UNIVERSITÀ DEGLI STUDI
DI GENOVA

Università degli Studi di Genova

Scuola Politecnica

A decentralized framework for cross administrative domain data sharing

PhD course in Computer Science and Systems Engineering

(systems engineering curriculum)

Candidate: Giancarlo Camera

Advisors: Prof. Pierpaolo Baglietto, Prof. Massimo Maresca

Cordinator: Prof. Giorgio Delzanno

PHD CYCLE XXXII

Table of Contents

Images

Tables

Listings

Abstract

Federation of messaging and storage platforms located in remote datacenters is an essential functionality to share data among geographically distributed platforms. When systems are administered by the same owner data replication reduces data access latency bringing data closer to applications and enables fault tolerance to face disaster recovery of an entire location. When storage platforms are administered by different owners data replication across different administrative domains is essential for enterprise application data integration. Contents and services managed by different software platforms need to be integrated to provide richer contents and services. Clients may need to share subsets of data in order to enable collaborative analysis and service integration. Platforms usually include proprietary federation functionalities and specific APIs to let external software and platforms access their internal data. These different techniques may not be applicable to all environments and networks due to security and technological restrictions. Moreover the federation of dispersed nodes under a decentralized administration scheme is still a research issue. This thesis is a contribution along this research direction as it introduces and describes a framework, called “WideGroups”, directed towards the creation and the management of an automatic federation and integration of widely dispersed platform nodes. It is based on groups to exchange messages among distributed applications located in different remote datacenters. Groups are created and managed using client side programmatic configuration without touching servers. WideGroups enables the extension of the software platform services to nodes belonging to different administrative domains in a wide area network environment. It lets different nodes form ad-hoc overlay networks

on-the-fly depending on message destinations located in distinct administrative domains. It supports multiple dynamic overlay networks based on message groups, dynamic discovery of nodes and automatic setup of overlay networks among nodes with no server-side configuration. I designed and implemented platform connectors to integrate the framework as the federation module of Message Oriented Middleware and Key Value Store platforms, which are among the most widespread paradigms supporting data sharing in distributed systems.

Chapter 1

1 INTRODUCTION

Applications often rely on external software platforms to access external data and to expose internal data. They offload advanced functionalities for data analysis, storage and management onto software platforms that greatly simplify data management and sharing. Recently, cloud computing introduced the Platform As A Service (PAAS) model to share the usage of virtual platforms among several end users. Platforms are managed by third parties that take care of access control, scalability, reliability, resource sharing etc., and expose an API to clients to allow them to use the platform functionalities. Despite its efficiency and simplicity, the adoption of the PAAS paradigm is often discouraged when applications handle sensitive data, especially in enterprises, and/or must guarantee performance levels. In those cases applications run on trusted software platforms, installed in enterprise protected administrative domain to avoid third-party data access. While data storage, analysis and management functionalities do not suffer from this choice, on the contrary sharing functionalities will be restricted to the hosts located in the same administrative domain. In this thesis I focus on the use cases (see chapter 7) in which sensitive data must be shared among different administrative domains to enable collaborative analytics and data integration from different sources owned by separate domains. For example, one of the use cases presented in chapter 7 regards data sharing among railway automation systems belonging to different administrative domains. In these systems sensitive data

regarding different areas of the railway networks are administered by separate entities. Data belonging to trains that cross different administrative regions must be shared with the administrative entity that will take control of the train. Another railway related use case is the creation of data analysis software for multimodal systems support where railway data are integrated with data coming from other transportation systems such as buses, airplanes or ships. Part of the second use case presented in chapter 7 describes decentralized sensitive data sharing in the port industry. The port industry involves different managers from carriers, shippers and agencies that owns data related to a small part of the entire process. Data must be shared in a decentralized and controlled way to create analysis and management services based on that information.

There are several candidate enabling technologies and methods to share data among different administrative domains. Direct access to resources and data is typically not allowed for security and performance reasons. As regards security, data needs to be protected by firewalls to avoid uncontrolled external access, whereas as regards performance direct communication between applications and data would increase the computational load both at the server side and at the client side, would cause availability problems and would increase network traffic for peer to peer communication. Indirect communication, through a software platform, is the solution. Software platforms are usually installed inside enterprise or department boundaries to maintain control over data location and access. They are easy to deploy and to use and speed up the development of complex systems. In order to share data with external domains, they frequently include a federation module that support data exchange among different administrative domains with no central control. However, convergence toward truly federated platform is still in progress. While centrally administered federated platforms were proposed and actually exist, on the contrary the federation of dispersed nodes under a decentralized administration scheme is still a research issue.

This thesis is a contribution along this research direction as it introduces WideGroups, a framework directed towards the creation and the management of an automatic federation and integration of widely dispersed platform nodes based on a client side

programmatic configuration of groups to exchange messages among distributed applications located in different administrative domains.

Following the high demand of administrative scalable distributed systems in these latest years (the most remarkable examples are the distributed ledger technologies), I envision a general purpose solution directed toward decentralized, dynamic software platforms federation. The system has a control plane that handles group creation, management and the configuration of overlay networks to let different nodes communicate, a data plane that handles the exchange of "WideGroups messages" and their global ordering, and platform-specific connectors supporting communication among software platforms using the WideGroups standard API.

More specifically WideGroups addresses the issue of federation of heterogeneous Software Platforms to support secure data access, in particular for platforms supporting message oriented data exchange and key value storage, and the issue of dynamic configuration of federated Software Platforms, in particular to support hot reconfiguration and service continuity.

This thesis describes the WideGroups model and architecture. The cross-domain scalable platform of WideGroups enables the extension of the software platform services to nodes belonging to different administrative domains in a wide area network environment. WideGroups let different nodes form ad-hoc overlay networks on-the-fly depending on message destinations located in distinct administrative domains. It supports multiple dynamic overlay networks based on message groups, dynamic discovery of nodes and automatic setup of overlay networks among nodes with no server-side configuration.
I designed and implemented connectors for platforms based on Message Oriented Data Exchange and Key Value Store, which are among the most widespread paradigms supporting data sharing in distributed systems. In particular

- Message Oriented Data Exchange supports data sharing through the message abstraction and Message Oriented Software Platforms offer primitives to exchange messages and to manage groups of clients. When clients belong to different platforms federation becomes mandatory to support message

exchange. The thesis starts by analyzing the available technologies and focuses on how these technologies solve broker federation to enable cross domain data sharing.

- Key Value Store supports data sharing by orchestrating data replication among data stores located at different sites and Key Value Store Platforms offer primitives to write and read data elements, freeing the clients from the need to manage the issues related to automatic data distribution and consistency. When the data stores are administered by different bodies, federation becomes mandatory to support enterprise application integration.

I propose to use WideGroups as a platform-independent framework to synchronize data among remote platforms excluding both data delocalization techniques and third party services.

To demonstrate the power of the WideGroups approach, I developed a WideGroups prototype, i.e., a software platform supporting the federation of Message Oriented Middleware platforms and of Key-Value-Store platforms.

I tested the prototype in two real world use cases, namely cross domain replication of railway data and Spreadsheet Space control plane decentralization. In the thesis I present the WideGroup concept, the prototype and the results of a set of experiments run on Apache ActiveMQ, Apache Kafka, RabbitMQ and Infinispan.

Chapter 2

2 BACKGROUND AND RELATED PROJECTS

In section 2 I briefly describe the two research projects I followed during the PhD. The concepts researched in these projects are strictly related to the ones investigated in my thesis project. Spreadsheet Space is a framework for secure and controlled spreadsheet data sharing among different administrative domains with fine grained data access control controlled by end users. I describe Spreadsheet Space in section 2.1. This section has parts that has been published in [17][18][19]. The integration layer is a middleware for heterogeneous platforms interconnection used for railway data analysis and storage. For both projects I describe how I investigated the platform integration issue and the cross domain data sharing technologies.

2.1 CROSS ADMINISTRATIVE DOMAIN SPREADSHEET DATA SHARING

Spreadsheets are a widely used tool for data visualization. The huge number of spreadsheet users and programmers [2] demonstrates the high impact of spreadsheet research and motivates an engineering approach [5]. Spreadsheets are also the best-known form of End User Programming [1][2]. As defined in [4] “end-user programmers” (EUP) are people “who write programs, but not as their primary job function”. They usually work with spreadsheets to support their accounting, scientific

research, decision support and data analysis activities using spreadsheets' formulas as a general purpose programming language suitable for tabular data manipulation. Spreadsheets' usage evolved from personal office tools aimed at improving people productivity to enterprise level tools aimed at supporting analytics and decisions [7]. Recently, office suites integrated a software module that enables collaborative authoring of office files, including spreadsheets, to facilitate the sharing process. Typically spreadsheets collaborative authoring applications, like Google Sheets or Excel online, need to delocalize the entire file in public cloud storage servers. This choice is not secure when spreadsheets analyze sensitive data and for enterprise usage because it exposes shared content to the risk of third party access. Moreover, available platforms usually provide coarse grained spreadsheet file sharing, where collaborators have access to all data stored inside a workbook and to all the spreadsheets' formulas used to manipulate those data. This approach limits users' possibilities to disclose only a small portion of tabular data and integrate data coming from different sources (spreadsheets or software platforms). For these reasons users handling sensitive information prefer to control fine grained confidential data exchange and their updates manually through copy, paste, attach-to-email, extract-from-email operations. However unsupervised data sharing and circulation often leads to errors or, at the very least, to inconsistencies, data losses, and proliferation of multiple copies. In this section I describe the Spreadsheet Space model and how it gives to users a different level of spreadsheet data sharing control, privacy and management. This approach enables collaborative analytics of tabular data focusing on fine grained spreadsheet data sharing instead of coarse grained file sharing. Spreadsheet Space implements a collaborative analytics system that implements a decentralized spreadsheet element sharing model and uses an asymmetric replication protocol to guarantee data consistency. This system handles tabular data sharing functionalities using software modules that guarantee an high level of spreadsheet data sharing control and privacy thanks to a platform that implements an end to end encrypted protocol that prevents third party access to confidential content. A PKI manages identities and keys. Data are never shared into public clouds but they are transferred encrypted among collaborators' nodes. In section 2.1.1 I briefly analyze the system describing its data model, its control plane software

modules, its data plane functionalities and finally how these modules are distributed on the architectural elements. I describe also the security layer: how the system uses end to end encryption, generates keys and handles access control to spreadsheet elements. This decentralized approach is particularly interesting for managing and sharing among **different administrative domains** sensitive enterprise data that cannot be exchanged using a public cloud and typically cannot leave the administrative domain of the owner unencrypted and uncontrolled. Spreadsheet Space works with Spreadsheet Space clients connected to software platforms and spreadsheet editing platforms, with fully distributed data storage servers (the view servers components) and with main control plane, security and event manager modules deployed in a cluster of cloud zero knowledge servers (Spreadsheet Space server component). With the Enterprise application data integration pattern end user programmers can **integrate data coming from different platforms** importing read only tabular data from software platforms that expose them through a Spreadsheet Space connector and integrate imported data within their spreadsheets.

2.1.1 Collaborative analytics systems model

A spreadsheet file is not a database or a service accessible from external processes and it was not designed for multi-user concurrent access. However, a large number of end user programmers that use spreadsheets for business intelligence and data analysis need to collaborative analyze spreadsheet data. They expose tabular data stored in their local spreadsheets and collect data from other spreadsheets or software platforms. Spreadsheet space, like WideGroups organizes users in domains. There are two types of spreadsheet data sharing: internal data sharing, where data are exchanged among users of the same domain connected to the same domain's private network and cross domain data sharing where data are exchanged among users located in different administrative domains (e.g.: business to business data sharing where users from an industry need to share data with external consultants and other industries). End user programmers organize data in their spreadsheets using user definable tabular data structures made of spreadsheet cells (2D arrays) that can have sizes (going from a single spreadsheet cell to a full sheet) set statically or dynamically (using respectively fixed ranges of cells or spreadsheet tables that can change their size during usage). I

distinguish two different sharing models that have these tabular data at the center of their design at two different levels of granularity: the **file sharing model**, if end users share the entire collection of tabular data elements and the **fine grained sharing model**, if they share one or more portions of the spreadsheet (In Figure 1 the orange box on the left contains data collected by the end user from another end user or software platform that is exposing this table to him. These data are not included in the blue box that contains the output of the end user analysis exposed to a set of selected users. These external users will see only this small portion of the spreadsheet. Other parts inside the spreadsheet are not shared and kept private into users' local spreadsheet.).

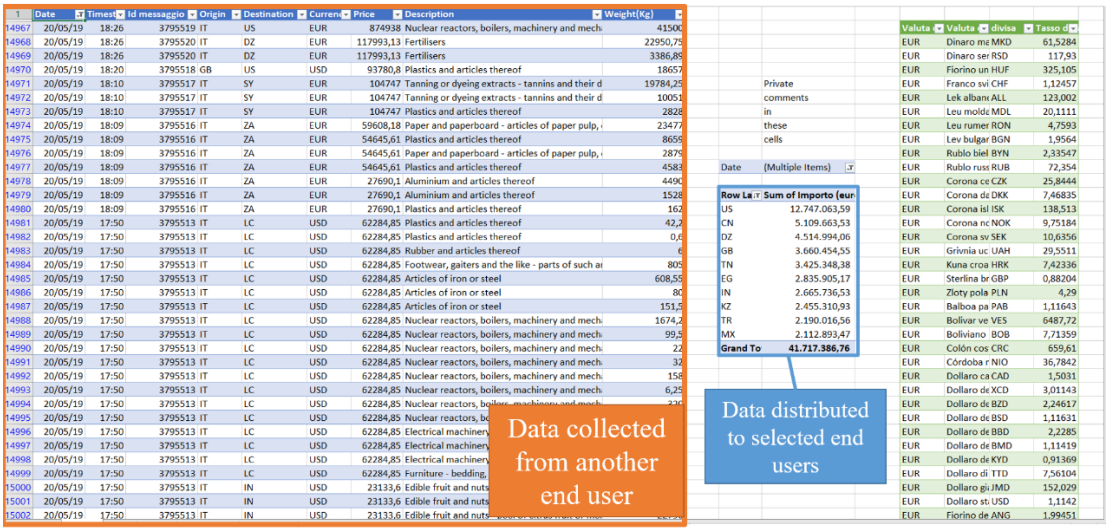


Figure 1 A spreadsheet with fine grained data sharing enabled.

End user collaborative data management and analytics can be considered as a complex data centric distributed system with various processes (the spreadsheets) distributed on the internet. This data centric system adopts a model based on tabular data objects shared among various processes (the spreadsheet applications and software platforms) in a distributed system. Collaboration happens sharing spreadsheet content and importing tabular data into spreadsheets at different levels of granularity that goes from the single cell to the entire collection of tabular data (the spreadsheet file). A collaborative analytics system must integrate or connect its software modules to a tabular data editing software (i.e. a spreadsheet application or a software that can

produce and process tabular data). I identify each node in this system based on its owner:

- **Collaborators**: nodes' owners directly involved in a shared tabular data element. A collaborator's node can be online or offline.
- **Cluster of collaborators**: a group of nodes' owners that have direct (read/write) access to the same spreadsheet data.
- **Off-cluster nodes**: nodes that do not belong to people/organizations directly involved in the cluster of collaborators. An off-cluster node can be online or offline.

Nodes include one or more software modules that provide basic services and functionalities for handling shared tabular data elements. I divide services into two categories: control plane services and data plane services. Control plane services are:

- **Sharing manager**: it has functions to create and delete a cluster of collaborators that share a table.
- **Collaborators manager**: it includes functionalities to add and remove collaborators to a cluster and change collaborators' permissions to read/edit data.

Data plane services handle the consistency of tabular data replicated on different spreadsheet applications. These services' implementation strongly depends on the architectural choices I will discuss later.

- **Consistency module**: a software module to maintain consistent replicated shared tabular data elements among different collaborators. It handles operations ordering and broadcast.
- **Shared memory**: an externally accessible storage element where a user can grant access permissions to collaborators.

End users can use spreadsheet formulas to develop "end user programs or software" inside spreadsheet applications and use the services described above to interconnect data coming from different sources. A collaborative analytics system shifts the single

user spreadsheet application to a multiuser environment. In [7] there is a description of the experiments has been carried on to understand what are end user expectations for a collaborative analytics platform. From that requirements comes a list of four collaborative analytics' interaction patterns identifiable in collaborative analytics systems:

- **Data distribution**: users share read only data from their spreadsheet distributing them to other end user programmers or to other software platforms. Data are correctly replicated and manually or automatically updated from the distributor.
- **Data collection**: end user programmers import read only data from other spreadsheets into their spreadsheets for analysis.
- **Enterprise application data integration**: end user programmers import read only data from software platforms and integrate them within their spreadsheets and with other end users' spreadsheets data.
- **Collaborative editing of data**: end user programmers collaborate on shared data structures that can be readable and writable.

2.1.2 Spreadsheet Space platform

Spreadsheet Space [7][16] is a collaborative analytics system that implements a decentralized spreadsheet element sharing model and uses an asymmetric replication protocol to guarantee data consistency. This system implements the model in section 2.1.1 using software modules that guarantee a high level of spreadsheet data sharing control and privacy. This approach enables collaborative analytics of tabular data focusing on fine grained spreadsheet data sharing instead of coarse grained file sharing for an higher control on what is shared. This solution works with a platform that implements an end to end encrypted protocol for sensitive data sharing that prevents third party access to confidential content. A PKI manages identities and keys. Data are never shared into public clouds but they are transferred encrypted among collaborators' nodes. In this section I analyze the system describing its data model, its control plane software modules, its data plane functionalities and finally how these modules are distributed on the architectural elements. In collaborators manager I describe also the

security layer: how the system uses end to end encryption, generates keys and handles access control to spreadsheet elements. Our decentralized approach is particularly interesting for managing and sharing sensitive enterprise data that cannot be exchanged using a public cloud and typically cannot leave the administrative domain of the owner unencrypted and uncontrolled.

2.1.2.1 Data Model

Spreadsheet Space implements a spreadsheet element sharing model. Spreadsheet elements' data can be the immediate result of a formula, values inserted locally, or data coming from external data sources. I model spreadsheet elements' data as shared tabular data structures concurrently accessed (read, written or read and written) by more than one process (end user programmed spreadsheet or software application). The fine grained external access to spreadsheet elements' data and not to entire spreadsheets gives end user developers a better control on what data are externally accessed and who can read/modify those data. Spreadsheet elements are abstracted into two data structures: **views** and **images** (Fig. 2). A view is a read only access to spreadsheet element's data: users can grant read only access rights to their internal spreadsheet elements at cell level granularity to other specific spreadsheets or software platforms they identify leveraging the spreadsheet reference technology to support data collection and data exposition. Views support the two types of spreadsheet elements: fixed and dynamic ranges. Only the view owner can edit view's data: this choice prevents view inconsistencies due to other users' concurrent updates and view's data certification as only the certified view owner can modify that data structure. Each view is stored with its metadata that include: view version, the owner id, a reference to the view's access control list, a reference to the view's history of edit operations and the address of where the view is stored.

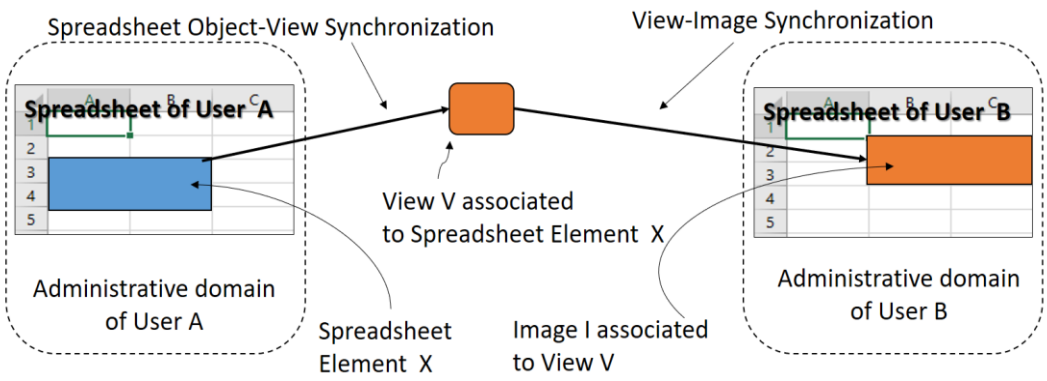


Figure 2 The Spreadsheet Space asymmetric view-image model.

2.1.2.2 Sharing manager

Spreadsheet Space gives a complete support to the four collaborative analytics' interaction patterns (Section 2) with three sharing management functionalities that I call: connection, exposition and collaborative editing. In [7] I describe in detail connection and exposition functionalities. In [16] I describe how I extended the Spreadsheet Space model to support concurrent access to shared spreadsheet elements leveraging our connection-exposition methods with collaborative editing sessions. In this section I recall some of the key concepts before focusing on enterprise use cases.

- **Connection** is the process that lets spreadsheets to connect their internal spreadsheet elements to external data sources and collect data from different spreadsheets and software platforms. Authorized collaborators can join images shared with them. Connection supports the data collection and the enterprise data integration patterns.
- **Exposition** is the process that exposes a spreadsheet element for external read only access to a set of collaborators (spreadsheets or platforms) creating a view. It supports the data distribution pattern.
- **Collaborative Editing** Creation is the process that grants editing permission to spreadsheet elements' data to a group of collaborators (spreadsheets and platforms) the. It supports the collaborative editing of data pattern. Collaborative editing works by integrating data coming from different spreadsheet elements' views. The collaborative editing creator initializes a collaborative editing session with virtually shared object that stores pointers to the views of the session

participants. It creates a collaborative editable spreadsheet element state table. Collaborative editing creator exposes the first view of the collaborative editing session accessible by other collaborators. A leader election starts among online collaborators to decide who will be the master node that will order operations. Spreadsheet Space uses RAFT consensus algorithm to manage collaborative editing clusters. The correctness of our approach is tightly coupled with the correctness of the RAFT consensus algorithm [17].

- **Join Collaborative Editing**: it connects to the global image and exposes a personal view of the local spreadsheet element to the session participants. The imported shared spreadsheet element is therefore both a view and a set of images collapsed in the global image (a view of my local changes exposed to other participants and a global image of other participants' changes). It creates a collaborative editable spreadsheet element state table to handle the collaborative editable spreadsheet element.

These methods are essential for distributed analytics and decision support systems with spreadsheets: applications must be interconnected to consolidate information coming from different processes and share results to other users, platforms or spreadsheets.

2.1.2.3 Collaborators manager

View owner can modify the set of target users changing the view's access control list. Views' data are stored encrypted with a session key generated by the view owner. Adding a user means sending the session key to the new collaborator encrypting it with its public key. Removing a collaborator triggers the generation of a new session key that must be sent to all collaborators to decrypt new operations. If the collaborator is part of a collaborative editing and is the current leader, the removal triggers a new leader reelection using the RAFT algorithm.

