed manually or map to the automatic invocation of components with logic using given parameters plus gateways that decide over the further flow of the process. The components with system logic will be invoked by the process via REST end points exposed by them on ESB.

The ESB and the corresponding adapters in each component will be responsible to translate data from the canonical data model (common data model of the Melodic system based on CAMEL) into the component's domain data model(s). Based on that approach, the translation of data will be uniformly performed in one place while there will be no possibility for different interpretation of data per each individual component driven by the meta-data.

This architecture, integration-oriented part introduces an abstraction layer between business flow and domain systems. The main benefits of such approach are:

- Unification of transport protocol - integration layer supports multiple communication protocols so connection of new components should be fairly easy
- Native support of data transformation between models (client domain model to system/canonical model)
- Traffic monitoring
- Support for service versioning

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

9.4 Architecture Overview

Figure 24 presents an overview of the preliminary high-level Melodic architecture. In the figure shows the main architectural sub-systems system. The figure also covers the general interconnection between components, while a detailed architecture will be presented in the architecture deliverable. The main Melodic sub-systems are:

- Upper ware – existing components and the new ones.
- Execution ware and Metric Collector – responsible for the deployment of the Cloud application, the monitoring infrastructure and the publishing of the corresponding measurements
- Integration layer – both control and monitoring plane. The control plane will be created from scratch (based on under consideration BPM and ESB technologies).

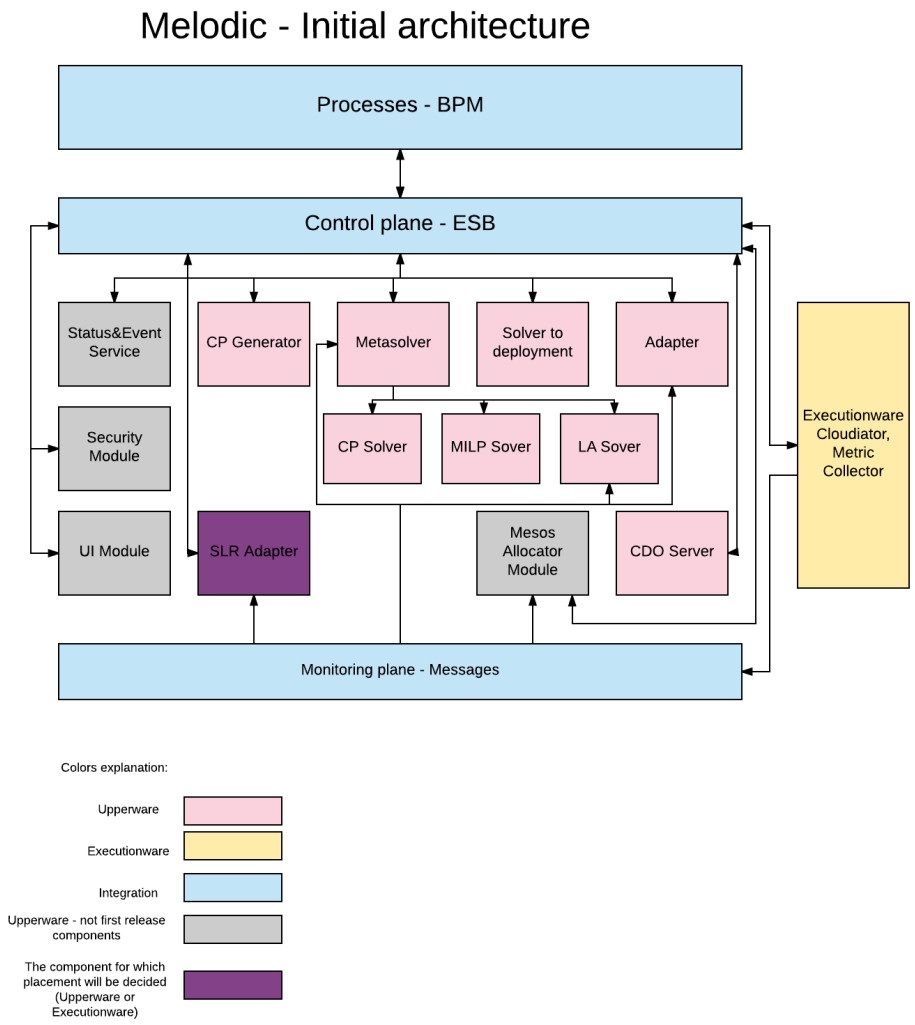

Figure 24: Melodic Preliminary Architecture

A high level integration overview is shown in Figure 25. For the sake of simplicity, some components are omitted to represent the general idea of integration, and thereby only a coarse-grained description of the Cloud application deployment process is supplied. The assumption is that all components will expose their services on ESB and will be invoked via ESB.