2.1.2.4 Shared memory module

Data plane has two modules: consistency module and shared memory module. The shared memory module handles an always on externally accessible memory region where external collaborators can access to data in read only mode. It contains only encrypted view objects that belongs to a specific collaborator. It has two primitives:

update through which spreadsheet element updates are sent to the corresponding view and **refresh** through which an image synchronizes its local state to correctly update data on the connected spreadsheet element. The update and refresh primitives can be triggered manually or automatically depending on users' requirements. Only the view owner can call an update operation and modify view data and only authorized collaborators can call a refresh operation to read data. The consistency module guarantees consistent and correct updates of a view.

2.1.2.5 Consistency module

The consistency module works as a replicated state machine that receives ordered operations to apply to replicated tables (spreadsheet elements, views, images). Each view edit operation has cell-grained granularity. Columns and rows deletion can cause a shift of cells. This shift generates a set of single cell editing operations ordered like any other single cell editing operation. View updates are incremental edits to the original view and are stored in view's operations' log ordered with an incremental version number that works as a logical clock. Lamport logical clocks [10] are a good choice to guarantee local order. View-Image replication protocol must guarantee only the local order of operations as views are single writer multiple readers registers that can be modified only from their owner from one client application at a time. The active owner's consistency module takes care of operations' ordering. Perceived user experience using a spreadsheet editing application that contains spreadsheet elements exposed or connected through Spreadsheet Space is tightly coupled with the perceived response time of the system defined by Ellis and Gibbs in [9] as "time necessary for the actions of one user to be reflected by their own interface". Spreadsheet Space synchronizes highly available spreadsheet elements and avoids write locks and read locks on their data. A non-blocking highly available application can be obtained only relaxing the strong consistency property as it states the famous CAP theorem [6]. The CAP theorem affirms that a strongly consistent system cannot guarantee availability and partition tolerance at the same time. Strong consistency can be obtained only with blocking transactions during the view-image synchronization process. Therefore I chose a different level of consistency called eventual consistency or optimistic replication. Ellis and Gibbs introduced in [9] the basis of eventual consistency (called

also optimistic replication) with the dOPT (distributed operational transformation) algorithm. Spreadsheet Space's data plane, like other groupware office suite applications and collaborative spreadsheet editing software, implements a non-blocking asynchronous process executed in background following this concept. Spreadsheet elements are eventual consistent with the corresponding views and images during the update and refresh operations. Images will be the exact replication of their corresponding view when the system is at quiescence (when there are not updates anymore). Before and during the view image synchronization process clients are never locked in reading images even if the image is not up to date with its corresponding view. The asymmetric views' connection and exposition operations are safe, consistent and secure, but they strongly limit end user development possibilities as I observed during the user acceptance evaluations listed in [7]. Collaborative analytics applications may have to work on the same spreadsheet elements and concurrently modify them at the same time or at different pace. With the Spreadsheet Space collaborative editing functionality I leverage the security and safety of the view-image abstraction to create virtually co-editable spreadsheet elements modifiable from different processes. Eventual consistency for a co-editable spreadsheet element table must guarantee convergence, causality preservation and intention preservation properties [13]. For fault tolerance and security Spreadsheet Space distributes the central ordering service at the edges inside the consistency module enabling single nodes to order operations. Spreadsheet Space leverages its asymmetric mechanism (the view-image paradigm) to generate virtually co-editable tables reconstructed client-side. Therefore, our collaborative editing approach can work with end to end encryption. Active collaborators own a consistency module to apply operations to the shared element. In order to reconstruct the same co-editable spreadsheet element we need a global order of operations. The order of operations is generated only from one consistency module at a time that acts as a leader and works like the view owner described above. I assign to one of the online collaborators the role of the leader using RAFT consensus algorithm [17]. The leader owns a special kind of view called "global view". Collaborators, thanks to the RAFT algorithm, elect a leader and import locally a global image connecting it to the global view of the leader.

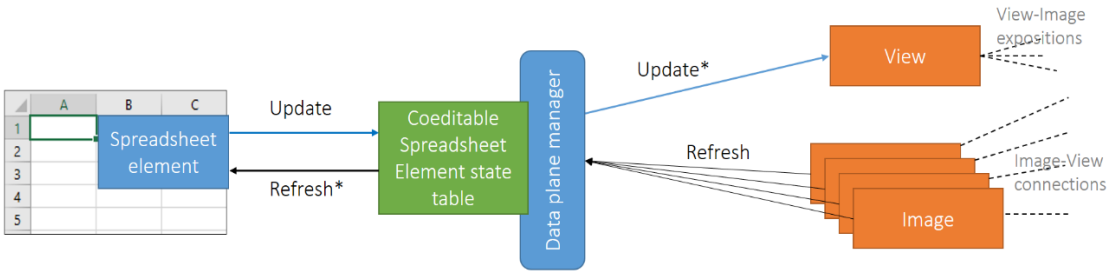


Figure 3 Coeditable Spreadsheet Element State Table at the collaborators' leader consistency module.

In Fig. 3 I show a high level scheme of how the client side co-editable spreadsheet element reconstruction works on the leader node. I model the leader's collaborative editable shared element as a collection of images and views. Each node that participates to a collaborative editing session creates its own view and exposes it to collaborators to transmit its own operations to other editors. After the election, the leader connects to all collaborators' images to merge edit operations coming from different images in a single ordered log of operations. At the same time each non-leader collaborator joins the global image associated to that collaborative editing session that contains other users' operations ordered by the elected collaborator. Each virtually shared element's view contains local operations of one collaborator, while the global image contains consistently global ordered collaborators' remote operations. Leader's consistency module handles update and refresh operations reconstructing the co-editable spreadsheet element in a co-editable spreadsheet element state table. The Co-editable Spreadsheet Element State Table is a data structure that contains a table of list of "operation" objects. Each operation object contains its value, the position in the spreadsheet element, the user id and a timestamp. Objects received from connected images and written locally with update operations are properly ordered using their version numbers and the time of their arrival to the leader's node. I extend the view version numbers logical clocks with the arrival time to guarantee global order of operations. The last operation object on the list is the object visualized on the corresponding cell of the spreadsheet element. The consistency module on followers' nodes guarantees the co-editable spreadsheet element state table eventual consistency. The consistency module orders operations coming from the global image with already visualized (but not applied to the log) local spreadsheet element edit operations. A fixed

leader can be a single point of failure for the system especially working with not centrally administered collaborators' nodes. Using RAFT consensus algorithm failures and network partitions are handled properly with the election of a new leader. [17]

2.1.2.6 Platform architecture

Spreadsheet Space works with Spreadsheet Space clients connected to software platforms and spreadsheet editing platforms, with fully distributed data storage servers (the view servers components) and with main control plane, security and event manager modules deployed in a cluster of cloud zero knowledge servers (Spreadsheet Space server component).

- **Spreadsheet Space server:** Spreadsheet Space server is a group of off-cluster nodes (nodes whose owners are never involved in a collaboration). Their owner cannot look at data shared for privacy and security reasons. Therefore it never receives spreadsheet elements' data even if they are encrypted. Spreadsheet Space Server component implements four modules: an event manager, a sharing manager, a collaborators manager and security module. Spreadsheet Space security module implements a PKI (public key infrastructure) to identify users and handle public keys for data encryption. It defines roles inside the platform and handles view-image data access. A sharing manager receives and handles view expositions and image connections. It is connected to the security module for granting proper authorizations based on views' access control lists. Control plane and notification primitives do not transfer data directly, but use a reference to the view id, its version and the view server id where data are stored and managed by a sharing memory module. Once operations are authenticated data are transferred directly and encrypted among clients and private shared memory modules. The Spreadsheet Space event manager notifies clients of view creation and handles view-image replication protocol dispatching view updates messages to authorized connected images. Also the event manager is connected to security modules for authorization checks.
- **Spreadsheet Space client:** It implements the consistency module, the sharing manager client and the control plane modules' clients that includes the notification channel client. The sharing manager client handles view exposition from internal

spreadsheet elements to external views and internal spreadsheet elements image connections with the possibility to setup access control to those elements and authorizations. The consistency module works with the shared memory module to handle runtime synchronization of spreadsheet elements with the Spreadsheet Space abstractions and co-editable spreadsheet element reconstruction. Spreadsheet Space client comes into two different forms: Excel addin or SDK. The Spreadsheet Space Microsoft Excel addin (Fig. 3) is installed together with the "notifier" client application. The "notifier" client is an external service running in background to handle network calls to the Spreadsheet Space server and the view servers. The two software components, addin and notifier client communicates using operating system's pipes, while notifier client communicates with the servers (the Spreadsheet Space server and the view servers) with REST web services. Spreadsheet Space implements Java, C#, Javascript SDKs and REST APIs to integrate data coming from external platforms and applications. The SDK includes all the Spreadsheet Space client's functionalities.

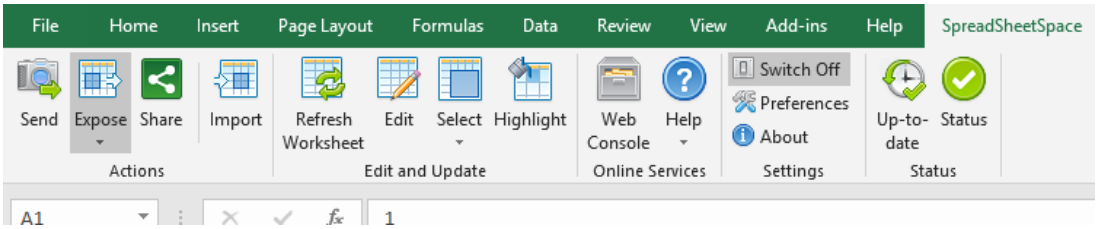


Figure 4 Spreadsheet Space addin

- **View servers**: Each collaborator associates himself to a view server that runs the shared memory module. Nodes containing view data can be theoretically the same personal devices and computers running the Spreadsheet Space client, but they must be always on and give continuous external access to locally stored views to make this protocol work. For technical reasons always on view servers are provided connected to client applications that can store encrypted views of spreadsheet elements to overcome availability problems of personal devices' peers. These servers have to be installed in the cloud or in personal data centers and can communicate with clients and to each other. Each user can have his personal set of view servers that can be accessed only by him. A view server is a WORM (write once read many) and

certified storage element that stores the operations performed on a view encrypted with the public keys of the users that can access to that operations.

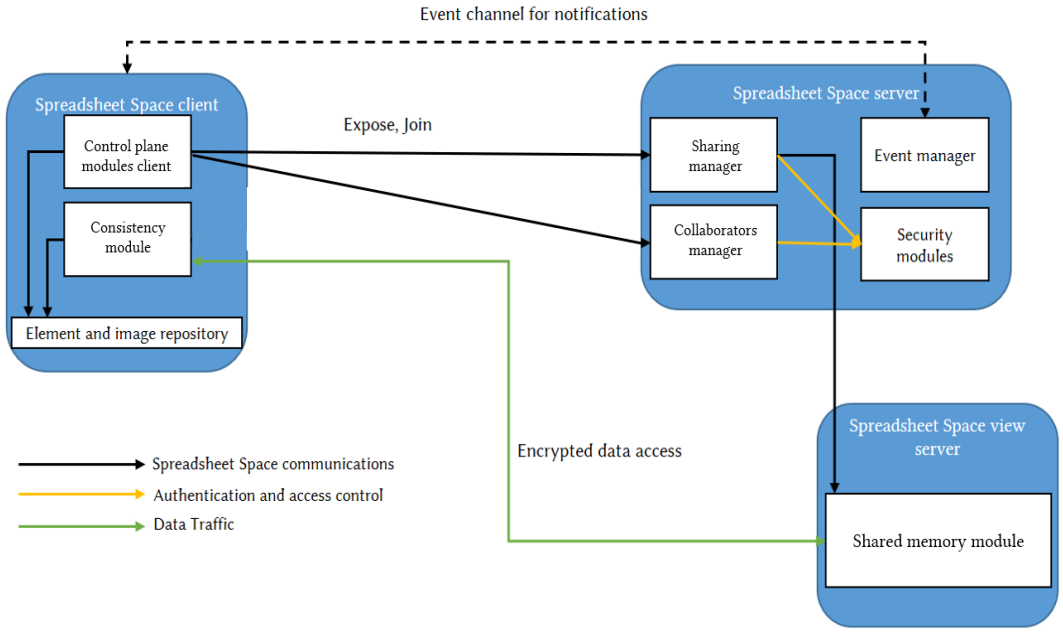


Figure 5 Spreadsheet Space platform

2.2 PLATFORM CONNECTORS

Choosing a particular software platform can cause issues regarding the so-called vendor lock-in: applications use platform related APIs to access software platforms functionalities and need to learn software-platform internal methods for configuring functionalities such as clustering and federation. In order to let different platforms share data and collaborate I chose a set of open source platforms that implements message oriented middleware functionalities and a set of platforms that implements key value store functionalities and created the so-called connector APIs. The platforms' choice was driven by the requirements of the Integration Layer project described in 2.2.3 and by the analysis of the mostly used solutions for the two typologies. I chose Mosquitto, Apache ActiveMQ, Apache Kafka and RabbitMQ as message oriented middleware implementations. For the key-value store typology the choice fell on in memory key value store platforms. This choice was driven by the analysis of their efficiency and faster data access compared to other NoSQL DBs and data grids. These two characteristics were fundamental for the Integration Layer use case. I analyzed the two

most used platforms in this category: Infinispan and Hazelcast. In this chapter I focus on the definition of generic connectors for message oriented middleware and key value store connectors in chapter 2.2.1 and 2.2.2. It is particularly interesting for the WideGroups project as some of its fundamentals were transported to the WideGroups implementation. I briefly overview the analysed platforms and the connector APIs designed as the outcome of this analysis. I conclude this chapter with a brief description of the Integration Layer for railway data analysis and how I applied connectors to this use case.

2.2.1 Key Value Store platforms connectors (in memory data grids)

I analysed key value stores focusing on a particular implementation: the in memory data grid platforms. I report here the definitions of data grid and in memory data grid from the official Infinispan documentation: "A **data grid** is a cluster of (typically commodity) servers, normally residing on a single local-area network, connected to each other using IP based networking. Data grids behave as a single resource, exposing the aggregate storage capacity of all servers in the cluster. Data stored in the grid is usually partitioned, using a variety of techniques, to balance load across all servers in the cluster as evenly as possible. Data is often redundantly stored in the grid to provide resilience to individual servers in the grid failing i.e. more than one copy is stored in the grid, transparently to the application. An **in-memory data grid (IMDG)** is a special type of data grid. In an IMDG, each server uses its main system memory (RAM) as primary storage for data (as opposed to disk-based storage). This allows for much greater concurrency, as lock-free Software Transactional Memory techniques such as compare-and-swap can be used to allow hardware threads accessing concurrent datasets. As such, IMDGs are often considered far better optimized for a multi-core and multi-CPU world when compared to disk-based solutions. In addition to greater concurrency, IMDGs offer far lower latency access to data (even when compared to disk-based data grids using solid state drives). The trade-off is capacity. Disk-based grids, due to the far greater capacity of hard disks, exhibit two (or even three) orders of magnitude greater capacity for the same hardware cost." [1]

[1] https://infinispan.org/docs/dev/titles/overview/overview.html

In this section I introduce IMDG platforms which has been analysed: Infinispan, Hazelcast and Apache Ignite. I chose platforms that implement the JSR-107 "JCache" specification, the standard Java caching API. JCache has a standard reference implementation that is not recommended to use since "it causes some concurrency issues"[2], implementations like Infinispan, Hazelcast, Apache Ignite, Oracle Coherence, Terracotta Ehcache are preferable. I chose Infinispan and Hazelcast for a deeper analysis of their APIs in order to develop a standard connector for in memory data grids. This standard connector maps JCache APIs and dynamically load the libraries and the implementation chosen among Infinispan, Hazelcast and Ignite.

2.2.1.1 Platforms Analysis

- **Infinispan: Infinispan** is an open source platform that implements an in memory data grid. It provides in memory key-value data replication and distribution across a network of data structures embedded into java applications and/or application servers running on virtual or physical machines. The Infinispan IMDG provides an integration layer for a group of servers and applications that have to access to the same data. Each Infinispan node is an instance of an Infinispan cache object and runs on a separate Java virtual machine (JVM). A cluster is a group of nodes. Infinispan is self-healing and self-discovery. Nodes can be added to and removed from a cluster at runtime. Thanks to this functionality, it is possible to scale the system without compromising the data availability. Infinispan is written in Java and exposes a map interface based on JCache specifications (JSR 107) . Data are stored in a cache as key-value pairs. Clients insert key-value pairs using put(key_object, value_object) functions and retrieved with get(key_object) functions. Key and value objects are Java objects: standard Java types can be used as String, Integer, etc. or custom Java types provided by the user. Infinispan provides bindings for other non-Java languages by means of an ad-hoc communication protocol (HotRod).

[2] https://www.baeldung.com/jcache

Infinispan can be used in two different modalities: embedded or remote server. These two modalities can be combined using the "compatibility mode" and sharing the same cache manager.[3].

- **Embedded (P2P):** Applications which need to use Embedded mode must include calls to Infinispan libraries in the code, adding the Infinispan jars in the classpath or through Maven configuration. Infinispan will then execute in the same JVM together with the applications, as shown in Figure 1. Embedded mode allows a more complete usage of Infinispan, at the cost of developing the application code only in Java language, and Infinispan instance must reside on the same JVM of the application. Embedded mode is more appropriate for non-distributed caches or caches distributed on a local area network where a cluster on a WAN is not needed.

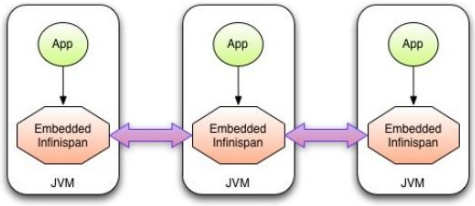


Figure 6 Infinispan embedded

- **Remote server (client-server with REST, Memcached, HotRod):** Infinispan is based on WildFly, the evolution of JBOSS AS. Remote server mode of Infinispan is installed as a module of WildFly : different modules take care of Infinispan management, client-server communications and communications via JGroups (the default protocol for communicating between Infinispan instances). In this mode, the client application is running outside the server. Infinispan provides client-side libraries for connecting to the server and for server management. The client protocol supported by Infinispan are REST, memcached, HotRod and Websocket. Infinispan features available

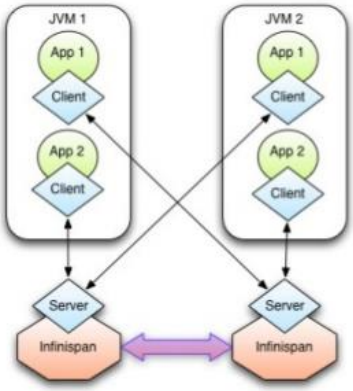


Figure 7 Infnispan remote server

[3] http://infinispan.org/docs/stable/user_guide/user_guide.html#interoperability_between_embedded_and_remote_server_endpoints

to the client depend from the protocol used for connecting to the server, and the same holds for the object types which can be used. Not all protocols allow the usage of all Infinispan functions, being HotRod the most complete one. Moreover, the client/server connection is slower with respect to the direct calls to remote cache. Nevertheless, with the Remote server mode Infinispan gains more flexibility, scalability, can be distributed over WAN and used by applications written in differente languages and for different execution environments.

Infinispan main objects are:

- **ENTRY**: key-value pair stored inside the cache
- **CACHE**: container of key-value pairs
- **CACHE CONTAINER**: Infinispan entity which can be used as configuration source for one or more data cache with common features (such as the type of communication protocol within the cluster, the security policy and so on). Infinispan internally manage a cache container entity as a cache manager which controls caches contained in the specific cache container.
- **NODE**: an instance of Infinispan running inside a JVM.
- **CLUSTER**: group of one or more instances of Infinsipan (nodes).

See the corresponding appendix for more details about Infinispan.

- **Hazelcast:** The Hazelcast In Memory Data Grid platform provides a scalable environment for distributed object caching, allowing different deployment modalities and cluster topologies and enabling different networking options, being it in a Local Area Network or in a complex geographically distributed scenario. Hazelcast provides a complete set of APIs for integrating the platform, allowing distributed caching, processing, data inquiry and event notifications. Even if the Hazelcast core is built in Java, the platform provides client APIs in the most common languages and technologies like MS .Net, C++, Python, Node.js

The Hazelcast IMDG solution has two main different deployment modes: **embedded** and **client/server**

In the **embedded** mode, different Hazelcast **nodes** participate in order to create a distributed cache **cluster**. Every node maintains data in **partitions**: partitions are owned by 1 or more nodes and replicated to 1 or more other nodes.

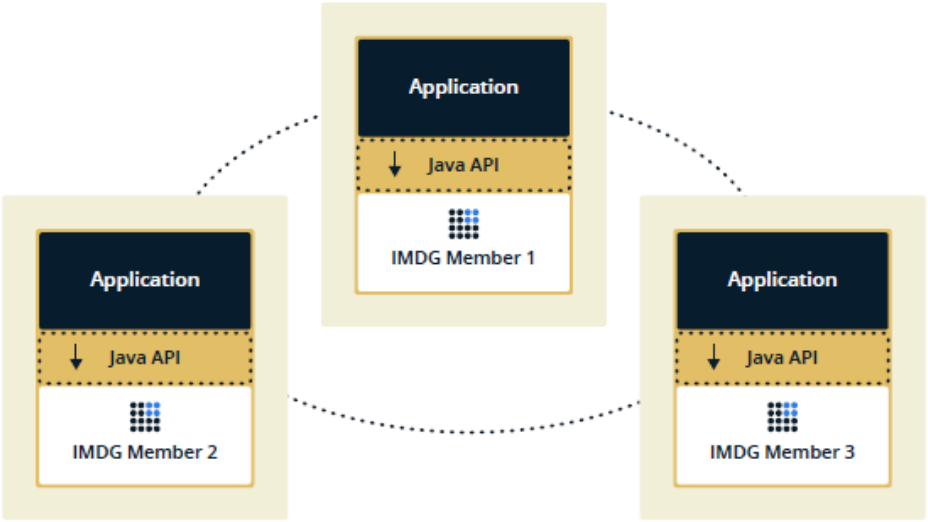


Figure 8 Hazelcast embedded deployment

Each Hazelcast node runs in a separate JVM, usually in a different network node. The Hazelcast runtime is included in each node and each JVM runs the Hazelcast node runtime plus the application code. The Hazelcast runtime takes care of keeping local data synchronized with the other cluster members and provides Hazelcast APIs to the application code.

In the **client/server** mode, also known as **client plus member** modality, application logic runs outside of the Hazelcast cluster and interact with the cluster through the client APIs. From an architectural point of view, this means that this type of deployment modality isolates the application code from purely cluster-level events.