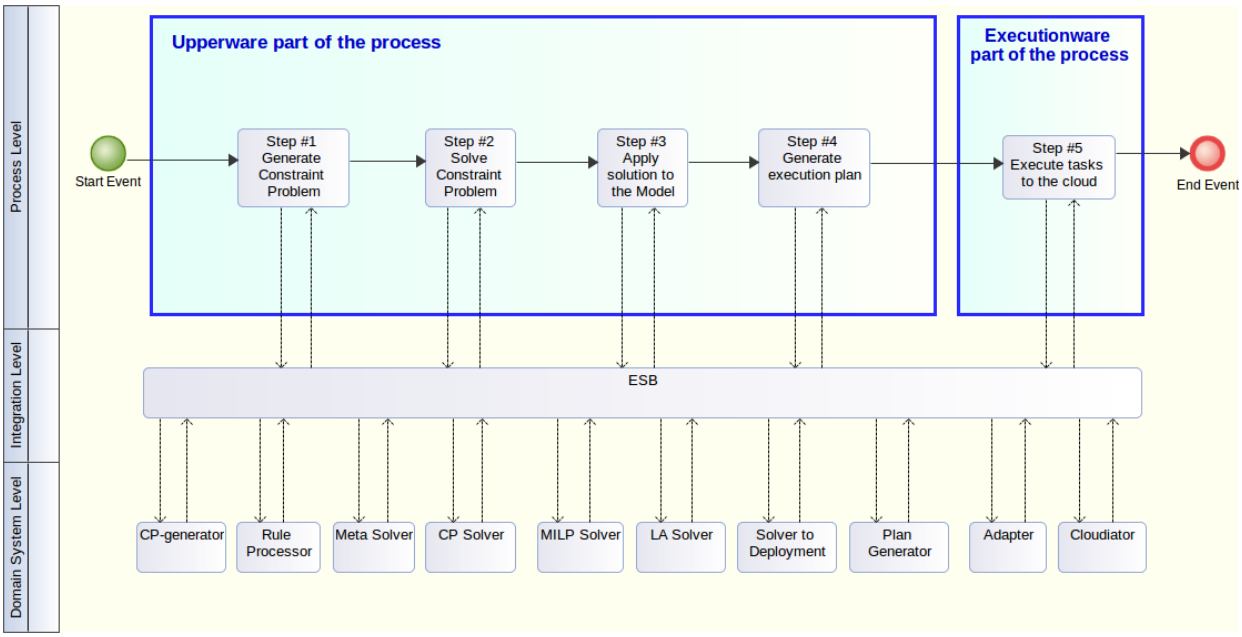

Figure 25: Component integration in Melodic

9.5 Control and Data Flow

The standard flow of data and actions in Melodic system for the successful deployment of new application flow will be as follows:

1. User uploads a new camel model of the application into CDO and invokes the *Status & Event* Service (by GUI or direct REST invocation).
2. The *Status & Event* Service uses the Drools rule engine from MuleESB to choose which process to initiate (based on event type, caller and any other configurable input in the Drools rules). For a new application deployment, this service invokes the Deployment process to deploy a new application.
3. The Deployment process invokes the CP-Generator component via a REST API with input parameter the (CDO) resource path of the CAMEL model.

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

4. CP Generator fetches the model from CDO and creates the CP model of the application for the solvers. The CP Generator stores the model in CDO and returns the resource path of CP model to the process.
5. The Deployment process invokes the Rule Processor with parameters: resource path of the Camel model and of the CP model, to create final CP model for solvers – this step will be removed as in the final solution Rule Processor should be integrated with CP-Generator component. The Rule Processor fetches model from CDO, removing unnecessary elements from CP model and updating that model in CDO.
6. The Deployment Process invokes the Metasolver component via REST API by passing the resource path of the CP model.
7. The Metalsover returns to the Deployment process which solver is chosen to derive the solution for the given problem (we note that the role of meta-solver might be further extended or modified as the work in WP3 progresses thus changes to the process steps 6-9 may be applied).
8. The Deployment process invokes selected solver for finding optimised deployment solution (the usage of more than one solver will be confirmed).
9. Selected solver fetches the CP model for optimisation from CDO and returns the optimal solution to the Deployment process. Each solver stores the updated CP model in CDO.
10. The Deployment process invokes Solver-to-deployment component via REST API, with given identifier of the best solution and the resource paths to the current CAMEL model and CP model to produce a concrete Camel deployment model.
11. The Solver-to-deployment fetches the proper models from CDO, updates the CAMEL model with the concrete deployment solution produced and stores it in CDO.
12. Process invokes the Adapter with the given resource of the camel deployment model to deploy. The Adapter component fetches the deployment model, generates and validates a deployment plan and then, if validation is successful, invokes the Cloudiator via a REST API exposed on ESB to start executing the deployment plan.
13. After the solution's deployment, the Adapter validates it (i.e., if it has been successful), monitors it and returns the proper information to the process. The latter means that the Adapter updates the camel deployment model in CDO to reflect the current application deployment structure.
14. Deployment Process returns positive information about the deployment to the *Status & Event* service.
15. The *Status & Event* service returns the successful deployment result to the invoking party.