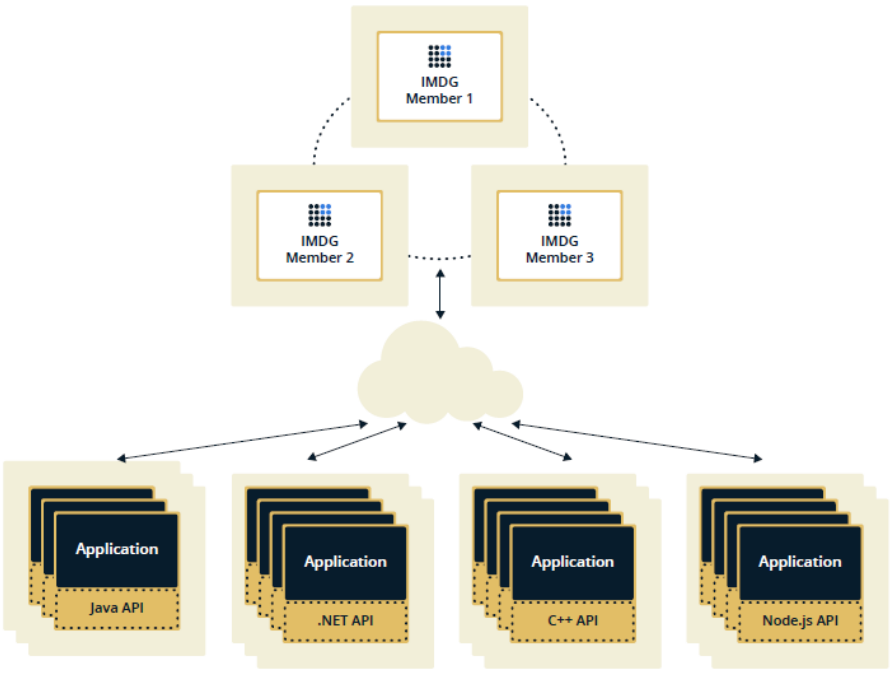


Figure 9 Hazelcast client/server (client plus member) deployment

While in the embedded scenario the application APIs are available only for the Java platform, adopting the client plus member deployment modality allows the exploitation of the client APIs written in different languages. At the moment, the available languages for the client APIs are Java, .NET, C++, Node.js, Python and Scala.

Moreover, the communication protocol between the client(s) and the Hazelcast cluster members exploits a publicly and freely available binary protocol, which allows to develop client APIs in other language, if needed.

In summary, the noteworthy strength points of Hazalcast IMDG solution, with respect to the runtime environments and deployment modalities, can be considered the following:

- No runtime server needed for any deployment modality: each Hazelcast node can be executed in a standalone JVM, so that no specific server is needed for the "server" part of the client/server modality, since the Hazelcast runtime library already manages incoming connections from the client APIs via multithreading, besides managing the discovery phase and join of the other cluster members.

- Most commonly languages available for integrating Hazelcast client APIs in the cluster clients
- The binary communication protocol used for client/server communication is freely available, so that new clients APIs in different languages and technologies can be implemented.
- Finally, even if a specific application environment is not required for running a node, Hazelcast has been successfully integrated into the Spring Framework.

Hazelcast supports federation. For a deeper analysis of Hazelcast federation across multiple datacenters see section 4.1.1.2 .

2.2.1.2 Key value store connector

Key Value Store ConnectorAPI is an abstraction of a generic key value store (or in memory data grid). ConnectorAPI is an interface for decoupling applications from the details of the underlying real implementation. This Interface will provide the client application with all the functions needed to interact with a generic KEY/VALUE store for storing/retrieving objects, removing objects or perform other types of operations. Instances implementing this interface may be obtained via the Key Value Store Factory, provided that a Factory Delegate class implementing Key Value Store Factory Delegate uses the proper configuration properties. The Factory-based implementation has been developed in order to decouple the definition of the Interfaces method from the real implementation, so that client applications will not have to deal with underlying details provided that a specific Factory Delegate class has been developed for communicating with the specific KEY/VALUE store. Each KEY/VALUE store (either based on IMDG or other technology like e.g.nosql) will be exposed via this API. Applications will use methods provided by this interface in order to communicate with the In Memory Data Grid. It is assumed that every KEY/VALUE store will provide one or more caches where the Key/Value pairs will be stored. When interacting with the Key/Value store via this interface a default cache will always be assumed, unless the specific interface method allows to specify a specific cache. In the chapter 5.5.7 I include a partial list of methods provided by the Key Value store connector API.

2.2.2 Message Oriented Middleware platform connectors

In the last two decades efforts from academia and industry produced solutions to enable and efficiently handle communication in distributed systems among heterogenous and loosely coupled applications. A complex distributed system formed by a large and varying number of applications works better with asynchronous communication. An asynchronous communication pattern lets two applications communicate without waiting for the other application to respond. One of the most used and studied communication services is the message oriented communication, frequently used as the asynchronous alternative to remote procedure calls that typically block the client until its request has been processed by the server[50]. In message oriented communication endpoints can play two different roles: message producers, when they send messages, and message consumers, when they receive messages. There are two different types of message oriented communication: the first one requires producers and consumers active at the same time to enable communication among them; the second one needs a middleware that take care of message delivery removing the necessity to have all parties online during transmission and providing a fully decoupling of message producers and consumers. Our work focuses on the second category of systems that are generally known as message-queuing systems, brokered Message Oriented Middleware or just Message-Oriented Middleware (MOM). MOMs rely on queue managers (or brokers) to handle communication among clients. Each client connects to a message queue manager or a broker to write messages on its send queues and read messages on its receive queues. A queue manager or a broker must deliver messages from a send queue to one or more receive queues. Existing message oriented middleware often include methods to create clusters of their nodes to load balance requests and guarantee a fault tolerant communication service. In this section I briefly recall some basic concepts about message oriented middleware technology and its architectural elements. I overview what are the most used protocols that enable communication among clients and servers. Then I overview the most used message oriented middleware implementation listing some of the most used platforms

A message oriented middleware has two types of data structures:

- **Queue:** a queue is the main abstraction in the message oriented middleware, it contains an ordered list of messages that must be dispatched to authorized message consumer clients.
- **Access control lists:** each queue has associated an access control list that specifies what clients can produce messages and what clients can consume messages for a specific queue.

A message oriented middleware has 3 core components (Figure 10):

- **Message receiver:** a message receiver is in charge of exposing an interface to let message producers send messages to the MOM. It has to check message producers' authorization to publish message on a specific queue and insert messages in that specific queue in the correct order.
- **Message dispatcher:** a message dispatcher waits for new messages on a specific queue (or set of queues) and dispatch them to interested active clients, checking a subscription list populated by the access control manager.
- **Access control manager:** an access control manager is in charge of populating access control lists and checking access control policies on queues. It exposes functions internally to let different MOM components check access control policies before enqueuing messages or dispatching messages to clients. It exposes functions externally to let clients register new producers and new receivers on a specific queue.

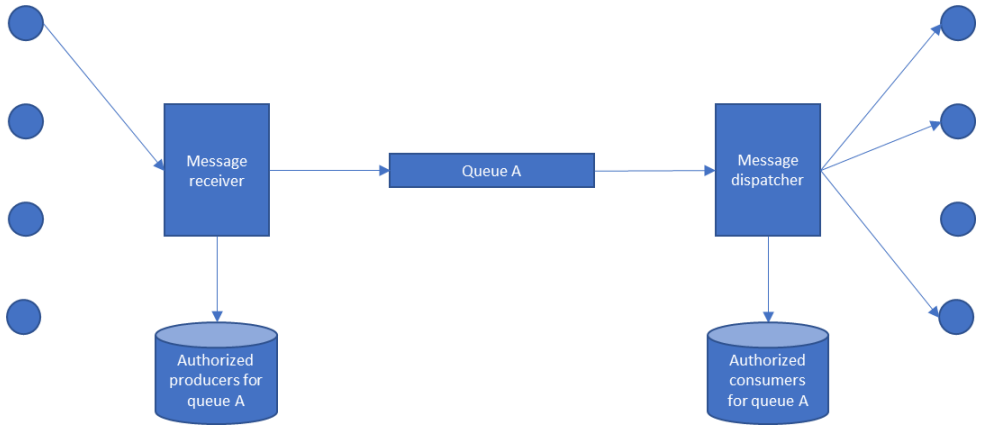


Figure 10 Message Oriented Middleware schema

Message Oriented Middleware enables communication among nodes in a distributed system implementing different patterns for message dispatching. Following the model described above, the different communication patterns can be applied to the same architecture changing the way clients are authorized to interact with a specific queue. Specifically, I can list five communication patterns:

- **Unicast or point to point**: direct communication among two clients connected to a message oriented middleware. The access control list of a unicast queue has one authorized producer and one authorized consumer .
- **Single producer Multicast**: direct communication among one producer client and a set of consumer clients. The access control list of a multicast queue has one authorized producer and one or more authorized consumers. Message dispatcher must send the message to all authorized (active) consumers.
- **Multiple producers multicast**: direct communication among a set of producer clients and a set of consumer clients. The access control list of a multicast queue has one or more authorized producers and one or more authorized consumers. Message receiver must globally order received messages. Message dispatcher must send the message to all authorized (active) consumers.
- **Anycast:** direct communication among one producer client and one consumer client randomly chosen from a list of authorized consumers. The access control list of a multicast queue has one authorized producer and one or more authorized consumers. Message dispatcher must send the message to one of the authorized (active) consumers. This communication pattern is typically used to load balance a message processing among several clients.
- **Publish Subscribe:** direct communication among a set of producer clients and a set of consumer clients. The access control list of a publish subscribe queue has one or more producers and one or more consumers. Default publish subscribe pattern does not restrict access to any of the MOM clients. It is an “open to all” multiple producers multicast. Message receiver must globally order received messages. Message dispatcher must send the message to all (active) consumers.

2.2.2.1 Platforms Analysis

- **Apache ActiveMQ Artemis:** Apache ActiveMQ Artemis is the latest version of the famous open source Message Oriented Middleware widely used as an asynchronous, multi-protocol communication medium in distributed systems. It supports a proprietary protocol called "core" along with the "old" ActiveMQ proprietary OpenWire protocol and the open source AMQP, MQTT and STOMP protocols. Thanks to its multi-protocol nature it is possible to use ActiveMQ Artemis as the message oriented middleware for a wide range of client applications without changing protocol's client libraries using protocol-specific libraries. ActiveMQ Artemis has also a core client API to exchange messages in all supported formats and a JMS 2.0 client API for Java clients. Brokers are Java based and can run on operating systems that support Java Virtual Machine version 7 or higher.

  Apache ActiveMQ Artemis supports to types of routing: **anycast** when messages are dispatched to only one consumer among the consumers subscribed to a queue and **multicast** when messages are sent to all the consumers subscribed to a queue. Queues are associated with addresses identified with a name.

  ActiveMQ can run with a single node in its simplest configuration or can scale to several nodes organized in different topologies. It supports clustering and federation of nodes as I will illustrate later in chapter 4.1.2.1.

- **Apache Kafka: Apache Kafka** is an open source distributed streaming platform written in Scala and Java with three main functionalities that enhances the capabilities of plain message oriented middleware:
  - **Message oriented middleware functionalities** with publish and subscribe functions to send and receive messages from streams of records.
  - **Storage functionalities** store messages sent on streams of records in a fault-tolerant way.
  - **Processing functionalities** to analyze streams of records.

Streaming data processing is an additional functionality that let users create stream processors using the Kafka Streams API that can consume several messages published

on different topics to analyze, transform and aggregate data republished on one or several different topics. “Streaming data includes a wide variety of data such as log files generated by customers using your mobile or web applications, ecommerce purchases, in-game player activity, information from social networks, financial trading floors, or geospatial services, and telemetry from connected devices or instrumentation in data centers. This data needs to be processed sequentially and incrementally on a record-by-record basis or over sliding time windows, and used for a wide variety of analytics including correlations, aggregations, filtering, and sampling.”[4]

Other mostly used streaming data platforms that I will not analyze in this thesis are Amazon Kinesis , Apache Flume, Apache Spark Streaming, and Apache Storm. The integration layer for railway data analysis can be categorized also as a streaming data platform.

Messages are sent by **producers** on **topics** that aggregate feeds of messages on a specific category to which **consumers** can subscribe to receive them. Each Apache Kafka sever is a **broker.**

Kafka has four core APIs:

- The **Producer API** allows an application to publish messages to one or more Kafka topics.
- The **Consumer API** allows an application to subscribe to one or more topics and process messages received on them.
- The **Streams API** allows an application to act as a stream processor, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
- The **Connector API** allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems.

[4] https://aws.amazon.com/streaming-data/?nc1=h_ls

For example, a connector to a relational database might capture every change to a table.

Kafka exposes all its functionalities over a **language independent protocol** running over TCP which has clients available in many programming languages. **However only the Java clients** are maintained as part of the **official Kafka project**, the others are available as independent open source projects. A list of non-Java clients is available here[5].

Kafka brokers can form clusters and federations. Clusters must use Apache Zookeper for distributed coordination, while federations use the external component MirrorMaker. I will analyze Kafka custers an federations in section 4.1.2.2 .

- **Eclipse Mosquitto:** Eclispe Mosquitto is a publish subscribe message oriented middleware that supports the multicast routing functionality. It implements the server-side/broker part of the MQTT protocol. Clients can connect to it using a supported MQTT client and one of the mostly used MQTT libraries (e.g. Eclispe Paho). Mosquitto does not support the clusters and runs with a single server in its default configuration. It implements a functionality called **bridge** that lets administrators connect several brokers. I will get into the details of the bridge functionality in section 4.1.2.3.
- **RabbitMQ:** RabbitMQ is an open source multi-protocol message oriented middleware written in Erlang and based upon the Open Telecom Platform framework for the clustering and failover management.
  It implements the classic producer/consumer pattern with a broker in the middle. Brokers are called exchanges and contain one or more queues that store and dispatch messages to subscribers. Publishers and subscribers can use APIs written in several programming languages (e.g. Java, Python, PHP, Ruby, C#, JavaScript, Go, Swift). For each protocol (AMQP, STOMP, MQTT) clients can use specific protocol APIs to connect to RabbitMQ's exchanges.

[5] https://cwiki.apache.org/confluence/display/KAFKA/Clients

- Administrators can configure clusters of RabbitMQ nodes and use federations to span across multiple datacenters as I will illustrate in 4.1.2.4 .

2.2.2.2 MOM Connector

The MOM Connector is an abstraction of a generic Message Oriented Middleware API.

Topics, trees and strings identify messages hierarchically inside a message oriented middleware. Publisher clients handle topics on groups. They publish messages using well defined topics and topic trees. Different message oriented middleware use different formats for topics. This project objective (and WideGroups objective) is to create a unique format to integrate different middleware technologies defining a standard format. MOMConnectorAPI is an interface for decoupling applications from the details of the underlying real implementation. I designed a standard topic format to not follow the syntax of a specific MOM.

Table 1 Topic formats

	WideGroups supported protocols	**MQTT**	**AMQP**	**ActiveMQ 5 supported protocols**	**ActiveMQ Artemis supported protocols**	**RabbitMQ supported protocols**
Topic separator	/	/	.	.	.	.
Multi-level Wildcard	#	#	#	>	#	#
Single-level Wildcard	+	+	*	*	*	*

A topic tree is a path that goes from the more generic topic to the more specific one separating each subtopic with a standard topic separator '/' (e.g.: Earth/Europe/Italy/Liguria /Genoa). The generic syntax uses

- **/ as separator;**
- *** as multilevel wildcard;**
- **+ as singlelevel wildcard**

Convenience methods are provided to translate topic syntax from the specific topic format to the generic connector topic format. In In section 5.5.7. there is a partial list

of methods provided by the MOM connector API. These methods have been selected and designed after the analysis of the MOM APIs described in the previous section.

2.2.3 Integration Layer for railway data analysis description

During the In2Rail European project I have designed and developed with Hitachi Rail STS, M3S and CIPI an entire working solution of this use case.[6] The main objective was the creation of a distributed communication platform, called Integration Layer, that manages the communication, data processing, sharing and analysis for railway services (Traffic, Asset and Energy Management Systems, field signaling infrastructure and vehicles) and clients. It is directed towards the train management systems integration with other multimodal operational systems. Therefore **cross domain data integration and replication** are key requirements for this system. The integration will use different software platforms to implement message oriented communication, data storage and analysis. One of the objectives of the integration layer project is the usage of common connectors for different platforms' typologies: message oriented middleware and key value store. I used the connectors described in section 2.2.1 and 2.2.2 to satisfy this request.

The integration layer has four main components: the message analyser, the message oriented middleware, the key value store and the client side wrapper. Users calls standard integration layer APIs through a wrapper component connected to integration layer message analyser and dispatcher. The message analyser performs three main operations for each received message: authorization, validation against a canonical data model and persistence. Integration layer components must be interchangeable. During the integration layer project I studied the main characteristics of possible external platforms that will implement the message dispatching and message storing functionalities and used these generic connectors in the Integration Layer to decouple it from the external platforms it uses.

[6] In2Rail Deliverable D08.3 contains some details of the preliminary requirements and design: http://www.in2rail.eu/Page.aspx?CAT=DELIVERABLES&IdPage=69d2e365-3355-45d4-bb3c-5d4ba797a3ac

Chapter 3

3 GROUP COMMUNICATION

Transmitting the same data to a group of processes in a distributed system and guarantee message ordering and delivery is a mature issue discussed in the literature since the eighties. It has several solutions and techniques implemented at different layers of the TCP/IP stack (data link layer, internet layer or application layer). This chapter focuses on algorithms, protocols and techniques that enable group communication. I start analyzing group communication without taking into account the "cross administrative domain" scenario. I describe the group communication issues, algorithms and technologies. An analysis of this topic must start with the description of multicast, in particular addressing routing algorithms and the way they are used in multicast protocols (network assisted and application layer). For both multicast types I briefly describe issues and solutions regarding group identification, discovery and membership management. Multicast does not guarantee that all processes connected to a multicast group receive all the messages produced on it and that they receive the messages with ordering guarantees. In the following section I analyze how to make group communication reliable with process groups and reliable group communication methodologies and toolkits. These methods extensively use message ordering algorithms for local total and causal order of messages in distributed networks and consensus groups. I briefly describe both topics and summarize distributed consensus algorithms. I conclude the chapter listing the most notable reliable group

communication toolkits. Among the described toolkits JGroups deserves a deeper analysis as it is the most frequently used toolkit in the platforms considered.

3.1 MULTICAST

Multicast is a distributed systems technique to send data from one sender to multiple receivers. There are two kinds of multicast protocols: network-assisted multicasting and application-level multicasting. In this section I describe multicast algorithms that can be applied both to network assisted and application layer multicast protocols, overview network assisted multicast protocols that implement those algorithms and show how they have been subsequently adapted in application level multicast protocols.

3.1.1 Routing algorithms for multicast

In this section I list and briefly describe the most used routing algorithms applied both in network assisted and application layer multicast. I use the generic word "node" to indicate the network element that runs the routing algorithm. This element can be a router in network assisted multicasting protocols or a server/client in application layer multicast protocols. Its neighbors are the nodes that can be reached directly without intermediates. A "network" can be the real physical/virtual network if I apply the algorithm to network assisted multicast protocols or an overlay network of clients/servers if I apply the algorithm to application layer multicast protocols. All these algorithms use solutions form graph theory to solve routing problems. A network is a graph and its nodes are the nodes of a graph. Multicast routing algorithms use either link state algorithms or distance vector routing algorithms to solve the problem depending on the network view. ***Link state routing algorithms*** need a complete view of the network topology. They use Dijikstra's algorithm for building shortest path trees and Kruskal and Prim's algorithms for Minimum Spanning Trees (see chapters 24 and 25 of [20] for details). ***Distance Vector routing algorithms*** view the network only from a node and its neighbors point of view (in network protocols they use a solution inspired by the Bellman Ford algorithm).

- **Flooding:** Flooding algorithm requires that each node in a network retransmits messages it receives to all its neighbors except the neighbor from which it received the packet. Its neighbors subsequently repeat the process transmitting the packet to their neighbors. It is simple to implement and it requires less memory than other protocols as each node does not require routing tables and keeps track only of most recently seen packets. On the other hand, it generates heavy traffic and duplicate packets. Path usage in networks is not efficient and requires distinct tables for each recently seen message causing bad memory usage.
- **Gossip/Epidemic:** It is a subclass of flooding where nodes send data to a subset of their neighbors. It is more efficient than plain flooding as it reduces messages exchanged in the network but it has still the same issues of flooding algorithms.
- **Reverse Path Forwarding**: [21] It is a subclass of flooding algorithms. A node forwards a message to all its neighbors only if the message arrived from the shortest path that connects itself to the source of the message. It has drawbacks similar to flooding algorithms
- **Usage of existing Network Spanning Trees:** It exploits a previously constructed spanning tree for the whole network and uses the internet topology that always connect two routers without loops through an active link. The multicast message is forwarded by a router to all interfaces that are part of the spanning tree except the one from which the message arrives. Easy to implement for network assisted multicast protocols. It causes heavy traffic on a subset of nodes and possible suboptimal solutions.
- **Reverse Path Broadcasting – RPB:** The reverse path broadcasting is the algorithm that *builds a specific spanning tree* for each (source, group) pair. A node forwards a message on all the other connected nodes (child) except the node from which it received the message only if the message arrives from a link that it considers the shortest path back to the source (the parent link), otherwise it discards it. It can be enhanced determining if one of the child will receive the same message directly from a parent link avoiding duplicate messages that will be discarded. It is easy to implement and provides a better network utilization

than flooding, gossip and Spanning Tree algorithms. It guarantees an optimal solution (shortest path for each multicast group). On the other hand, it forwards packet also to networks that do not contain multicast group members.

- **Truncated Reverse Path Broadcasting – TRPB:** It enhances the RPB algorithm taking into account if leaf subnetworks have nodes that belong to a multicast group. It guarantees better performances avoiding leaf subnetworks that do not belong to a multicast group. However it does not take into account removing intermediate networks that forwards data to the avoided leaf subnetworks including them in the distribution tree. Earlier version of DVMRP network assisted multicast protocol uses it.
- **Reverse Path Multicasting -RPM** It is an enhancement of RPB and TRPM. Spanning trees contain only live branches (only including branches that lead to subnetworks containing group members). It uses prune messages to remove unused subnetworks. Leaf nodes send a prune message to a parent node if they do not contain any child that is a member of a specific group (each group has a different spanning tree). If a parent node receives a prune message from all its child it stops receiving messages for that group and sends a prune message to its parent. There is a periodic update of spanning trees at regular intervals. Compared to RPBT it has better performances avoiding leaf and intermediate subnetworks that do not belong to a multicast group, but its dynamic nature requires more messages exchanged to update the tree on each node. There are also scaling issues as state information must be maintained in each router. DVMRP, PIM DENSE MODE network assisted multicast protocols use it.
- **Link state multicast routing:** When a node wants to join a multicast group it sends a LS (link state) packet to the whole network using flooding. When a node receives a message it can determine if it is part of the distribution tree and has to forward a message. In order to determine it has to use the complete map of the network is constructed using a unicast link state algorithm (Dijikstra). If it is the first message it has to calculate the complete map, otherwise it uses the cached map. Unlike RPB, TRPB and RPM it does not have to use a flood and prune approach thus increasing its efficiency. It is slower when it receives the

first message and has an higher CPU and memory usage than distance vector based algorithms (RPB, TRPB and RPM). It is used in MOSPF network assisted multicast protocol.

- **Core-Based Trees:** This algorithm [22] distinguishes two kinds of nodes: core and non-core. All nodes store a predefined distribution tree (the CBT) that involves a set of non-core nodes and a set of core nodes. Messages will be forwarded across 0 or N non-core nodes according to the CBT to reach the core nodes that will distribute the message to all group members. There is a different CBT for each multicast group. Non-core nodes send only unicast messages in the tree. Core nodes send messages to multiple destination in the tree. One of the advantages of CBT is that only core nodes maintain information about the state of each group. Another advantage is that less traffic is generated from group nodes thanks to core nodes. However there can be a possible traffic bottleneck on core nodes and using a predefined CBT can lead to a suboptimal solution. It is also necessary to develop new algorithms to support core nodes management. CBT protocol, PIM- SPARSE MODE network assisted multicast protocols use it.

Table 2 Multicast routing algorithms

Algorithm	**Type of routing**	**PROs**	**CONs**
Flooding	Not Available	Simple to implement, No routing tables, Keep track only of most recently seen packets.	Heavy traffic, Duplicate packets, Path usage in networks not efficient, Distinct tables for each recently seen message (bad memory usage)

Usage of an existing spanning tree	It depends on the method used to build the spanning tree.	Easy to implement for network assisted multicast protocols.	Only for network assisted multicast protocols. Heavy traffic on a subset of nodes. Possible suboptimal solutions.
Gossip/Epidemic	Not Available	Higher efficiency that Flooding algorithms	Still heavier traffic, possible loops
Reverse Path Broadcasting - RPB	Distance vector routing	Easy to implement. Better network utilization. Optimal solution (shortest path for each multicast group)	More suitable for NLP. It forwards packet also to networks that do not contain multicast group members.
Truncated Reverse Path Broadcasting - TRPB	Distance vector routing	Better performances avoiding leaf subnetworks that do not belong to a multicast group.	It does not take into account removing intermediate networks that forwards data to the avoided leaf subnetworks including them in the distribution tree.

Reverse Path Multicasting -RPM	Distance vector routing	Better performances avoiding leaf and intermediate subnetworks that do not belong to a multicast group.	Its dynamic nature requires more messages exchanged to update the tree on each node. Scaling issues as state information must be maintained in each router.
Reverse Path Forwarding	Distance vector routing	Similar to flooding	Similar to flooding
Link state multicast routing	Link state routing	Unlike RPB, TRPB and RPM it does not have to use a flood and prune approach.	Slower when it receives the first message. Higher CPU and memory usage

Core-Based Trees	It depends on the method used to build the spanning tree that connects the core nodes.	Only core nodes maintain information about the state of each group. Less traffic generated from group nodes thanks to core nodes.	Possible traffic bottleneck on core nodes. The predefined CBT can have a suboptimal solution. It is necessary to develop new algorithms to support core nodes management.

3.1.2 Network Assisted Multicast (NAM)

Internet Protocol Multicast [23] is an extension of the Internet Protocol that supports multicast groups. Multicast groups allow hosts of a network to send a message to multiple destinations using a special class of IP addresses: "class D" [224.0.0.0 – 239.255.255.255]. Class D has "1110" as its higher order four bits. A class D address identify uniquely a multicast group in a network. Hosts can associate to the group to receive messages and can send messages to the group. If sender and receivers belong to the same subnetwork the address is mapped to the ethernet multicast address and forwarded to it. If they belong to two different subnetworks routers must implement a multicast algorithm and a group membership protocol to transmit data among the different subnetworks.

IP multicast uses **IGMP** (Internet Group Membership protocol) [24] to handle:

- Group Join requests from network's hosts for hosts that wish to receive messages from a specific group.
- Periodic queries to check if hosts that joined a group are still active.
- Leader election of the querier in a network that contains more than one multicast router.

IGMP has three versions that has been defined in three documents: RFC1112 for its first version [23] RFC2236 for IGMPv2 [24] and RFC3376 [25] with an extension in

RFC4604 [26] for the third version. The first version defines the host membership query message format and how queries are handled by querier and host with the query protocol. Version 2 adds the election procedure of the querier in multi-router scenarios (it was left to the specific multicast routing protocol in the previous version), a new type of query message to specify the multicast group to which a router sends a query message and the leave message to let hosts explicitly leave all multicast groups. Version 3 adds the possibility to select specific sources from which receive messages and to specify which group leave in the leave message.

IGMP is used by all the protocols described below for the group membership management.

3.1.2.1 Protocols

- The **Distance Vector Multicast Routing Protocol (DVMRP)** implements the protocol Reverse Path Multicasting. It is an extension of the RIP protocol. Receivers that join a multicast group start calculating the optimal spanning tree for each possible source. The group membership is monitored using the IGMP protocol described previously in this chapter. Multicast messages are routed using the RPM algorithm. It uses poison reverse techniques to handle disconnection and graft data units techniques for handling reconnections of subnetworks after pruning.
- While the DVMRP uses the underlying RIP unicast protocol, OSPF based routers can exploit a multicast extension of their unicast routing protocol called **MOSPF**. MOSPF routers are compatible with routers running previous versions of OSPF. As DVMRP it uses IGMP for group membership management. It is based on link state algorithms (Dijikstra) used by OSPF to construct distribution trees. It exploits link state multicast routing algorithm for efficient multicast data distribution.
- **Core Based Trees (CBT)** is the protocol that implements the CBT algorithm on routers.
- **Protocol Independent Multicasting (PIM)** is an approach with two subcategories the PIM-sparse mode that uses CBT is more efficient for sparse

groups (long distances among group members) and PIM-dense mode that uses RPM it is more suitable for dense groups (short distance among members of a group and high availability). Unlike DVMRP and MOSPF they do not depend on the underlying unicast protocol implemented.

3.1.3 Application Layer Multicast (ALM)

Application-layer multicasting techniques have been introduced along with peer to peer solutions to overcome issues regarding the network assisted multicasting at the price of performance penalty compared to network layer solutions. Network layer multicasting typically requires the setup of the communication paths for information dissemination with a huge management effort. Moreover, Internet Service Providers typically do not support multicast over wide area networks making network-level multicasting techniques inapplicable for several use cases. [27] Application Layer Multicasting is deployed at end points without taking into account the underlying network infrastructure.

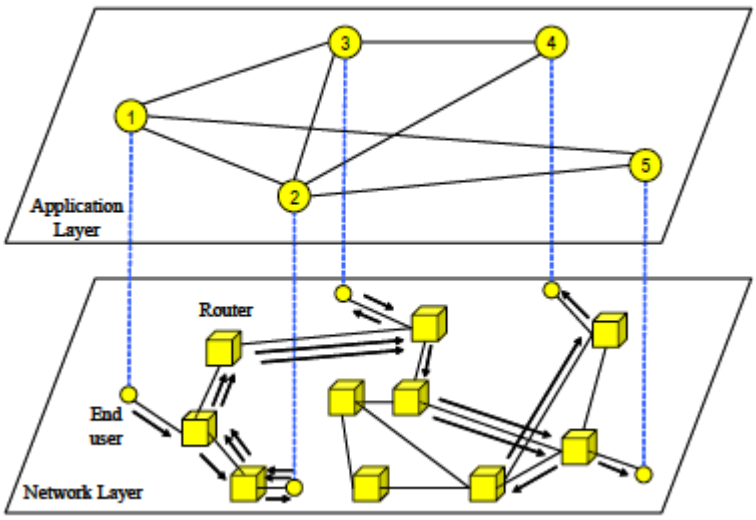


Figure 11 Overlay network of endpoints performing ALM mapped on the network layer

In [29] Hosseini et al. wrote an extensive survey of application layer multicast protocols. Hossein et al. define a series of categories that can be used both as a guideline for taking design decisions for application layer multicast protocols and to classify existing solutions. Hosseini et al. categorized protocols depending on five characteristics:

- **Application Domain**: Classification on application layer multicast protocols depending on the application domain to which they can be applied. Protocols' categories are Audio/video streaming, Audio/video conferencing, Generic multicast service and Reliable data broadcast and file transfer.
- **Deployment Level:** a multicast protocol can be deployed at
    - **Infrastructure-level or proxy based** if there are some endpoints (server nodes) that must be deployed to provide multicast service to other endpoints (multicast service clients). They act as application layer multicast routers and organize themselves in overlay networks
    - **End system level**: if the multicast protocol runs directly on peers without the necessity to deploy external servers to expose the multicast service.
- **Group Management** includes basic group management choices such as:
    - How hosts find out about multicast sessions,
    - How hosts join a session (through rendezvous points or p2p flooding)
    - How hosts leave a multicast session and
    - The possibility for the users to still contribute to multicast sessions even if they are not a part of them,
    - The nature of the users: transient and anonymous or more permanent and known.

    All these choices guide the creation of basic group management functionalities that can be **centralized** or **distributed**.

    The second part of group management functionalities are about how group nodes' are interconnected with each other and how they efficiently exchange messages. Depending on the applications' requirements nodes can be organized with a **mesh-first approach** (nodes are connected to each other in a mesh overlay network without building a distribution tree at the system startup) or a **tree-first approach** (nodes are connected immediately to a tree based overlay network, computation of the best tree depending on the requirements is performed immediately). Another efficiency

enhancement can be making the multicast group aware of **IP multicast islands**. As I said before network layer multicasting is more efficient than application layer multicasting. A smart application layer multicast algorithm can take into account the possibility to dispatch messages in part of the network exploiting the underlying network layer IP multicast for example in a trusted local area network.

- **Routing Mechanism:** routing mechanism are defined to solve a graph problem using the routing algorithms described [29] or a combination/a variation of them. In application layer multicast the algorithms are classified as:
    - Shortest Path: if the protocol builds a minimum diameter spanning tree according to some constraints and metrics.
    - Minimum Spanning Tree: if the protocol builds a tree that creates optimal paths to all its leaves with the minimum cost.
    - Clustering Structure: this group of mechanisms constructs a mesh network of nodes used to build the tree.
    - Peer to peer structure: this group needs a peer to peer solution that builds the mesh network upon which a routing tree is built.
- **Degree-constraint routing:** this characteristic regards the feasibility to solve a multicast routing problem if there are constraints in the network of nodes (e.g. nodes cannot perform multicast operations because they do not have enough bandwidth available)

3.1.3.1 Protocols

The protocols described here are a subset of the protocols described in [28] and [29]. I briefly describe only the subset of protocols belonging to the generic multicast service and reliable data broadcast / file transfer domains as they are the categories that best fit the main topic of this dissertation.

- **Borg** [30] is a protocol deployed at infrastructure level on proxy servers for generic multicast services. It uses a mesh-first approach. The mesh network is provided by a P2P substrate, Pastry. Borg builds a source specific multicast tree on Pastry for each multicast group. It uses a combination of a reverse and

forward path forwarding algorithms to build the upper part of a multicast tree and the reverse path forwarding alone for the lower part. It uses delay and bandwidth as the metrics to measure link weight.

- **BTP (Banana Tree Protocol)** [32] is a protocol deployed at end-system level directly on endpoints for reliable data broadcast and file transfer. It uses a tree-first approach. A node that wants to join the multicast network chooses a node as its parent from a bootstrap list, then it optimizes the multicast tree (that can be unique for all multicast groups or distinct for each group) using the occasional parent switching. A node changes its parent (to another a sibling node or a grandparent node using an updated list of nodes received directly from its current parent node to prevent loop creation) to reduce tree cost using delay as metric. Two siblings cannot switch to the same sibling to prevent loop creation.
- **CAN Multicast** [34] is a protocol deployed at infrastructure level on proxy servers for generic multicast services. It uses a mesh-first approach and needs a P2P substrate (CAN) to create the mesh network (see [33] for more details). It does not create a tree but an optimized flooding algorithm. The optimization rules (general forwarding rule, halfway rule, cache suppress rule, corner filter rule) are grouped in the forwarding algorithm MCAN1 [34]. MCAN1 operates on the mesh network and neighbors defined in the CAN network starting the propagation from the multicast message producer.
- **Delaunay [31]**is a protocol deployed at infrastructure level on proxy servers for generic multicast services. It uses geographical position of nodes to build an efficient mesh network for large overlay networks. It does not build a tree, but uses Delaunay triangulation and compass routing for connecting a node to its children for data distribution.
- **Grossamer and Scattercast** [36] is a protocol deployed on *proxy* servers (the Scattercast Proxies) for generic multicast services. It uses a *mesh-first* approach using the Grossamer protocol to create the best mesh network of proxies (one for each multicast group). On top of these mesh Scattercast networks multicast trees are created from each source to destinations using a *distance vector*

protocol (such application layer variations of DVMRP running the RPM algorithm using delay and bandwidth as metrics). *Periodic improvements* to the mesh network performances are run using one proxy elected to send refresh messages to the mesh network. A reliable version of Scattercast called RMX is described in [38]

- **Overcast [35]** is a protocol deployed at infrastructure level on proxy servers for reliable data broadcast and file transfer. It uses a tree-first approach and uses bandwidth as the metric to optimize the single source distribution tree. Each newcomer must send a join request to the root of the tree that becomes the potential parent node. Then an iterative process starts to place the new node as far from the root as possible without reducing the bandwidth. To do that the node estimates what is the bandwidth among itself and the potential parents using the download time of 10 Kb of data from the root and all its children. If one of the children has higher bandwidth it chooses it as the new parent and iterates the process. It stops when it cannot fine nodes' connections with an higher bandwidth. If more than one node can become the parent it always chooses the closest in terms of network hops. Periodically health checks are performed to avoid dead nodes and optimize the tree.
- **Scribe** [37] is a protocol deployed at infrastructure level on proxy servers for generic multicast services. It needs Pastry, a P2P substrate, as the mesh network on which it builds a multicast tree using reverse path forwarding for each multicast group combining Pastry paths from each group member to the tree root. Messages produced by members of a multicast group will always be sent to the root node of the multicast group and then forwarded to the multicast tree built using a reverse path forwarding algorithm and delay and bandwidth as metrics.
- **TBCP (Overlay tree building control protocol)** [39] is a protocol deployed at end-system level directly on endpoints for generic multicast services. The main sender node of a multicast group is designed to be the root. The root inserts newcomers in the multicast tree using round trip time as the metric to measure

the distance from their parent and calculating all possible solutions. Each multicast group has a unique tree.

- **Yoid (Your Own Internet Distribution):** Yoid [40] is a protocol deployed at end-system level on endpoints (but uses rendezvous points that keep a list of group members when a newcomer performs group join operation) for generic multicast services. It is a tree-first protocol that creates a unique shared tree for all the multicast groups. It uses also a mesh topology to control the network and recover from tree partitions. Tree creation is simple, but requires several refinements to avoid loops and performance issues. A node randomly selects its parent from the list given to it from one of the rendezvous points. The selected node can refuse to become a parent if it contains the requiring node in its root list. This mechanism could still create loops that is why Francis et al. proposed the *switchstamp* additional information in the root path as described in [40]. As described in [40] two more refinements runs in parallel to solve loss-rate and latency issues.

3.1.4 Multicast routing algorithms usage in NAM and ALM protocols

In the following table I include a subset of the analyzed multicast routing algorithms that can be found in the analyzed routing protocols.

Algorithm	**NAM Protocols**	**ALM Protocols**
Flooding	-Not Available	CAN Multicast (optimized flooding according to MCAN1 rules)
Reverse Path Forwarding	Not Available	Borg[7], Scribe
Truncated Reverse Path Broadcasting - TRPB	Earlier versions of DVMRP	Not Available
Reverse Path Multicasting -RPM	DVMRP, PIM DENSE MODE	Scattercast and Gossamer
Link state multicast routing	MOSPF	Not Available
Core-Based Trees	CBT protocol, PIM-SPARSE MODE	Not Available

[7] in combination with forward path forwarding for the upper part of the multicast tree

3.2 RELIABLE GROUP COMMUNICATION

IP multicast and application layer multicast services provide group communication without any guarantees in their default model. A reliable group communication system add reliability to a basic multicast solution. Basically reliability means that all participants in a group will receive all the messages sent to that group in a specific order. Therefore a reliable group communication system must solve issues regarding:

- **Message Ordering:** Correctly deliver all the messages in the same order (or following a certain policy) to all processes in a group.
- **Reliable participants' connections and disconnections:** Keep a consistent view of the system and message distribution guarantees while handling dynamic groups where participants can join and leave during system operation. This part is fundamental to determine who are the participants that must receive a message.
- **Up to date group Information, Discovery and Membership:** Manage group information and group membership in a consistent and fault tolerant way.

Some of the protocols described in section 3.1.3.1 already implemented reliable group communication in their default configuration, but a "multicast protocol" usually does not need to include reliability.

3.2.1 Message Ordering

There are four different message ordering possibilities:

1. **Unordered multicast**: no ordering guarantees for messages sent to a multicast group.
2. **FIFO (first in first out) ordered multicast**: it guarantees that messages will be delivered to group participants following the same order of the sender that produced them. There are only guarantees about the ordering of messages from the producer and not a global order of messages in the group.
3. **Causally Ordered Multicast**: it preserves causality among messages. Independent messages can be delivered in different orders on different

participants. Related messages must be delivered with the same order to all the nodes in a group. Vector Clocks [10][11] provide a solution to this problem.

4. **Totally Ordered Multicast:** it requires that all the nodes in a group will deliver messages to participants exactly in the same order.

To decide the correct order of messages there must be a method to tag messages with the correct time. There are two types of clocks: physical and logical clocks. Physical clocks can be prone to errors and must be synchronized using a completely distributed network of time servers, like it happens with NTP (network time protocol, http://www.ntp.org), or with a central master time server. Precision in physical clocks usage cannot be always accurate and requires an heavy overhead of communication to maintain clocks synchronized. A solution comes from Lamport and logical clocks. Lamport logical clocks [10] are a good choice to guarantee FIFO order, but they cannot be exploited to order messages globally. Vector clocks [10][11] can order events globally using the happened before relation. The happened before relation of vector clocks detects concurrency coarsely and there can be cases in which two messages can be considered concurrent without the possibility to decide what message comes first. IN these cases there must be a unique rule to decide which message comes first (based on the IP address of the sender, on which node joined the multicast group before, etc.) These decisions can be taken in a distributed way or in a centralized way with the election of a "sequencer" or "leader" that will decide the correct order based on logical or physical timestamps.

3.2.2 Process groups and reliable group communication toolkits

Process groups concepts date back to the eighties and were one of the first attempts to create fault tolerant distributed applications. Some examples of gro They provide a basic fault tolerance technique to applications organizing identical processes into groups and delivering the same message to all group members. This method makes the process groups resilient to fails of single processes. If one process fails another process in the group can take over for it starting from a replicated state. One of the process group computing style pioneers, Kenneth Birman, recently adopted the **process groups and reliable group communication** techniques in his new project Derecho [40], but

he noted that "Process group software libraries fell into disuse as cloud computing emerged, reflecting perceived scalability and performance issues". One of the reasons is that before cloud computing diffusion these libraries were frequently used for interconnecting applications to form reliable clusters. Applications developers had to use them to deploy reliable and fault tolerant applications. With the advent of cloud computing the usage of these libraries decreased as applications can be deployed in cloud platforms that provide fault tolerance as a service. However, the platforms where developers deploy applications still use these libraries and concepts to create fault tolerant clusters of platform's nodes in cloud environments or in datacenters. As these libraries and concepts are pretty old they are implemented in solid frameworks and solutions and the academic interest decreased after the first decade of the 2000s. My thesis will use these concepts and methodologies extensively. One of the objective of this thesis is the cross domain collaboration with a decentralized platform federation. Cloud computing and virtualization techniques would ensure better performance for fault tolerance in applications' clusters, but they are limited to clusters located in the same datacenter or at least to sites administered by the same owner or company. Software platforms federations techniques are designed for cross site and cross administrative domain platforms' distribution and must run in wide area networks connecting different datacenters. Typically they use group communication toolkits such as JGroups to coordinate federation of nodes, or distributed group coordination and service discovery toolkits like Apache Zookeeper[8], Netflix Eureka[9] or Consul[10], etcd[11], etc.. Another issues in distributed data storage system is to guarantee the consistency of shared information among a group of processes. Distributed and parallel programming research have studied consistency defining several consistency models and consistency protocols. A consistency model defines what inconsistencies can be tolerated by a system to be defined in a correct state. The "inconsistency level" can be measured using the "**continuous consistency"** framework from Yu and Vahdat []. There are two types of consistency models: data centric and client centric. Data centric

[8] https://zookeeper.apache.org/
[9] https://github.com/Netflix/eureka
[10] https://www.consul.io/
[11] https://etcd.io/

consistency models focus on systemwide consistency, that is the consistency of data inside a system. In a system where clients can connect to different system's nodes containing replicas of the same data item client centric consistency models regards the study of client-side consistency for a client that can connect to different system's nodes over time and need to access to a consistent replica of the same data item even if it is stored in different locations. Depending on the requirements you can choose from strict consistency or strong consistency to eventual consistency and sequential consistency. Most used consistency protocols are those that follow a simple model. "As soon as consistency models become slightly difficult to understand for application developers, we see that they are ignored even if performance could be improved." []

3.2.2.1 Focus on JGroups group communication toolkit

Kenneth Birman's research group developed JGroups [42], a reliable group communication toolkit for Java applications. It is widely used in software platforms for clustering and federation of their nodes as I will illustrate in section 4.1. JGroups implements a reliable group communication protocol for multicast communication. JGroups was initially intended as a Java library to support network layer multicasting, then it added reliability to the network-layer multicast becoming a reliable group communication toolkit and finally evolved to support a more complex protocol stack implementing multicast at application level to decouple applications to the underlying network layer services.

The main JGroups features include[12]:

- Cluster creation and deletion. Cluster nodes can be spread across LANs or WANs
- Joining and leaving of clusters
- Membership detection and notification about joined/left/crashed cluster nodes
- Detection and removal of crashed nodes
- Sending and receiving of node-to-cluster messages (point-to-multipoint)

[12] Source: http://jgroups.org/

- Sending and receiving of node-to-node messages (point-to-point)

The main JGroups' characteristic is its flexible protocol stack (Figure 12). JGroups' users can choose which protocols best suit the distributed applications' requirements such as message delivery guarantees, message ordering, failure detection of group members and which protocols are supported by the network in which they will run their software.

JGroups provides support to several protocols to implement different functionalities. Developers have to declare what protocols want to use for each functionality in the JGroups stack using an xml file associate to a multicast group:

- **Transport:** developers have to define with transport tag what protocols (UDP, TCP, tunnel) want to use for sending and transporting messages among nodes communicating with JGroups.
- **Initial membership discovery**: The task of the discovery is to find an initial membership, which is used to determine the current coordinator. Once a coordinator is found, the joiner sends a JOIN request to the coordinator.
- **Fragmentation:** protocols for fragmentation and reconstruction of large messages.
- **Ordering protocols**: FIFO, Total Order
- **Merging after a network partition**: protocols to handle merge of data in a cluster after a split brain scenario.
- **Failure Detection**: The task of failure detection is to probe members of a group and see whether they are alive. When a member is suspected (= deemed dead), then a SUSPECT message is sent to all nodes of the cluster. It is not the task of the failure detection layer to exclude a crashed member (this is done by the group membership protocol, GMS), but simply to notify everyone that a node in the cluster is suspected of having crashed. The SUSPECT message is handled by the GMS protocol of the current coordinator only; all other members ignore it.
- **Reliable message transmission**: Reliable unicast and multicast message transmission. Lost messages are retransmitted

- **Message stability**: protocols to guarantee storage of messages not already received by all cluster members and automatic removal after acknowledgment of cluster wide reception.
- **Group Membership**: protocols for managing joins of new nodes in a cluster, failure detection (crashed nodes are excluded from the membership) verification and handling of SUSPECT messages.
- **Flow control**: Flow control to prevent slow receivers to get overrun by fast senders.
- **State Transfer**: cluster state transfer protocols.
- **Statistics**: how to expose statistics (number of received multicast and unicast messages, number of bytes sent etc)
- **Security**: message encryption protocols and access control protocols for joining cluster communication.
- **Compression**: message compression protocols.