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

Error handling at the process level will be handled by default error steps. The more detailed description of the error handling will be provided in D2.2 deliverable.

Each step within process should allow for re-execution (repeat execution of the step) with stored process context. At the ESB level the error handling should be implemented, with support to configurable retrying policy in case of technical error. The complete control and data flow is depicted in Figure 26.

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

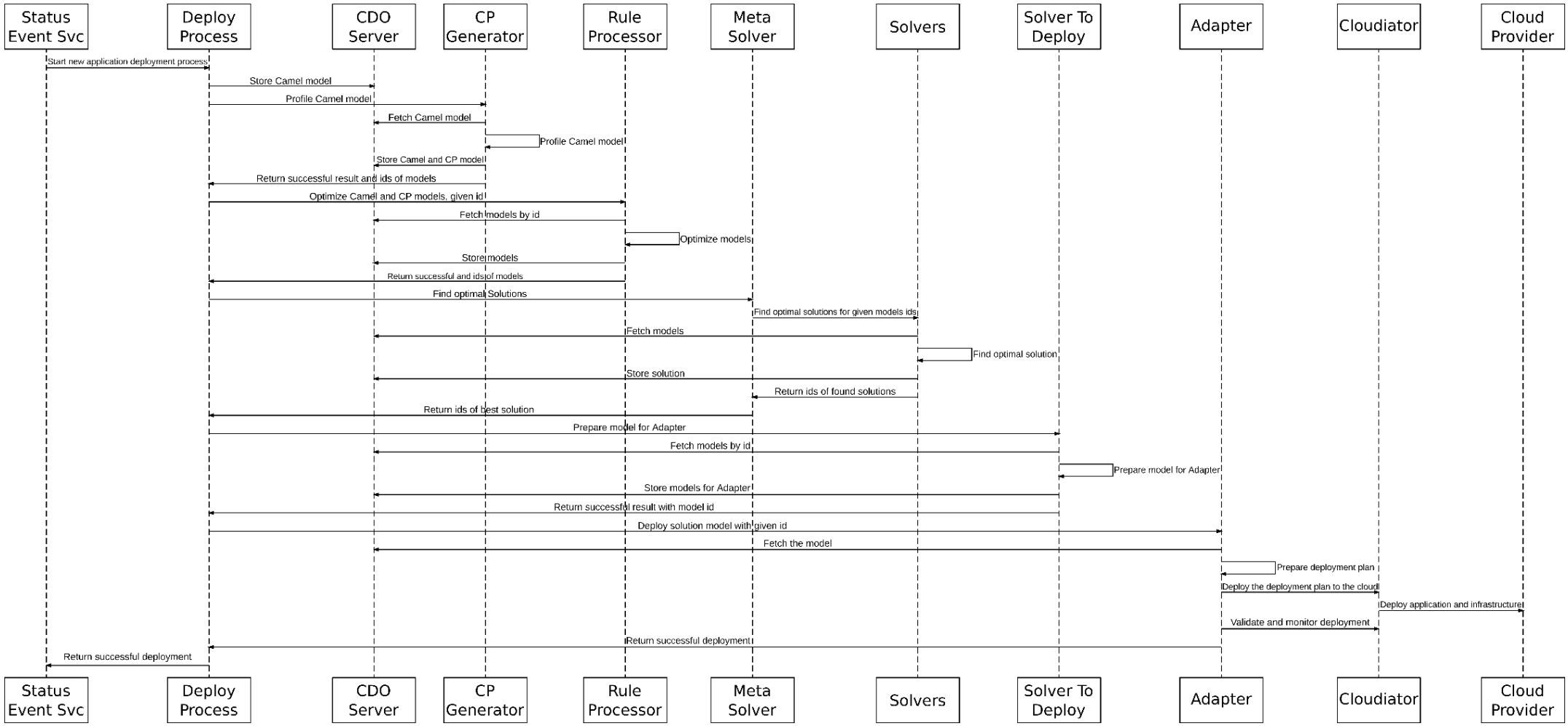

Figure 26: Control and data flow

Conclusions

The Melodic project aims to provide a unified automated solution to enable deployment, configuration, monitoring, and adaptation of data-intensive applications on geographically distributed and federated Cloud architectures. Based on the requirement analysis and a comprehensive technology evaluation, an initial system specification of the middleware platform has been drafted. A careful assessment of the available software components from three European projects, PaaSage, CACTOS, and PaaSword, together with the evaluation of the state-of-the-art big data technologies, a preliminary architecture of the Melodic middleware platform has been designed. The preliminary architecture includes identification of the system components, their integration and interaction, as well as the outline of the control and data flow in the Melodic system. The system specification and the preliminary architecture presented lays the foundation for the subsequent work in the Melodic project.

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

References

[1] P. Mell, T. Grance, and others, "The NIST definition of cloud computing," *Natl. Inst. Stand. Technol.*, 2011.

[2] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Key Challenges in Cloud Computing: Enabling the Future Internet of Services," *IEEE Internet Comput.*, vol. 17, no. 4, pp. 18–25, Jul. 2013.

[3] A. J. H. Simons *et al.*, "Advanced Service Brokerage Capabilities As the Catalyst for Future Cloud Service Ecosystems," in *Proceedings of the 2Nd International Workshop on CrossCloud Systems*, New York, NY, USA, 2014, p. 7:1–7:6.

[4] J. M. Cavanillas, E. Curry, and W. Wahlster, "The Big Data Value Opportunity," in *New Horizons for a Data-Driven Economy*, J. M. Cavanillas, E. Curry, and W. Wahlster, Eds. Springer International Publishing, 2016, pp. 3–11.

[5] N. Ferry, H. Song, A. Rossini, F. Chauvel, and A. Solberg, "CloudMF: Applying MDE to Tame the Complexity of Managing Multi-cloud Applications," in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, 2014, pp. 269–277.