Here[13] you can find a complete list of JGroups protocols. JGroups works both with TCP and UDP.

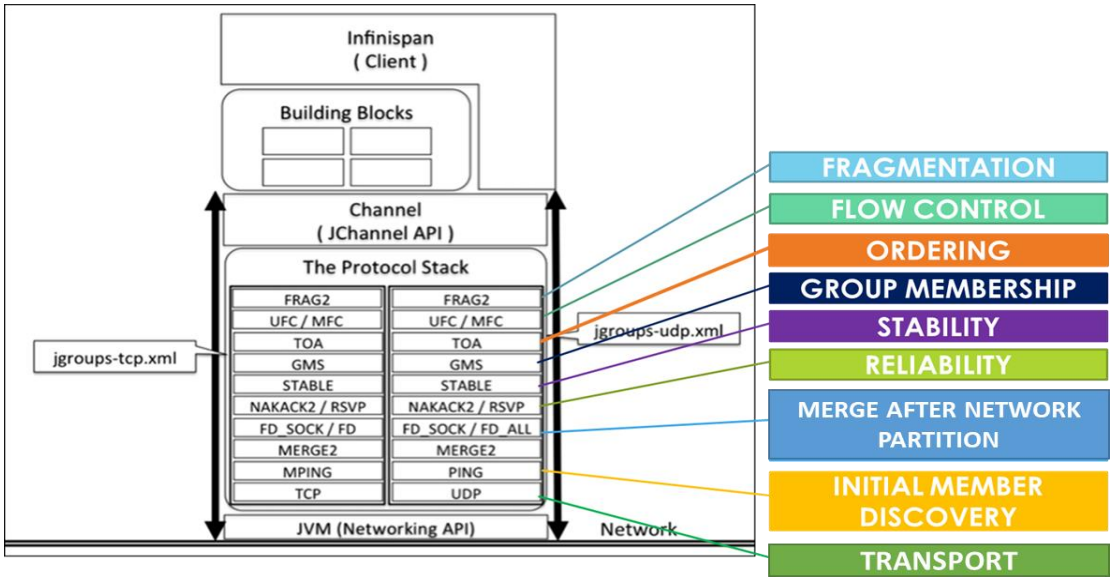


Figure 12 JGroups stack

[13]List of JGroups protocols: http://www.jgroups.org/manual/html/protlist.html

Chapter 4

4 Cross Administrative Domain Group Communication

Standard definition of ***administrative domains*** comes from the networking world as "A collection of End Systems, Intermediate Systems, and subnetworks operated by a single organization or administrative authority." Components in an administrative domain trust each other, while they do not trust components in other administrative domains with which they can interoperate in a "mutually suspicious manner". Technologies that enable data and message sharing among systems located in different administrative domains can require direct access to resources and nodes in a peer to peer fashion. This solution is typically not possible for security and management reasons, applications could be behind a firewall and direct access can open security breaches. Moreover direct communications among applications will increase the computational load on each node that could not have the necessary resources to handle it, cause availability problems and increase network traffic for peer to peer communication. Another solution is to mediate communication through

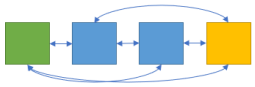


Figure 14 Peer to peer data sharing

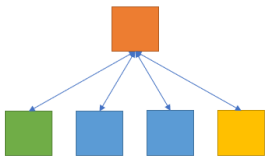


Figure 13 Data sharing through an external node

proxy nodes that can be managed centrally by a third party or be distributed inside enterprise or department boundaries to maintain control over the data location and their usage. The third solution best fit the use cases and issues described in this thesis.

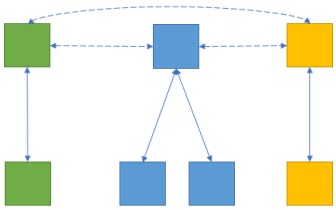


Figure 15 Data sharing through a federation of proxy servers

Currently available methods have the drawbacks that I describe in the introduction to software platform federation technologies section of this chapter. In order to solve them I designed the WideGroups solution. The ideas behind WideGroups solutions are inspired by two popular protocols and technologies: XMPP and SMTP.

I include a brief description of XMPP and SMTP protocols after focusing on the DNS usage and some common concepts they share that will be exploited for WideGroups. They both provide federations of servers owned by different domains and implement a completely decentralized data sharing protocol. Group of nodes are created client side and without server configuration modification.

4.1 Software Platforms Federation

Scaling a platform to multiple administrative domains is still an open issue especially when stringent data privacy requirements cannot be implemented by cloud-solutions alone, but demand decentralization. Software platforms federation can be an efficient solution, but current implementations have some drawbacks that the solution described in this thesis solves. In this chapter I include examples from platforms analysed for the integration layer project with Hitachi Rail STS. I conclude the section with a paragraph describing issues and strengths of the analysed implementations. First of all software platforms of the same type (message oriented middleware, distributed key value store, …) but from different vendors cannot be federated, because they implement platform specific federation technologies. Different administrative domains can choose different vendors for the same software platform type, they should agree on the platform vendor before starting any federation and this could lead to management issues when a large number of administrative domains is involved. Supposing that all domains choose the

same platform vendor there still could be problems regarding the platform version chosen that can be not compliant with the one used in the external domains. Even if each domain agrees on the same platform vendor and on the same version there are still issues regarding platform nodes' configuration to enable federation. Federation is typically setup server side (applications' developers and users must involve system administrators to configure cross domain connections) and statically (users need to deeply know what group of client applications will share data before starting up the software platform). Moreover, federating software platforms is usually perceived as a loss of control about what data platforms can receive from external nodes and it is often discouraged to prevent security and denial of service attacks.

In section 4.1.1 I overview the federation methods available for the two In Memory Data Grids analysed in section 2.2.1. In section 4.1.2 I briefly analyse methods for federating message oriented middleware brokers analysed in 2.2.2 .

4.1.1 Key Value Store federation

4.1.1.1 Infinispan

Infinispan enables cache federation through the cross site replication functionality. The cross site functionality can be exploited for cross administrative domain replication. Infinispan uses JGroups for group communication and the REALY2 protocol for remote cluster data backup. This solution enables mirroring of a local cluster entries on a set of remote clusters located in geographically separated sites. It requires some effort in local and remote clusters configuration: administrators need to touch the configuration files on both the local and the remote sites accordingly. Default configuration mirrors data in read only mode from a source site to a list of destination sites. The source site administrator must associate a cache to a list of remote clusters' caches, the destination sites administrators configure one of their caches to receive data from the remote site (caches **must have the same name** or there must be an explicit association of one of the caches to the name of the remote cache they want to mirror using the "backup-for" attribute). Different clusters can use different replication solutions for entries inside their clusters and are completely separated. Each edit on an entry in the remote cache would be

mirrored on the remote site, this will not happen in the other direction: edits on a destination site are not replicated on the source site. To avoid inconsistencies, destination sites administrators must setup read only permissions on mirrored cache entries. It is possible to setup cross site replication also with additional settings regarding:

- strategy: SYNC for a synchronous backup of data, ASYNC for asynchronous data backup (default)
- failure-policy: it specifies what policy nodes have to follow in case of failures during the backup operations
    - IGNORE: let the local operation/transaction terminate without considering the error.
    - WARN: let the local operation/transaction terminate throwing a warning message to the local node. (default)
    - FAIL: it works only with the SYNC strategy. The backup operation/transaction fails throwing an exception.
    - CUSTOM: it is possible to implement a user definable behavior in case of failures using the failurePolicyClass.
- FailurePolicyClass: in case of custom failure policies implementation administrator has to indicate what is the class implementing the org.infinispan.xsite.CustomFailurePolicy interface.
- Timeout: remote backup timeout in milliseconds (default 10000 ms)

Figure 16 shows an example of a system using the cross site replication functionality. It represents three different datacenters that share the “users” cache to replicate edits on the three remote sites. Internal cluster communication uses a local JGroups configuration while remote replication and communication exploits the **JGroups’ protocol RELAY2**.

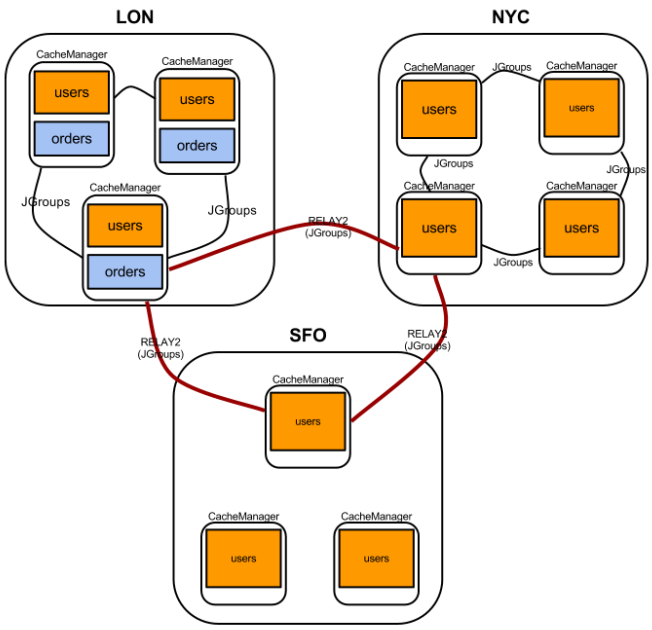


Figure 16 Infinispan cross site replication example

4.1.1.2 Hazelcast

Hazelcast supports two types of key value store federation with the WAN replication feature [14]: active-passive and active-active. Active-passive federation is used for failover scenarios when an active cluster backups its data to one or more passive clusters. Active-Active federation is used to let more than one cluster to be active. Clusters use REST calls and a proprietary protocol to replicate data across different clusters. Discovery of clusters can be done statically through a list of IPs or using a discovery service provider interface (Discovery SPI), a discovery API that can be implemented using various service discovery providers such as Network Layer Multicast, Eureka, Aws cloud, Azure cloud, Zookeeper and others.

4.1.2 Message Oriented Middleware broker federation

4.1.2.1 Apache ActiveMQ

A federation links a source ActiveMQ broker to an external broker located in a different administrative domain. Messages coming from the source are forwarded from the

[14] The WAN replication feature is available only in Hazelcast Enterprise. In the open source version WAN replication is enabled only for the Map datastructure.

receiving broker to local clients. The federation allows to share messages among different brokers without the creation of a unique cluster and it is particularly useful to send messages among remote clusters in a WAN. Brokers can be in different administrative domains and with different versions, configurations or users. The federation can take place between clusters towards others.

At the configuration level it is possible to setup some filter to decide which messages are replicated and with which policies.

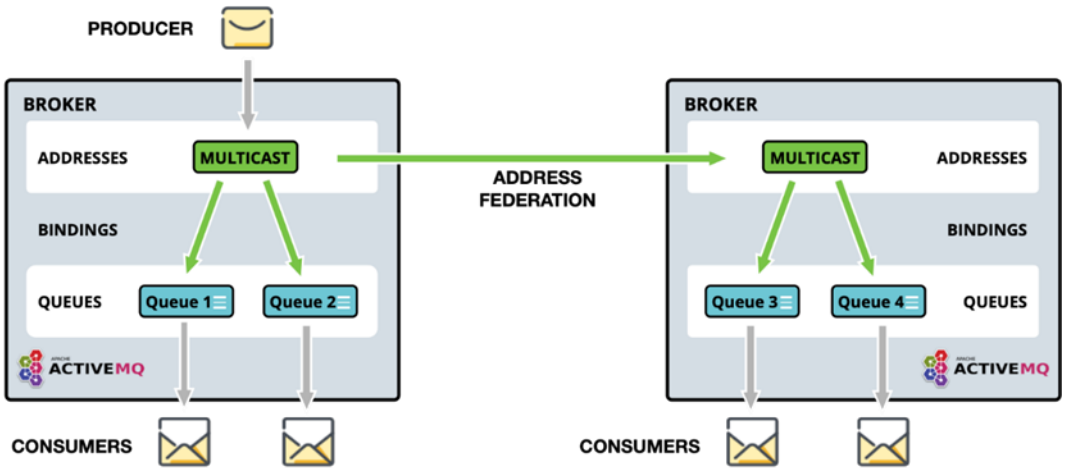


Figure 17 Apache ActiveMQ address asymmetric federation

Address federation is similar to a multicast: a broker connects to another, as a consumer, and it will receive messages about one or more topics. When a producer creates a message, the first broker distributes it to its own consumers including the second broker, that consequently distributes them to its own consumers.

Federations can be

- Asymmetric: messages are replicated to a remote cluster for failover scenarios
- Symmetric: messages are exchanged among the two brokers in both directions

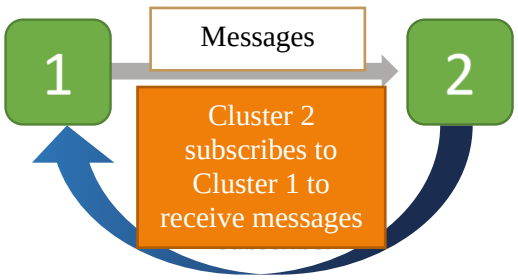


Figure 18 Federation in ActiveMQ

4.1.2.2 Apache Kafka

Kafka runs as a cluster on one or more servers. Under the hood Kafka uses Apache for managing, coordinating Kafka brokers in a cluster. Different clusters can be federated using an external tool called **MirrorMaker.** MirrorMaker is a tool distributed with kafka, suitable to reply messages between two different clusters, in a way which can be similar to a federation. MirrorMaker copies messages from a source cluster, as consumer, and then it publishes them to another cluster, this time as a producer, with the ability to filter topics based on a pattern.

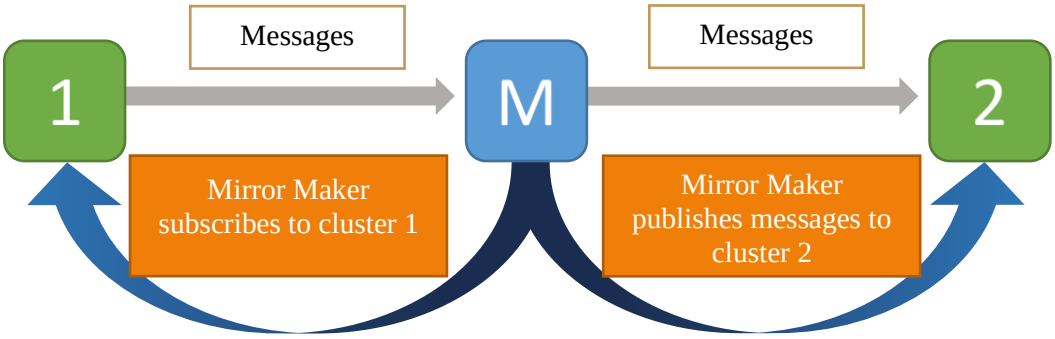


Figure 19 Apache Kafka Mirror Maker

4.1.2.3 Eclipse Mosquitto

Mosquitto does not allow creating clusters, therefore domains are always made up of single nodes. Through a bridge, domain 2 connects to domain 1 as a publisher and / or subscriber. Domain 2 can forward some messages to Domain 1; domain 1 can forward other messages to domain 2.

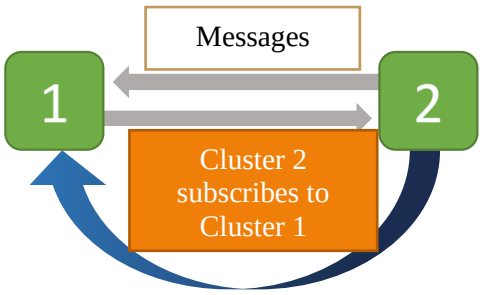


Figure 20 Mosquitto bridge

4.1.2.4 RabbitMQ

RabbitMQ federation is similar to Apache ActiveMQ federation. You can setup asymmetric or symmetric federations among different clusters through upstream links that connects two or more "exchanges" nodes.

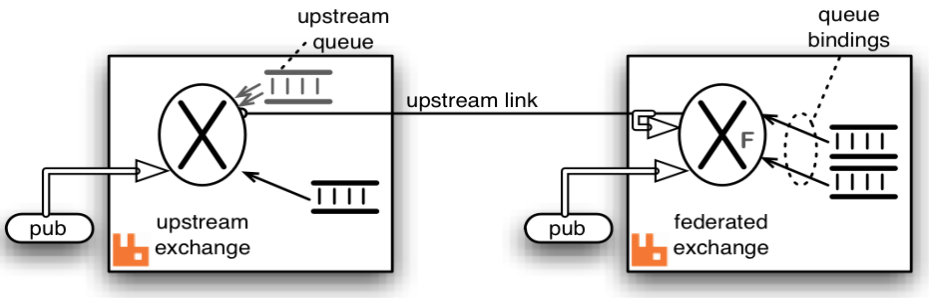


Figure 21 Asymmetric federation in RabbitMQ

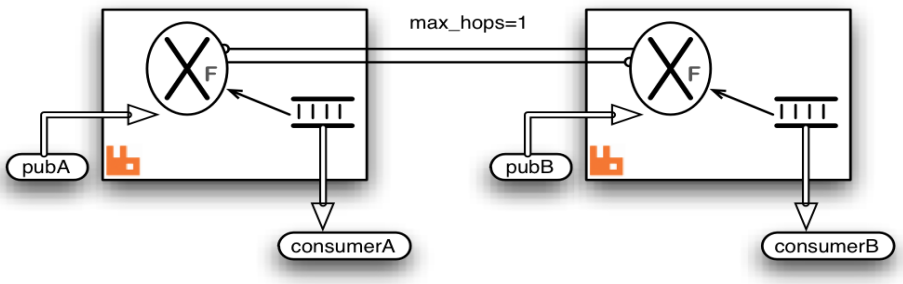


Figure 22 Symmetric federation in RabbitMQ

4.2 DECENTRALIZED PROTOCOLS TO EXCHANGE MESSAGES AMONG DIFFERENT ADMINISTRATIVE DOMAINS

The federation techniques described in section 4.1 requires the same platform running in different administrative domains with configuration setup accordingly to enable cross site replication of data. My approach wants to decouple as much as possible the separated administrative domains to reduce interactions among system administrators and the necessity to touch configuration files to make the federation possible. My research explored existing methods to do that. SMTP and XMPP protocols are good examples of this approach. They are open protocols that enable cross domain communication among servers that setup ad hoc connections without a priori knowledge of the network they will create to make data exchanges possible. Both methods use DNS to discover external domains and establish connections. I start this section showing how DNS can be used to setup decentralized connections among servers and how the two protocols (SMTP and XMPP exploit it).

4.2.1 DNS usage for domain discovery in decentralized protocols

DNS standard has been designed to provide several information about an address, not only its IP. These information are stored in the **resource records (RR).** Main resource records are defined in RFC 1035, additional records has been defined in subsequent documents (e.g. SRV is described in RFC 2782). Each resource record has a different meaning, for example:

- **A** (Address): it represents the binding between a name and its IPv4 address.
- **AAAA**: it represents the binding between a name and its IPv6 address.
- **CNAME:** the canonical name record provides information regarding aliases that can be exploited to bind different services on a host to different aliases.

- **MX:** Mail exchange record are standard SMTP records used to address e-mail servers where to forward e-mails for a specific domain.
- **SRV:** Service records identify the server that provides a service inside a domain.
- **TXT:** text field for domain's additional information.

Other records are listed in the official RFC documents[15].

4.2.1.1 The SRV Resource Record

Protocols such as SMTP, DHCP, HIP or SPF have standard resource records registered on DNS. For newer protocols DNS standard introduced a generalized service location record, the SRV. Here is the format of the SRV RR[16]:

```
_Service._Proto.Name TTL Class SRV Priority Weight Port Target
```

Listing 1 DNS SRV record

Where:

- **Service** is the symbolic name of the desired service, as defined in the RFC 1700 document or locally.
- **Proto** is the symbolic name of the desired protocol (e.g. _TCP and _UDP).
- **Name** is the domain name this Resource Record refers to.
- **TTL** specifies the time interval (in seconds) that the resource record may be cached before it should be discarded.
- **Class** is a strong hint to the protocol family which should be used to communicate. SRV records occur in the IN (Internet) class.
- **Priority** indicates the priority of this target host. A client MUST attempt to contact the target host with the lowest-numbered priority it can reach; target

[15] https://tools.ietf.org/html/rfc2782
[16] This section contains part of the official RFC 2782 document.

hosts with the same priority SHOULD be tried in an order defined by the weight field. The range is 0-65535.

- **Weight** is a server selection mechanism. The weight field specifies a relative weight for entries with the same priority. Larger weights SHOULD be given a proportionately higher probability of being selected. (Weight 0 when there isn't any server selection to do).
- **Port** is the TCP or UDP port on which the service is listening. The range is 0-65535.
- **Target** is the canonical hostname of the machine providing the service. It always ends with a dot. It must correspond to an address record (type A or AAAA), the name must not be an alias.

For more details I recommend the reading of the standard RFC documents RFC 1035 and RFC 2782.

An example of a SRV record for the XMPP protocol is:

```
_xmpp-server._tcp.example.com. 18000 IN SRV 0 5 5269 xmpp.example.com.
```

Listing 2 SRV record for the XMPP protocol

In Listing 2 the SRV record points to a server named xmpp.example.com listening on TCP port 5269 for XMPP services.

Servers' addresses are of the form <administrative_domain_ID> (e.g., unige.it), while clients registered on an administrative domain have addresses like <client_ID@administrative_domain_ID> (e.g., giancarlo@unige.it).

4.2.2 SMTP

SMTP (Simple Mail Transfer Protocol) [51] is an open standard that includes the core protocols used to transfer email messages among computers. A client connects to its own server called MTA (mail transfer agent) to send an email to a set of recipients. Each recipient is identified by a unique e-mail address <client_ID@administrative_domain_ID>. The mail transfer agent will contact the DNS to perform the discovery operations of the receiving MTA and establish a

connection to transfer the message. MTA works as proxy server of the decentralized network and must be registered on the DNS server for discovery. “Federation” are setup at runtime and MTA are loosely coupled and contacted only when necessary without a a-priori setup.

4.2.3 XMPP

XMPP (Extensible Messaging and Presence Protocol)[46][47][48] is an open standard that includes a set of protocols and specifications to implement a decentralized middleware for instant messaging. Its decentralized nature makes XMPP very similar to SMTP. Instead of MTAs XMPP uses Jabber servers that exchange a stream of xml stanzas according to the XMPP protocol. Each recipient is identified by a unique JabberID <client_ID@administrative_domain_ID>. As in SMTP, the Jabber server will contact the DNS to perform the discovery operations of the receiving Jabber server/servers and establish a connection to transfer the message. Unlike SMTP where connection are opened and closed after the mail transfer succeeded, XMPP establishes a TCP binary connection that is kept alive until the end of the XML stream. Unlike platform federations where nodes must be from the same vendor, here nodes must implement the same protocol version, but can be different implementations from different vendors. Connection setup is performed at runtime without touching a static configuration file. My approach is inspired by XMPP, but it is oriented to be a general purpose message oriented middleware. XMPP was born as a text protocol for instant messaging and was intended for human to human interactions. Several efforts to make it a machine to machine protocol suitable for IoT applications are reported in [45]. My approach is natively machine to machine.

Chapter 5

5 WIDEGROUPS FRAMEWORK

Wide Area Message Groups (hereon WideGroups) is a large scale administrative scalable group communication framework used to exchange messages among applications belonging to groups that spans multiple separate administrative domains. It supports two types of messages: ***basic messages*** and ***platform messages***. Basic messages are simple messages exchanged among client applications without external platforms' usage. Platform messages integrate external platforms for additional functionalities such as publish/subscribe and storage. Messages have a standard format called WGMessage. Compared to state of the art technologies, WideGroups provides a new method to handle platform nodes' federation with a programmatic approach. It does not require an admin to touch every node's configuration to setup peers' networks to exchange messages with other administrative domains. WideGroups provides an infrastructure-level ALM protocol and its implementation in the WideGroups framework. I follow the categories and the nomenclature Hossein et al. provided in [29] to describe the WideGroups framework in the following sections. WideGroups handles federations with an abstraction called **group**. A group is a set of actors connected across different administrative domains. Actors are other WideGroups nodes and clients that have access to a globally ordered message queue where clients can write and read messages. Depending on what clients can write messages in a group, groups can be asymmetric or symmetric. WideGroups has two types of functionalities: the control

plane functions and the data plane functions. The control plane services include all the functionalities to manage groups (**create group, delete group, edit group settings**). The data plane services include four functionalities: **sendMessage** (asynchronous function to publish messages to a specific group using a description)**, subscribe** (asynchronous function to subscribe to messages published to a group), **get** (function to get synchronously the latest message published to a group) and **receive** (synchronous function to wait for the next message on a group)**.** WideGroups has 3 architectural components that implements the functions described above:

1. **WideGroups Server**: the WideGroups server is the access point to the WideGroups network. It exposes an interface to let clients connect to create groups, exchange basic messages on groups, use groups to federate platforms across a cluster. It has a modular architecture with seven main modules: **Group Controller, Authorization Module, Discovery Module, Consistency Module, WGServer communication module, WGClient communication module and Local platform connector.**
2. **External components:** it includes **local platforms** and **nodes registry**. Local platforms enhance WideGroups functionalities with additional features: topic based publish subscribe inside a group through a federation of message oriented middleware, data replication among a group of storage platforms through the key value stores. Nodes registry is necessary to perform discovery operations for cross administrative domain communication.
3. **WideGroups Client:** clients connect to the WideGroups to publish messages on WideGroups groups and subscribe to messages published on those groups.

As described in chapter 3, in [29] Hosseini et al. wrote an extensive survey of application layer multicast protocols. Hossein et al. define a series of categories that can be used both as a guideline for taking design decisions for application layer multicast protocols and to classify existing solutions. WideGroups servers run an application layer multicast protocol. They are oriented to generic multicast **application domain**. WideGroups is deployed at **infrastructure-level or proxy based.** WideGroups servers are the endpoints (server nodes) that must be deployed to provide

multicast service to other endpoints (WideGroups clients). They act as application layer multicast routers and organize themselves in overlay networks. Group Management includes well known users that use control plane functionalities to find out about multicast sessions. Clients are well known and identified through a PKI and certificates. They can join groups (that provide multicast sessions) through the WideGroups servers that act as rendezvous points (see section 5.5.2 for the discovery module). Discovery techniques are similar to the ones described in chapter 4.2. They can leave a group using control plane functionalities as described in chapters 5.4.2 and 5.5.1. Group management is distributed on WideGroups servers. WideGroups servers are interconnected using a tree constructed using minimum ping time if the groups exceeds the five servers. Otherwise they use a flooding approach with messages sent in parallel to all group servers. The current version of the prototype and the results showed in section 6 run only the flooding version of the WideGroups multicast protocol.

5.1 FRAMEWORK PROTOTYPE DETAILS, TECHNOLOGIES AND MOTIVATIONS

In this chapter I will describe the WideGroups framework as a generic model that can be implemented with different programming languages. I developed a prototype in Java (Java version 11) with CIPI support (Joint Research center in computing platforms) both for clients and servers. Communication among clients and servers is based on the HTTP2 protocol. I chose Java for speeding up server deployment on different servers' operating systems thanks to the wide adoption of Java Virtual machines. Moreover, as this framework is enterprise systems' oriented, Java is a widely accepted programming language and enterprise systems already support it. These advantages overcome performance disadvantages comparing Java to compiled languages such as C++ or GoLang.

5.1.1 Protocol Buffers

I implemented the prototype using protocol buffers[17] as the message format for the data exchanged among WideGroups clients and servers. Protocol buffers serialize messages to bytearrays and offer a better compression than JSON and XML reducing exchanged messages size. They are efficient for remote procedure calls. Developers can define protocol buffers using .proto files that define types that will be exchanged on the wire using combination of generic types. Types can be generic or user-defined through "message" keywords. Protocol buffers files are processed from proto serializers and deserializers to generate translator from proto to bytes. Serializers are available in several languages[18].

5.1.2 HTTP/2

HTTP2 [49]is generally faster than previous versions of HTTP. It adds to old HTTP these features:

- Request Reply headers for remote procedure calls
- Binary format for headers and payloads to increase speed and efficiency
- Possibility to use a single connection for multiple requests and responses.
- Usage of streams.

5.1.3 gRPC

GRPC is a remote procedure call language that exploits HTTP/2 and protocol buffers feature to define a new type of service oriented architecture. Developers can add inside ".proto" files the "rpc" keyword to define remote procedure calls. The proto compiler will automatically create client and server interfaces for the chosen programming language as it happens with wsdl (web service definition language) in SOAP web services. Protocol buffers support classic request reply pattern plus streaming pattern that let developers use HTTP/2 new functionalities. At some extent gRPC, thanks to HTTP/2, maps better the underlying TCP so cketsadding more reliability, speed and efficiency. Moreover with the support of several programming languages[19] it makes

[17] https://developers.google.com/protocol-buffers
[18] https://developers.google.com/protocol-buffers/docs/tutorials
[19] https://grpc.io/

easier to write client server protocols using protocol buffers and implement it using gRPCs.

5.2 IDENTITIES AND GLOBAL ADDRESSES

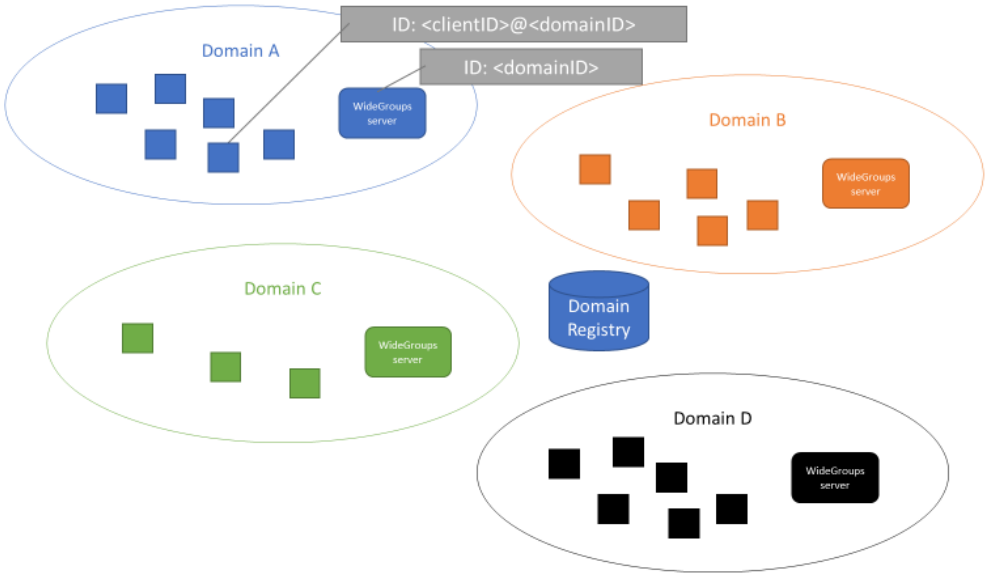


Figure 23 WideGroups domains and identities

WideGroups' global addresses are inspired to SMTP and XMPP clients' IDs. Considering that WideGroups is designed for wide area networks, an efficient system to assign unique addresses is necessary to route and deliver messages correctly. All WideGroups actors are addressable on the wide area network with WideGroupsIDs. As in SMTP and XMPP WideGroups servers' addresses are of the form <administrative_domain_ID> (e.g., unige.it), while clients registered on an administrative domain have addresses like <client_ID@administrative_domain_ID> (e.g., giancarlo@unige.it). Client information are always exchanged in the GroupInfo object. Messages are signed with the client's private key to identify the user.

Table 3 ClientInfo

Field Name	Type	Description
ClientID	String	Client unique identifier in the form <client_ID@administrative_domain_ID>

5.3 GROUPS

WideGroups implements a group communication framework directed towards the integration of several applications installed in different administrative domains that can interoperate exchanging WideGroups standard messages (WGMessage). The main abstraction in WideGroups are **groups**. Each group enables the exchange of messages among applications connected to WideGroups servers installed in different administrative domains. WideGroups supports two types of groups: **asymmetric groups** and **symmetric groups (**Table 4**)**. In asymmetric groups only the group owner can send messages to other group members. In symmetric groups more than one client can send messages at the same time in a group. Groups' properties are programmatically defined by one or more clients acting as group creators and managers. A group is characterized by its properties and by the message queue it handles. **Group queue** is a message queue accessible only by group members. Authorized clients can publish messages (basic and platform) on the group's queue. Different groups cannot interact with other groups' message queues. Access control policies are defined at group level and regulate group membership (add/ remove users to groups) and their permissions. The group controller is the WideGroups control plane module that handles groups (creation, deletion and management).

Table 4 GroupType

Enumeration entry name	Enumeration constant	Description
SYMMETRIC	0	The group to which this message refers to is a symmetric group with multiple writers.
ASYMMETRIC	1	The group to which this message refers to is an asymmetric group with a single writer.

5.3.1 Naming Convention

Each group has a unique identifier in the WideGroups network. The unique identifier contains the name of the group, the creation time and the creator identifier (<group_name><group_creation_time><creatorclientID>). If the group changes its administrator and the creator leaves the group, the creator's ID will remain as part of the group identifier to guarantee the uniqueness of IDs. The timestamp is useful

because if a creator leaves a group and wants to recreate it later the two groups must have two different identifiers. Other participants can still participate to both groups even if the original creator decided to disconnect from one of them.

5.3.1 Atomic multicast and virtual synchrony

In a reliable multicast group the message delivery reliability is guaranteed by the atomic multicast property. It states that a message should be delivered to all processes in a multicast group or not delivered at all. To achieve this multicast groups implement the concept called "virtual synchrony" [43]. A virtual synchronous distributed system uses the **group view** to keep a consistent view of a group at a certain point in time. Messages must be sent to all non-crashed members in a group view using atomic multicast. Events happening between a group view change and the other are events happening in the same epoch. Each time a new member joins or leaves (gently or crashing) the epoch changes. If a member crashes while multicasting a message, other members should remove the partially sent message or designate one of the survivors to send the message to the remaining nodes in the group view. Messages have typically two states: unstable and stable. Messages are stable after all the nodes have acknowledged the reception. Acknowledges can cause the problem of feedback implosion. To solve this issue several solutions have been proposed (e.g. negative acknowledgments [44]). For the framework described in this thesis I will use a hierarchical solution to control feedback. Participants are organized in subgroups or domains that have a local coordinator. A local coordinator is the proxy server that implements the application layer multicast protocol at the infrastructure level.

5.3.2 Asymmetric groups

An asymmetric group grants read only access to group messages to a set of clients that can be defined at the group creation time or later using the group management functionalities. Only the group writer (that corresponds to the group administrator) can send messages to the group members. Group members can be connected to the same writer's domain or to external domains. Asymmetric group usage can either be useful to send basic messages from a main application to back-up applications' data or to enable failover for external platforms using asymmetric groups with platform messages.

Only the group administrator can send messages: this choice prevents message ordering inconsistencies due to other clients' concurrent message dispatching. Each asymmetric group is stored only in the group owner's domain WideGroups server with all its metadata that include: last received message id, group id, a reference to the group's access control list, a reference to the group's history of messages and the address of where the group information and message queue are stored. Asymmetric groups use the **virtual synchrony** paradigm to keep a consistent view of the group participants. As there is only one administrator, the server associated to the client will keep a consistent view of other servers and other clients connected.

5.3.3 Symmetric groups

In symmetric groups the creator lets different clients concurrently send messages to a group. There is more than one active client that can be connected to different administrative domains. Only one of the clients act as a leader and all communications must pass from its message orderer. The leader election is performed using the RAFT algorithm. [16] The leader must order messages, order group management requests from group administrators and keep a consistent view of the members through the virtual synchrony.

5.3.4 Access Control Policies

There are three actors in WideGroups:

- **Readers:** actors that can access to groups in read only mode.
- **Writers:** actors that can access to groups in read and write modes.
- **Administrators:** actors that can change group information and modify actors' permissions.

- *Table 5 Role*

Enumeration entry name	Enumeration constant	Description
READER	0	Actor that can access to groups in read only mode.
WRITER	1	Actor that can access to groups in read and write modes.

ADMINISTRATOR	2	Actor that can change group information and modify actors' permissions.

Clients can decide to bind access control policies to single clients or create an access control rule equal for all the clients connected to a specific administrative domain. Access control policies are group related and are managed by group administrators.

5.4 MESSAGES

Messages can be control plane messages (message format varies for different WideGroups functionalities) or data plane messages (WGMessages). Each message has associated a quality of service.

5.4.1 Quality of service

WideGroups supports three levels of quality of service. As WideGroups' communication happens across administrative domains, it cannot guarantee client to client quality of service directly. The quality of service that a client specifies in a message is directly related to server-to-server guarantees. Server to clients guarantees are left to each external domain server. Domains server will follow the same requirements that a client specifies for a server to server quality of service.

Each message exchanged on a group has associated a quality of service. It supports three levels of quality of service:

- **Level 0:** (fire and forget) does not guarantee message delivery to all interested servers and clients. An acknowledge is returned from the corresponding domain server without waiting acknowledgments from external domains. It means that it does not require that all WideGroups server are online to receive messages and it does not require that all clients involved in the message exist (they can register to their domains' WideGroups servers later). It retries sending information to WideGroups server that are not online for a configurable number of retries. It returns only an asynchronous warning message to inform what are the domains that did not receive the message on a non-blocking callback that clients can optionally configure.

- **Level 1:** it guarantees that at least one of the involved external servers has received the message, but not that all WideGroups servers will receive the message. Client sending the message synchronously wait for the acknowledge from its domain server. Acknowledgment is sent to the client only after an external domain responds to the message. It requires that WideGroups domains' servers exist and are online to receive the message. It does not require all clients to exist on the domains' servers, they can register later.
- **Level 2:** Level 2 guarantees that all messages are delivered to interested WideGroups servers. Client sending the message waits for the acknowledge from its server that must wait for acknowledges from all the domains. It requires that WideGroups domains' servers exist and are online to receive group creation information. It requires that all clients exist (they can be offline) and are registered on the WideGroups' servers.

Table 6 summarizes the enumeration QOS type entries.

Table 6 QOS

Enumeration entry name	Enumeration constant	Description
NO_WAIT	0	Level 0 (fire and forget) does not guarantee message delivery to all interested servers and clients.
WAIT_ONE	1	Level 1 guarantees that all servers will receive the message, but not that all clients will receive the message, leaving this responsibility to the external servers.
WAIT_ALL	2	Level 2 guarantees that all messages are delivered to interested WideGroups servers.

5.4.2 Control plane messages

Control plane messages contains the parameters of the following functionalities. A more detailed description can be found in chapter 5.5.5.

Table 7 Control plane functions and related control plane messages

Function Name	Related Control Plane Message
Connect	BasicRequestMessage
Disconnect	BasicRequestMessage
CreateGroup	GroupCreationMessage
DeleteGroup	GroupDeletionMessage
ModifyGroup	GroupModifyMessage
GetGroups	GetGroupsMessage
GetGroup	GetGroupMessage

BasicRequestMessage is used in connect and disconnect functions.

Table 8 BasicRequestMessage

Field name	Type	Description
ClientInfo	ClientInfo	Parameters to identify the client. (Table 3)

GroupCreationMessage carries information necessary to create a group.

Table 9 GroupCreationMessage

Field name	Type	Description
ClientCredentials	BasicRequestMessage	Parameters to identify the client. (Table 3)
GroupID	String	Unique group identifier (automatically generated by WideGroups client)
GroupType	GroupType (Enum)	Asymmetric or Symmetric groupTable 4
QOS	QOS (Enum)	Quality of service (Table 6)
InitialParticipants	Map of (ClientInfo, Role)	(optional) The initial participants (Table 3) and the corresponding role (Table 5).
Description	String	An optional description of the group.

GroupDeletionMessage carries information necessary to delete a group.

Table 10 GroupDeletionMessage

Field name	Type	Description
ClientCredentials	BasicRequestMessage	Parameters to identify the client. (Table 3)
GroupID	String	Unique group identifier (automatically generated by WideGroups client)
GroupType	GroupType (Enum)	Asymmetric or Symmetric group Table 4

GroupModifyMessage carries information necessary to modify group information and access control lists.

Table 11 GroupModifyMessage

Field name	Type	Description
ClientCredentials	BasicRequestMessage	Parameters to identify the client. (Table 3)
GroupID	String	Unique group identifier (automatically generated by WideGroups client)
GroupType	GroupType (Enum)	Asymmetric or Symmetric groupTable 4
QOS	QOS (Enum)	Quality of service (Table 6)
ParticipantsToAdd	Map of (ClientInfo, Role)	(optional) The participants (Table 3) to add/ modify and the corresponding role Table 5.
ParticipantsToRemove	Array of (ClientInfo)	(optional) The participants (Table 3) to remove.
Description	String	(optional) the new description

Table 12 GetGroupsMessage

Field name	Type	Description
ClientCredentials	BasicRequestMessage	Parameters to identify the client. (Table 3)
GroupType	GroupType (Enum)	Asymmetric or Symmetric group Table 4

Table 13 GetGroupMessage

Field name	Type	Description
ClientCredentials	BasicRequestMessage	Parameters to identify the client. (Table 3)
GroupID	String	Unique group identifier (automatically generated by WideGroups client)
GroupType	GroupType (Enum)	Asymmetric or Symmetric group Table 4

5.4.3 Data plane messages

Data plane messages are well defined messages called WGMessages exchanged with data plane functions such as “sendmessage” and “subscribetomessages”. Current WideGroups framework model and implementation supports three types of messages defined by the enumeration MessageType described in Table 15. In Table 14 I describe WGMessage fields and their content.

Table 14 WGMessage

Field name	Type	Description
Description	String	A description of the message content.
Payload	Byte array	Message payload. The type is specified by the “messagetype” field.
MessageID	String	A universal unique identifier of the message.
ClientInfo	ClientInfo	See the ClientInfo description in section
MessageType	Enum (MessageType)	It identifies the content of the payload that can be: - BASIC - MOM (a platform message that contains a MOM connector API message) - KVSTORE (a platform message that contains a Key Value Store connector message)
Timestamp	Int64	Timestamp (client time between client and corresponding WGServer, client domain’s server time among servers)
Properties	Map<String, Byte Array>	User defined additional properties.

The message type field is fundamental to trigger different pipelines on widegroups servers depending on the message received. There are two categories (BASIC and PLATFORM) and three types of messages (BASIC, MOM, KVSTORE).

Table 15 MessageType

Enumeration entry name	Enumeration constant	Description
BASIC	0	Used for basic messages

MOM	1	Used for message oriented middleware platform messages
KVSTORE	2	Used for key value store platform messages

Table 16 Payload of a MOM message

Field name	Type	Description
MOMFunctionName	Enum (MOMFunctionName)	The name of the function.
Topic	ByteArray	Message Topic in the standard format
Payload (optional)	ByteArray	In publish functionalities it contains the payload.
Timestamp	Int64	Timestamp (client time between client and corresponding WGServer, client domain's server time among servers)
Properties	Map<String, Byte Array>	Function related additional properties.

Table 17 MOMFunctionName

Enumeration entry name	Enumeration constant	Description
PUBLISH	0	Publish a message on a specific topic.
SUBSCRIBE	1	Subscribe to a topic with standard wildcarding. Additional properties: - DURABLE
RECEIVE	2	Get the latest message published on a topic.
UNSUBSCRIBE	3	Unsubscribe from a topic.

Table 18 Payload of a KVSTORE message

Field name	Type	Description
KVSTOREFunctionName	Enum (KVSTOREFunctionName)	The name of the function.
Key	ByteArray	The entry's key
Value (optional)	ByteArray	(optional)In set functionalities contains the value(s).

Timestamp	Int64	Timestamp (client time between client and corresponding WGServer, client domain's server time among servers)
Properties	Map<String, Byte Array>	Function related additional properties.