[6] E. D. Nitto *et al.*, "Supporting the Development and Operation of Multi-cloud Applications: The MODAClouds Approach," in *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2013, pp. 417–423.

[7] E. D. Nitto *et al.*, "Supporting the Development and Operation of Multi-cloud Applications: The MODAClouds Approach," in *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2013, pp. 417–423.

[8] T. Kohonen, *Self-Organization and Associative Memory*. Springer Science & Business Media, 2012.

[9] N. Paladi, A. Michalas, and C. Gehrmann, "Domain Based Storage Protection with Secure Access Control for the Cloud," in *Proceedings of the 2Nd International Workshop on Security in Cloud Computing*, New York, NY, USA, 2014, pp. 35–42.

[10] G. E. Krasner and S. T. Pope, "A Cookbook for Using the Model-view Controller User Interface Paradigm in Smalltalk-80," *J Object Oriented Program*, vol. 1, no. 3, pp. 26–49, Aug. 1988.

[11] Y. Verginadis, A. Michalas, P. Gouvas, G. Schiefer, G. Hübsch, and I. Paraskakis, "PaaSword: A Holistic Data Privacy and Security by Design Framework for Cloud Services," *J. Grid Comput.*, pp. 1–16, Mar. 2017.

[12] V. C. Hu *et al.*, "Guide to Attribute Based Access Control (ABAC) Definition and Considerations," National Institute of Standards and Technology, NIST SP 800-162, Jan. 2014.

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

[13] S. Veloudis, I. Paraskakis, C. Petsos, Y. Verginadis, I. Patiniotakis, and G. Mentzas, "An Ontological Template for Context Expressions in Attribute-Based Access Control Policies," in *7th International Conference on Cloud Computing and Services Science (CLOSER 2017), Porto, Portugal, April 24-26, 2017*, .

[14] G. Blair, N. Bencomo, and R. B. France, "Models@ run.time," *Computer*, vol. 42, no. 10, pp. 22–27, Oct. 2009.

[15] V. K. Vavilapalli *et al.*, "Apache Hadoop YARN: Yet Another Resource Negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, New York, NY, USA, 2013, p. 5:1–5:16.

[16] B. Hindman *et al.*, "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center.," in *NSDI*, 2011, vol. 11, pp. 22–22.

[17] M. Zaharia *et al.*, "Apache Spark: A Unified Engine for Big Data Processing," *Commun ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016.

[18] A. Alexandrov *et al.*, "The Stratosphere platform for big data analytics," *VLDB J.*, vol. 23, no. 6, pp. 939–964, Dec. 2014.

[19] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Bull. IEEE Comput. Soc. Tech. Comm. Data Eng.*, vol. 36, no. 4, 2015.

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