Table 19 KVSTOREFunctionName

Enumeration entry name	Enumeration constant	Description
GET	0	Get an entry from the key value store. Additional properties: - KVStoreName
GETALL	1	Get all the entries from a key value store. Additiona properties: - Filter with regex - KVStoreName - Only keys (to get only the keys)
SET	2	Put a key value pair in the store
SETVALUES	3	Put a set of key value pairs
REMOVE	4	Removes the specified entry

5.4.4 Protocol buffers definition

Messages are defined through protocol buffers and have the format described in Listing 3 and Listing 4

```
syntax = "proto3";
option java_package = "it.cipi.wgroups.protocol.grpc";

message ProtoClientID {
        string name = 1;
}
message ProtoGroupID {
        string name = 1;
}
message ProtoDomainID {
        string name = 1;
}
message ProtoClientAddress {
        ProtoClientID client_id = 1;
        ProtoDomainID domain_id = 2;
}
message ProtoGroupAddress {
        ProtoGroupID group_id = 1;
        ProtoDomainID domain_id = 2;
}
message ProtoGroupInfo {
        ProtoGroupID group_id = 1;
        int64 version = 2;
        ProtoGroupType group_type = 3;
        repeated ProtoClientAddress client_addresses = 4;
}
message ProtoWGMessage {
        string description = 1;
        bytes payload = 2;
        string message_id = 3;
        ProtoMessageType message_type = 4;
        int64 timestamp = 5;
        map<string, bytes> properties = 6;
}
enum ProtoGroupType {
        SYMMETRIC = 0;
        ASYMMETRIC = 1;
}
enum ProtoMessageType {
        BASIC = 0;
        MOM = 1;
        KVSTORE = 2;
}
enum ProtoQOS {
        NO_WAIT = 0;
        WAIT_ONE = 1;
        WAIT_ALL = 2;
}
```

Listing 3 WideGroups messages (part 1)

```
message ProtoClientRequest {
        ProtoClientID sender_client_id = 1;
}
message ProtoEmptyParam {
        ProtoClientRequest basic_request = 1;
}
message ProtoConnectParam {
        ProtoClientRequest basic_request = 1;
}
message ProtoMessageAckParam {
        ProtoClientRequest basic_request = 1;
        repeated string acknowledged_message_ids = 2;
}
message ProtoCreateGroupParam {
        ProtoClientRequest basic_request = 1;
        ProtoGroupID group_id = 2;
        ProtoGroupType group_type = 3;
        ProtoQOS qos = 4; // Used only for asymmetric groups
        repeated ProtoClientAddress initial_addresses = 5;
        string description = 6; // Used only for symmetric groups
}
message ProtoGetGroupsParam {
        ProtoClientRequest basic_request = 1;
        ProtoGroupType group_type = 2;
}
message ProtoModifyGroupParam {
        ProtoClientRequest basic_request = 1;
        ProtoGroupID group_id = 2;
        ProtoGroupType group_type = 3;
        ProtoQOS qos = 4;
        repeated ProtoClientAddress added_addresses = 5;
        repeated ProtoClientAddress removed_addresses = 6;
}
message ProtoSingleGroupParam {
        ProtoClientRequest basic_request = 1;
        ProtoGroupID group_id = 2;
        ProtoGroupType group_type = 3;
}
message ProtoClientSendMessageParam {
        ProtoClientRequest basic_request = 1;
        ProtoGroupID group_id = 2;
        ProtoGroupType group_type = 3;
        ProtoQOS qos = 4;
        ProtoWGMessage message = 5;
}
message ProtoGroupResponse {
        ProtoGroupInfo group_info = 1;
}
message ProtoGroupsListResponse {
        repeated ProtoGroupInfo groups_info = 1;
}
message ProtoWGMessageResponse {
        ProtoWGMessage message = 1;
        ProtoClientAddress sender = 2;
        ProtoGroupID group_id = 3;
        ProtoGroupType group_type = 4;
        ProtoDomainID group_host_domain = 5;
}
```

Listing 4 WideGroups messages (part 2) continues from part 1

5.5 WIDEGROUPS SERVER

5.5.1 Group Controller Module

Group controller module is the module in charge of registering groups in which a specific WideGroups server is involved because one of its clients is either the owner (MangedGroup) or participates in a group (ExternalGroup). Calls to the Group Controller are always filtered by the Authrorization Module that checks clients' permissions to modify group information and to accept group information updates from external domains. A group controller always contains the group registry where groups are stored.

A group registry has two main data structures:

- **Managed Groups**: managed groups contains the groups in which the administrator (or manager) is one of the clients registered to the group registry's WideGroups domain.
- **External Groups:** external groups are groups to which clients registered, but are not administrators.

Group controller exposes an interface to manage groups through control plane messages. Group management receives client requests from the WGClient Communication module that include: external and managed group join requests, group creation requests, group management requests and group deletion requests.

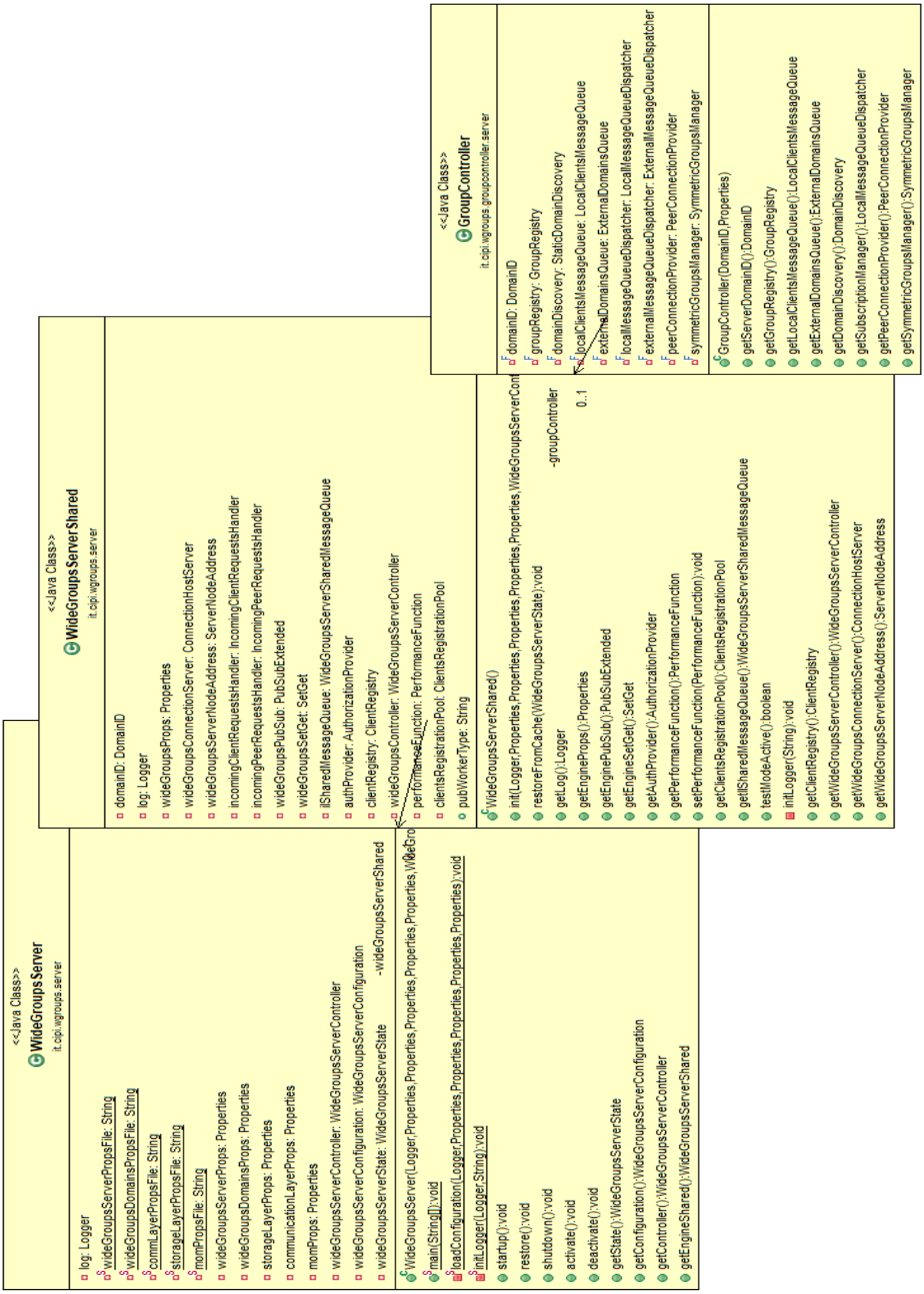
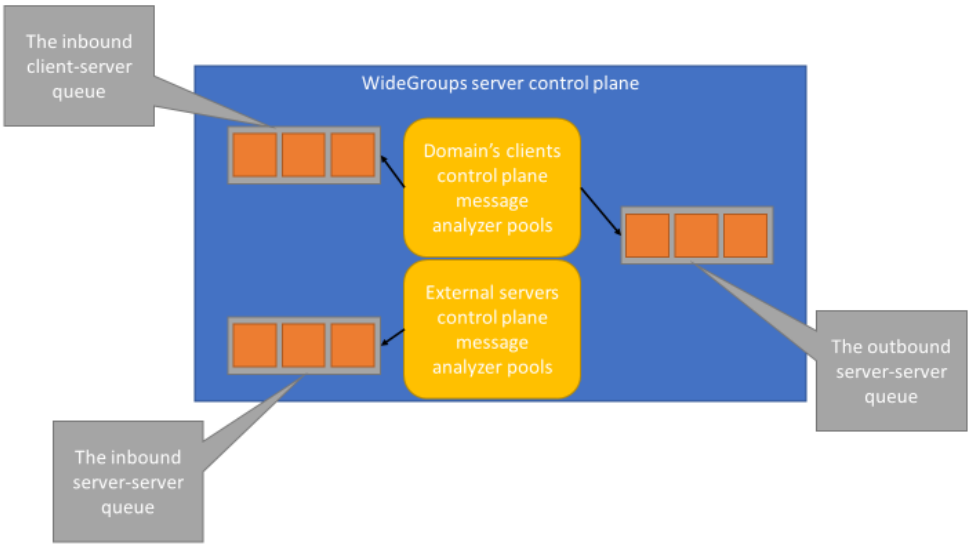


Figure 24 The Group Controller Module inside the WideGroups Server

5.5.1.1 Group creation



This is the list of operations carried on to create a new group in WideGroups after a client calls the "createGroup" function.

1) A WideGroups client connected to its domain's WideGroups server sends a group creation request to the WideGroups server. The group creation request has the parameters listed in Table 20 .

Table 20 Group Creation parameters

Parameter name	**Description**
Name (required)	A descriptive name of the group
Description (optional)	A description of the group
GroupMessagesTypes (required)	The type of messages that will be exchanged among group participants: - Basic messages - Platform messages
Participants (optional parameter)	A list of participants addresses "clientID@domainID"

2) The WideGroups server processes the group creation request.

- The WideGroups server client communication interface receives the request and forwards it to the authorization module.
- The authorization module checks client's credentials and if it is authorized to create a new group.
- If yes, the group controller module checks if participants belong to existing and running domains verifying if the WideGroups servers are registered with the domain IDs declared. Depending on the domain registry type (DNS or private) there will be different implementations of the domain verification step.
- Depending on the quality of service chosen. If the WideGroups server cannot find some domains it returns either an error message to the WideGroups client or a warning message listing the domains that cannot be found. (Clients can retry sending the group creation request removing the participants that belong to the domains not found). Otherwise it proceeds with the group creation routine checking if the WideGroups servers associated to the domains are currently online. Depending on quality of service it has different behaviors for offline domains:
    - **Level 0:** it does not require that all WideGroups server are online to receive group creation messages and it does not require that all clients exist (they can register to their domains' WideGroups servers later). It retries sending group creation information to WideGroups server that are not online for a configurable number of retries. It returns only a warning message to inform what are the domains that received the group creation information and creates a group only with the online servers.
    - **Level 1:** it requires that WideGroups domains' servers exist and are online to receive group creation information. It does not require all clients to exist on the domains' servers, they can register later.

- **Level 2:** it requires that WideGroups domains' servers exist and are online to receive group creation information. It requires that all clients exist (they can be offline) and are registered on the WideGroups' servers.

- It retries to establish a connection for a settable number of retries before giving up. If it cannot connect to all domains, WideGroups server returns an error message to the client indicating the unreachable servers. (Clients can retry sending the group creation request removing the participants that belong to the unreachable domains). Otherwise it proceeds with the group creation routine storing group information locally.

3) Online external WideGroups receive the group creation message and check authorization.
4) External domains return an acknowledge message to the calling server.
5) The WideGroups server that initiated the routine calls the group controller module function to create the group.
6) The group creation function returns an acknowledge message to the domain A.
7) The domain A receives the first acknowledge message and can stop waiting for other acknowledge messages or wait for more acknowledge messages depending on the chosen quality of service.
8) The group creation message is enqued and sent to other servers
9) External servers process the request.
10) They finally send back an acknowledge message
11) The WideGroups server that started the creation process receives acknowledges from other servers and stores information locally ending the process.

For symmetric groups there is an additional step to elect the new leader among participants through the RAFT algorithm.

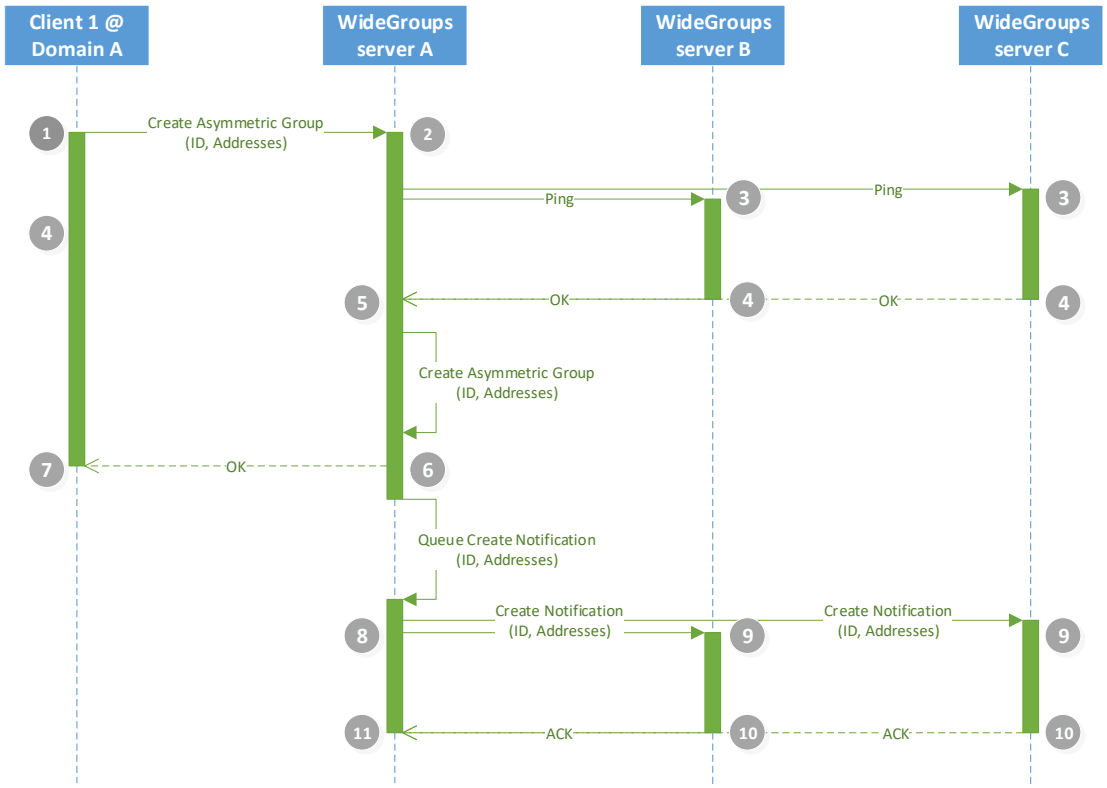


Figure 25 Asymmetric Group creation process involving three domains for quality of service 0

5.5.1.2 Group management

Group management requests let clients to delete groups or edit group settings adding clients to a group. They follow the same sequence described for group creation.

5.5.1.3 Group Controller functions

In the table below I include an extract from the group registry JavaDoc

ManagedGroup	createGroupRequest(ClientAddress requestExecutor, GroupID groupID, java.util.Collection<ClientAddress> initialAddresses) Handle the request of a client to create a group.
ManagedGroup	deleteGroupRequest(ClientAddress requestExecutor, GroupID groupID) Handle the request of a client to delete a group.

java.util.Collection<GroupInfo>	getAllGroupsRequest(ClientAddress requestExecutor) Handle the request of a client to get the groups it's a part of.
ExternalGroupsView	getExternalGroupsView()
ManagedGroup	getGroup(GroupID groupID)
ManagedGroup	getGroupRequest(ClientAddress requestExecutor, GroupID groupID) Handle the request of a client to get a group.
void	onRemoteVersionCommitted(ManagedGroup group, DomainID remoteDomain, long versionCommitted) Called when a version is received on a remote server.
void	setRemoteVersionCommittedListener(RemoteVersionCommittedListener listener) Set a global listener for when a version of a group is received on a remote server.

5.5.2 Discovery module

Discovery of servers and clients is essential to setup groups and enable cross administrative domain communication. WideGroups uses global addresses instead of IP addresses for connections and it needs registries to translate global addresses into IP addresses. WideGroups servers can register their names to public DNS or to private registries.

5.5.2.1 DNS usage

As in XMPP and SMTP WideGroups implements a decentralized network that needs an efficient distributed discovery mechanism to setup connections. DNS usage is optional in WideGroups, but is the mandatory mechanism for public WideGroups networks. In order for clients to connect and register to WideGroups server, they need to find the IP address of the server associated with their domain. Similarly remote servers which need to send a message to clients connected to external domains need to discover those servers to correctly forward messages. WideGroups domains must be registered as an SRV record (see section 4.2.1.1). As in XMPP WideGroups has two

types of SRV records: one for client to server communications and one for server to server communications.

```
_wg-client._tcp.example.com. 18000 IN SRV 0 5 50000 wg.example.com.
_wg-server._tcp.example.com. 18000 IN SRV 0 5 50100 wg.example.com.
```

Listing 5 WideGroups SRV records

5.5.2.2 Private Registry usage

For private networks, WideGroups can use private registries to setup lookup services for private WideGroups servers. Private registries must be stored on a distributed and efficient key value store where entries will have a key containing a unique domainID and a value containing a WGDomainInfo record with the following fields:

- **Domain-ID**: the domain ID corresponding to the entry's key
- **IP-address**: the IP address where the Widegroups server is running
- **Client-Communication-Port:** the TCP port where client communication services are reachable.
- **Server-Communication-Port:** the server to server communications TCP port.
- **TTL:** as in the DNS SRV record, it specifies the time interval (in seconds) that the resource record may be cached before it should be discarded.

5.5.3 Consistency Module

Control plane message consistency is guaranteed by group manager module. Data plane consistency regards message ordering for each group. WideGroups guarantees sequential consistency to WideGroups clients and two levels of consistency to WideGroups servers through the consistency module.

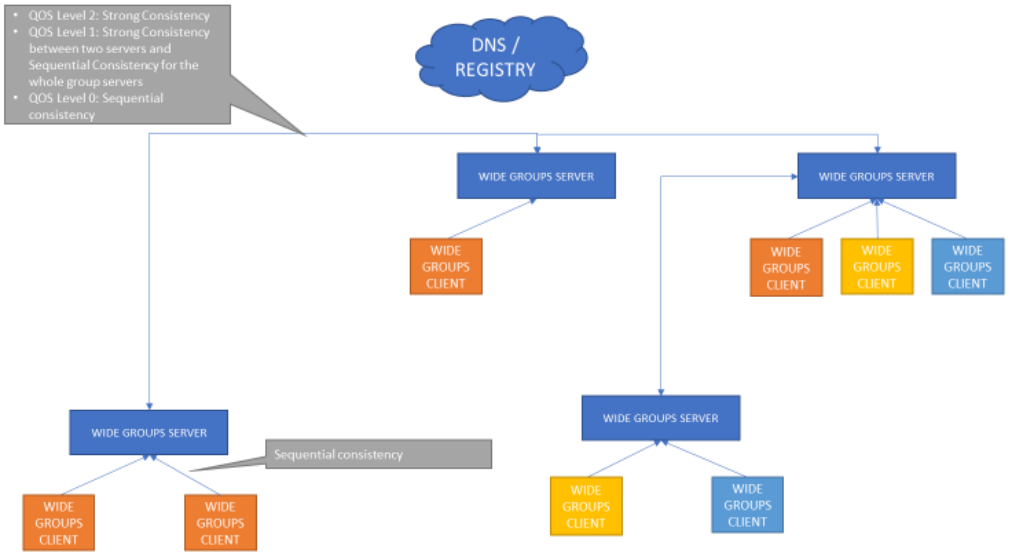


Figure 26 Consistency types

In asymmetric groups only one client can send messages to other group members. Messages have a sequence number that works as a logical clock. Logical clocks [10] are a good choice to guarantee local order and sequential consistency. Asymmetric groups must guarantee only the local order of operations as they are implemented using single writer multiple readers queues that can be modified only from their owner from one client application at a time. As it happens in the Spreadsheet Space view-image synchronization protocol, the active owner's consistency module takes care of operations' ordering. Talking about server to server consistency across different administrative domains following the CAP theorem principles [6] group owners can decide if guarantee strong consistency with the highest level of quality of service (level 2) sacrificing availability or sequential consistency (as it happens with clients) relaxing the consistency property (level 0 and 1). With quality of service 0 and 1 all group message queues will be the exact replication of their corresponding owner's group message queue when the system is at quiescence (when there are not new incoming messages). Symmetric groups consistency involves different group participants concurrently sending messages to the same group. It is equal to concurrently add messages to a shared message queue. In order to have a consistent sequence of messages at each peer, there must be an ordering service that guarantees consensus

among the distributed peers with a global order of messages. The order of messages is generated only from one consistency module at a time that acts as a leader. WideGroups assigns to one of the online collaborators' WideGroups servers the role of the leader using RAFT consensus algorithm [17]. The group queue is maintained by the current leader. The current leader collects group messages from special queues exposed by group participants. Each node that participates to a symmetric group creates its own queue and exposes it to other group members to transmit its own messages. After the election, the leader connects to all group members queues to merge messages coming from different nodes in a single globally ordered message queue. Messages received from group members are properly ordered using their version numbers and the time of their arrival to the leader's node. Using RAFT consensus algorithm I guarantee fault tolerance and network partition tolerance through the election of a new leader. [17]

5.5.4 Authorization Module

In WideGroups security and confidentiality are primary concerns as it is designed for cross administrative domain communication. Trusted certificate authorities release X.509v3 certificates for WideGroups servers. Each WideGroups server is always associated with a domain identified with a certificate. The certificates are used from other participants (clients and other WideGroups servers) to verify identities. WideGroups servers release and sign certificates for their clients during the client's registration phase. WideGroups' elements are hierarchically organized in domains. In this section I focus on security aspects of the WideGroups framework.

Nodes in WideGroups belong always to a domain. Each domain has an handler, the WideGroups server, identified with a certificate signed by a trusted certificate authority. Each domain configures a list of trusted certificate authorities and verifies other WideGroups servers identities before connection. Each WideGroups server will authorize and sign certificates for its clients. WideGroups supports the certificate extension .pem. Certificates are Base64 encoded and can be generated for test purposes using a certificate generation tool like **OpenSSL** on Linux. In production environments it is necessary to use external **trusted certificate authorities** as Symantec (Verisign), GeoTrust, DigiCert, GoDaddy and Comodo. Let's Encrypt is a free solution to generate

server certificates, but it has limited functionalities. In Listing 5 I include an example of a WideGroups certificate.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            10:67:f8:9d:62:99:48:b0:d1:5b:6f:57:69:4a:94:be
    Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=US, ST=California, L=San Francisco, O=widegroups.domainA.com, CN=ca.widegroups.domainA.com
        Validity
            Not Before: Mar  8 13:40:00 2019 GMT
            Not After : Mar  5 13:40:00 2029 GMT
        Subject: C=US, ST=California, L=San Francisco, CN=peer0.widegroups.domainA.com
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:73:a0:10:8b:00:e3:12:07:49:85:5d:af:86:45:
                    c3:23:be:25:b6:4d:79:34:40:ff:76:25:c7:74:68:
                    92:be:ff:0e:7e:f0:e4:05:ac:0d:0d:55:c4:a8:4b:
                    18:b6:f8:7b:7c:b7:04:80:c6:51:36:2c:b9:53:41:
                    b8:9b:87:ba:b1
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Authority Key Identifier:
                keyid:F7:BD:A3:D0:2E:3F:D5:09:52:8E:92:02:A5:E9:FB:D5:A8:FC:80:C1:33:7D:C5:C1:1E:15:1C:6B:D7:62:B2:B8

    Signature Algorithm: ecdsa-with-SHA256
         30:44:02:20:14:93:af:69:bc:c4:c7:fb:e9:8f:e1:5b:f0:4f:
         5e:69:dd:60:dc:bc:07:e9:a4:05:95:94:22:4f:f3:f9:f9:90:
         02:20:40:14:55:4a:bf:d4:f4:af:87:22:46:54:ae:ca:4f:9f:
         e7:47:96:1e:1f:be:fe:61:67:2c:1f:ec:dd:fa:08:c4
```

Listing 6 example of WideGroups certificate

Certification authorities can revoke a certificate adding it to a certificate revocation list. Certificate revocation causes the expiration before the end of its validity.

The authorization module checks clients and servers identities and groups' authorization. It has a default interface called authorization provider.

5.5.5 WGClient Communication Module

The WGClient communication module accepts both control plane and data plane remote procedure calls. See chapter 5.4.2 for messages description.

Table 21 Control plane functions list

Control plane function	Message exchanged	Description
Connect	BasicRequestMessage	It connects a WGClient to its WGServer. It contacts the authorization module to check authorization.
Disconnect	BasicRequestMessage	It disconnects a WGClient from its WGServer

CreateGroup	GroupCreationMessage	Through the authorization module and group controller it creates a new group.
DeleteGroup	GroupDeletionMessage	Through the authorization module and group controller it deletes an existing administered group.
ModifyGroup	GroupModifyMessage	Through the authorization module and group controller it modifies information regarding a group.
GetGroups	GetGroupsMessage	Through the authorization module and group controller it gets a set of groups information.
GetGroup	GetGroupMessage	Through the authorization module and group controller it gets a specific group information

Table 22 Data plane functions list

Data plane function	Message exchanged	Description
SendMessage	WGMessage	This function is used to send data plane messages to the WideGroups servers. The payload is a WGMessage described in Table 14.
SubscribeToMessages	WGMessage	This function is used to receive WideGroups messages (WGMessage) exchanged on a specific group.

The Client to Server protocol is defined through protocol buffers and gRPC services.

```
syntax = "proto3";
option java_package = "it.cipi.wgroups.protocol.grpc";
import "commons.proto";

service GroupControllerClientHandler {
        rpc Connect (ProtoConnectParam) returns (ProtoEmptyResponse);
        // Control plane
        rpc CreateGroup (ProtoCreateGroupParam) returns (ProtoEmptyResponse);
        rpc DeleteGroup (ProtoSingleGroupParam) returns (ProtoEmptyResponse);
        rpc GetGroups (ProtoGetGroupsParam) returns (ProtoGroupsListResponse);
        rpc GetGroup (ProtoSingleGroupParam) returns (ProtoGroupResponse);
        rpc ModifyGroup (ProtoModifyGroupParam) returns (ProtoEmptyResponse);
        // Data plane
        rpc SendMessage (ProtoClientSendMessageParam) returns (ProtoEmptyResponse);
        rpc SubscribeToMessages (stream ProtoMessageAckParam) returns (stream
ProtoWGMessageResponse);
}
```

Listing 7 WGClient Communication Module

5.5.6 WGServer Communication Module

The WGServer Communication Module must take care of the domain to domain communication among WideGroups servers.

```
syntax = "proto3";
option java_package = "it.cipi.wgroups.protocol.grpc";
import "commons.proto";

service GroupControllerPeerHandler {
          rpc Ping (ProtoPingParam) returns (ProtoEmptyResponse);

          // Control plane
          rpc GroupCreated (ProtoGroupCreatedParam) returns (ProtoEmptyResponse);
          rpc GroupModified (ProtoGroupModifiedParam) returns (ProtoEmptyResponse);
          rpc GroupDeleted (ProtoGroupDeletedParam) returns (ProtoEmptyResponse);

          rpc SymmetricGroupCreateCommit (ProtoSymmetricGroupCreateCommit) returns (ProtoEmptyResponse);
          rpc SymmetricGroupMessageCommit (ProtoSymmetricGroupMessageCommit) returns (ProtoEmptyResponse);

          // Data plane
          rpc SendMessage (ProtoPeerSendMessageParam) returns (ProtoEmptyResponse);
          rpc SendSymmetricMessageAsLeader (ProtoSymmetricMessageAsLeaderParam) returns (ProtoEmptyResponse);
}

message ProtoPeerRequest {
          ProtoDomainID sender_domain_id = 1;
}
message ProtoPingParam {
          ProtoPeerRequest basic_request = 1;
}
message ProtoGroupCreatedParam {
          ProtoPeerRequest basic_request = 1;
          ProtoGroupID group_id = 2;
          int64 new_version = 3;
          repeated ProtoClientAddress initial_addresses = 4;
}
message ProtoGroupModifiedParam {
          ProtoPeerRequest basic_request = 1;
          ProtoGroupID group_id = 2;
          int64 new_version = 3;
          repeated ProtoClientAddress added_addresses = 4;
          repeated ProtoClientAddress removed_addresses = 5;
}
message ProtoGroupDeletedParam {
          ProtoPeerRequest basic_request = 1;
          ProtoGroupID group_id = 2;
          int64 new_version = 3;
}

message ProtoPeerSendMessageParam {
          ProtoPeerRequest basic_request = 1;
          ProtoClientAddress sender = 2;
          ProtoGroupAddress group = 3;
          repeated ProtoClientID client_recipients = 4;
          ProtoWGMessage message = 5;
}

message ProtoSymmetricMessageAsLeaderParam {
          ProtoPeerRequest basic_request = 1;
          ProtoClientAddress sender = 2;
          ProtoGroupID group_id = 3;
          ProtoWGMessage message = 4;
}

message ProtoSymmetricGroupCreateCommit {
          ProtoPeerRequest basic_request = 1;
          ProtoGroupID group_id = 2;
          string group_description = 3;
          ProtoDomainID leader_id = 4;
          repeated ProtoClientAddress addresses = 5;
}

message ProtoSymmetricGroupMessageCommit {
          ProtoPeerRequest basic_request = 1;
          ProtoClientAddress sender = 2;
          ProtoGroupID group_id = 3;
          ProtoWGMessage message = 4;
}
```

Listing 8 WGServer Communication Module

5.5.7 Local Platform Connector

Software platforms are usually installed inside enterprise or department boundaries to maintain control over data location and access. I call these platforms "local" platforms. They are easy to deploy and to use and speed up the development of complex systems. WideGroups local platform connector is a connector able to communicate with locally installed platforms through a standard API. The standard API is designed based on the platform's category common APIs. I designed and implemented connectors for platforms based on Message Oriented Data Exchange and Key Value Store, which are among the most widespread paradigms supporting data sharing in distributed systems.

WideGroups server can dynamically load Local Platform Connectors through the methods provided in the Connector Delegates. After loading delegate properties it is possible to register connectors through the registerDelegate function (Connector factory class in Table 23 provides a registerDelegate method to register an instance of a Connector for a specific local platform). WideGroups server can support more than one connector and platform messages can specify which platform use. After registering the delegate, in order to use it must be instantiated through the getInstance function. Each connector must implement the connector delegate interface and the standard API defined for that platform's cathegory. Table 23 and Table 24 define the connector factory and connector delegate common interfaces. The MOM Connector standard interface is defined in Table 25 while the Key Value Store Connector components are defined in Table 26

Table 23 Connector Factory

static **MOMConnector**	**getInstance**(java.lang.String delegateName)
static void	**registerDelegate**(java.lang.String delegateClassName, java.util.Properties delegateProperties) Method for registering delegates.
static void	**shutdown**()

Table 24 Connector Delegate Interface

java.lang.String	**getId**()
MOMConnector	**getInstance**() invoked by the Factory to create this type of MOM Connector Factory delegates the creation to the delegate
boolean	**isStarted**() Method used by Factory to check whether the Delegate has been initialized
void	**shutdown**() In the shutdown method the delegate will have to perform every cleanup operations that are needed when MOM Connector creation is no longer active (connection closing, freeing resources, etc) This method will be called when MOMConnectorFactory shuts down
void	**startup**(java.util.Properties p) method for starting the delegate.

Table 25 MOMConnector standard interface

void	connect()perform the low level connection operation and authentication.
void	disconnect()perform the low level disconnection operation.
void	publish(java.lang.String topic, MomMessage message)perform the publish operation on the message oriented middleware.
java.lang.String	subscribe(java.lang.String topic, java.lang.String uuid, SubscriberConnectorCallback cb)Performs a subscription to a specific topic or to a family of topics using wildcarding.
java.lang.String	subscribe(java.lang.String topic, SubscriberConnectorCallback cb)Performs a subscription to a specific topic or to a family of topics using wildcarding.
void	persistentPublish(java.lang.String topic, MomMessage message)perform the persistent publish operation on the message oriented middleware.

java.lang.String	**durableSubscribe**(java.lang.String topic, java.lang.String uuid, **SubscriberConnectorCallback** cb) Performs a durable subscription to a specific topic or to a family of topics using wildcarding.
MomMessage	receive(java.lang.String topic) Receive is a blocking function to receive a message published on a specific topic.
void	unsubscribe(java.lang.String subscriptionID)It stops a subscription identified with a subscription id

Table 26 KVSTORE standard interface

void	disconnect()Connector API to disconnect client from the KV store.
java.lang.Object	get(java.lang.String key)Retrieves the Object stored as VALUE in the IMDG at the corresponding KEY, accessing the cache which has been configured as default
java.lang.Object	get(java.lang.String key, java.lang.String cacheName)It has the same behavior as get(String) but it operates on a specific cache
java.util.Set<java.lang.String>	getAllKeys()This method returns all the keys available in the default cache of the KEY/VALUE store
java.util.Set<java.lang.String>	getAllKeys(java.lang.String cacheName)It has the same behavior as getAllKeys() but it operates on a specific cache
java.util.Map<java.lang.String,java.lang.Object>	getAllValues()This method will return all the KEY/VALUE pairs currently stored in the KEY/VALUE store NOTE:Invoking this method can be time consuming
java.util.Map<java.lang.String,java.lang.Object>	getAllValues(java.lang.String cacheName)It has the same behavior as getAllValues() but it operates on a specific cache NOTE:Invoking this method can be time consuming
java.util.Map<java.lang.String,java.lang.Object>	getValues(java.util.Set<java.lang.String> keyFilter)This method will return all KEY/VALUE pairs stored in the KV

	store provided that the KEY is an exact match one of the keys contained in the set passed as parameter.
java.util.Map<java.lang.String,java.lang.Object>	getValues(java.util.Set<java.lang.String> keyFilter, java.lang.String cacheName) It has the same behavior as getValues(Set) but it operates on a specific cache.
java.util.Map<java.lang.String,java.lang.Object>	getValues(java.lang.String keyFilter)This method will return all the KEY/VALUE pairs currently stored in the KEY/VALUE store applying a Regular Expression Filter on the KEYS.
java.util.Map<java.lang.String,java.lang.Object>	getValues(java.lang.String keyFilter, java.lang.String cacheName)It has the same bahavior as getAllValues(String) but it operates on a specific cache
VersionedObject	getVersioned(java.lang.String key)For the KEY/VALUE stores which provides a versioning on the VALUES stored at a specific KEY, this method returns a VersionedObject The versioned object returned will contains the VALUE and the VERSION of the value.
VersionedObject	getVersioned(java.lang.String key, java.lang.String cacheName)It has the same behavior as getVersioned(String) but it operates on a specific cache
java.lang.Object	remove(java.lang.String key, java.lang.String cacheName)Remove an element from a specific cache and return its value.
void	set(java.lang.String key, java.lang.Object element)Perform the "set" operation, i.e. will save a KEY/VALUE to the IMDG In case a value has already stored for the specific KEY, it will be updated
void	set(java.lang.String key, java.lang.Object element, java.lang.String cacheName)Same behavior as the method set(String, Object), but it allows to specify the cache name to be used

void	setValues(java.util.Map<java.lang.String,java.lang.Object> elements)Performs a bulk store of KEY/VALUE pairs, using the default IMDG cache
boolean	setVersioned(java.lang.String key, VersionedObject oldValue, java.lang.Object newValue)For the KEY/VALUE stores which provides a versioning on VALUES and ATOMIC OPERATIONS on values, this method atomically updates an existing value.
boolean	setVersioned(java.lang.String key, VersionedObject oldValue, java.lang.Object newValue, java.lang.String cacheName)It has the same behavior as setVersioned(String, VersionedObject, Object) but it operates on a specific cache.

5.6 WIDEGROUPS CLIENT

The WideGroups client exposes API to connect, disconnect a client from the WideGroups network and use the main functionalities to create and manage groups and to send messages (both basic and platform messages). In the diagram above I include an overview of the WideGroups client UML class diagram for the main classes and interfaces.

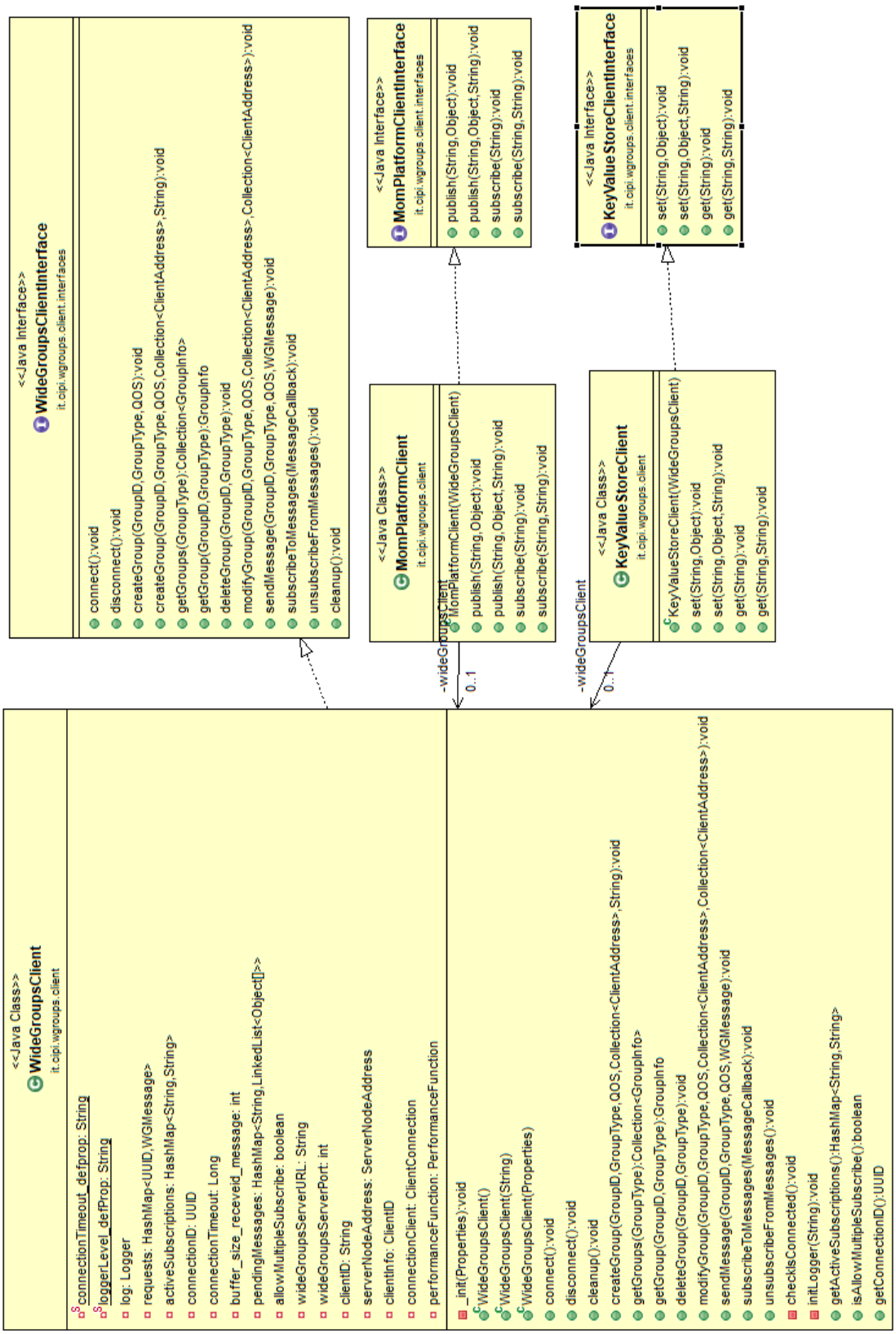

<<Java Interface>>
WideGroupsClientInterface
it.cipi.wgroups.client.interfaces
connect():void
disconnect():void
createGroup(GroupID,GroupType,QOS):void
createGroup(GroupID,GroupType,QOS,Collection<ClientAddress>,String):void
getGroups(GroupType):Collection<GroupInfo>
getGroup(GroupID,GroupType):GroupInfo
deleteGroup(GroupID,GroupType):void
modifyGroup(GroupID,GroupType,QOS,Collection<ClientAddress>,Collection<ClientAddress>):void
sendMessage(GroupID,GroupType,QOS,WGMessage):void
subscribeToMessages(MessageCallback):void
unsubscribeFromMessages():void
cleanup():void
<<Java Interface>>
MomPlatformClientInterface
it.cipi.wgroups.client.interfaces
publish(String,Object):void
publish(String,Object,String):void
subscribe(String):void
subscribe(String,String):void
<<Java Interface>>
KeyValueStoreClientInterface
it.cipi.wgroups.client.interfaces
set(String,Object):void
set(String,Object,String):void
get(String):void
get(String,String):void
<<Java Class>>
MomPlatformClient
it.cipi.wgroups.client
MomPlatformClient(WideGroupsClient)
publish(String,Object):void
publish(String,Object,String):void
subscribe(String):void
subscribe(String,String):void
<<Java Class>>
KeyValueStoreClient
it.cipi.wgroups.client
KeyValueStoreClient(WideGroupsClient)
set(String,Object):void
set(String,Object,String):void
get(String):void
get(String,String):void
-wideGroupsClient
0..1
-wideGroupsClient
0..1
<<Java Class>>
WideGroupsClient
it.cipi.wgroups.client
connectionTimeout_defprop: String
loggerLevel_defProp: String
log: Logger
requests: HashMap<UUID,WGMessage>
activeSubscriptions: HashMap<String,String>
connectionID: UUID
connectionTimeout: Long
buffer_size_receveid_message: int
pendingMessages: HashMap<String,LinkedList<Object[]>>
allowMultipleSubscribe: boolean
wideGroupsServerURL: String
wideGroupsServerPort: int
clientID: String
serverNodeAddress: ServerNodeAddress
clientInfo: ClientID
connectionClient: ClientConnection
performanceFunction: PerformanceFunction
_init(Properties):void
WideGroupsClient()
WideGroupsClient(String)
WideGroupsClient(Properties)
connect():void
disconnect():void
cleanup():void
createGroup(GroupID,GroupType,QOS,Collection<ClientAddress>,String):void
getGroups(GroupType):Collection<GroupInfo>
getGroup(GroupID,GroupType):GroupInfo
deleteGroup(GroupID,GroupType):void
modifyGroup(GroupID,GroupType,QOS,Collection<ClientAddress>,Collection<ClientAddress>):void
sendMessage(GroupID,GroupType,QOS,WGMessage):void
subscribeToMessages(MessageCallback):void
unsubscribeFromMessages():void
checkIsConnected():void
initLogger(String):void
getActiveSubscriptions():HashMap<String,String>
isAllowMultipleSubscribe():boolean
getConnectionID():UUID

Figure 27

5.7 WIDEGROUPS EXTERNAL COMPONENTS

WideGroups uses domains registry for discovery and local platform for adding additional processing functionalities to basic messages.

- **Domain Registry:** The WideGroups domains' registry can be WideGroups servers can register their names to public DNS or to private DNS or to other types of private registries kike the key value store registry. A common interface is provided to communicate with registries.
- **Local Platform:** A local platform is a WideGroups connectors' compatible platform that is installed in the same administrative domain of a WideGroups server. Details of the platform including the platform address, credentials, permissions and connection properties are included in local platform connectors properties and never shared with external administrative domains. WideGroups server acts as a filter for external platform's connections. Details regarding the fault tolerance of a platform inside a domain are platform dependent and directly administered with platforms' management tools. WideGroups will handle only the federation of messages sent to a local platform with external domains.

Chapter 7

6 WIDEGROUPS EVALUATION

In this section I evaluate the performance of WideGroups framework compared to state of the art federate message oriented middleware. I include a description of the test environment, a test book with the description of tests carried on and the results. To evaluate the correctness of the framework I've written and run JUnit tests. Tests described in this section are focused on performance.

6.1 WIDEGROUPS TEST ENVIRONMENT

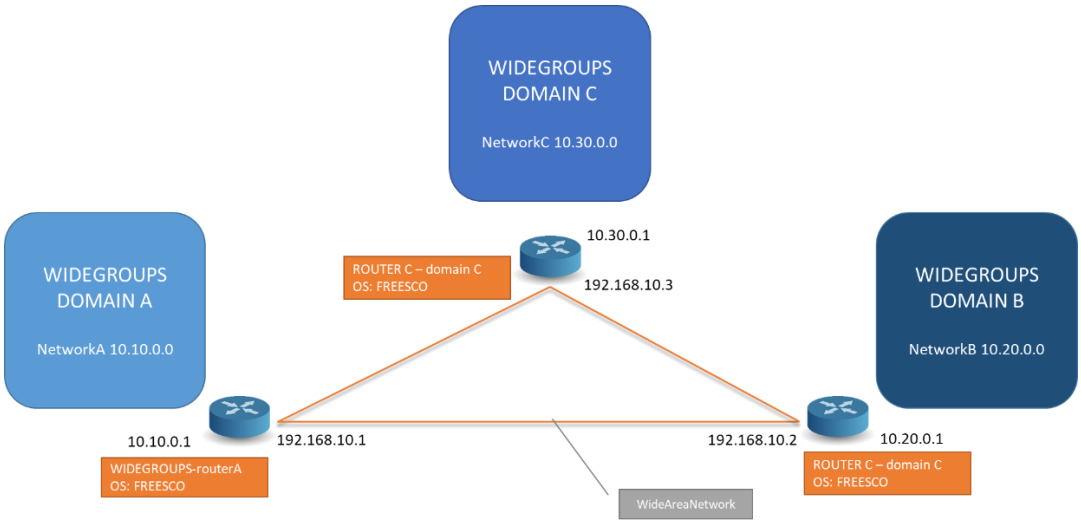


Figure 28 WideGroups test environment – domains and networks

In Figure 28 I illustrate how the WideGroups domains are mapped to test environment networks. For each domain I installed the four virtual machines illustrated in the upper part of the image below. I created three separated networks associated to three distinct domains. Each domain has its own local area network. The three domains are connected to each other through a wide area network. The test VM used to collect test results is connected to the WideAreaNetwork. Three virtual routers' interconnect different networks. Their setup contains the routing table necessary to let the three domains exchange messages.

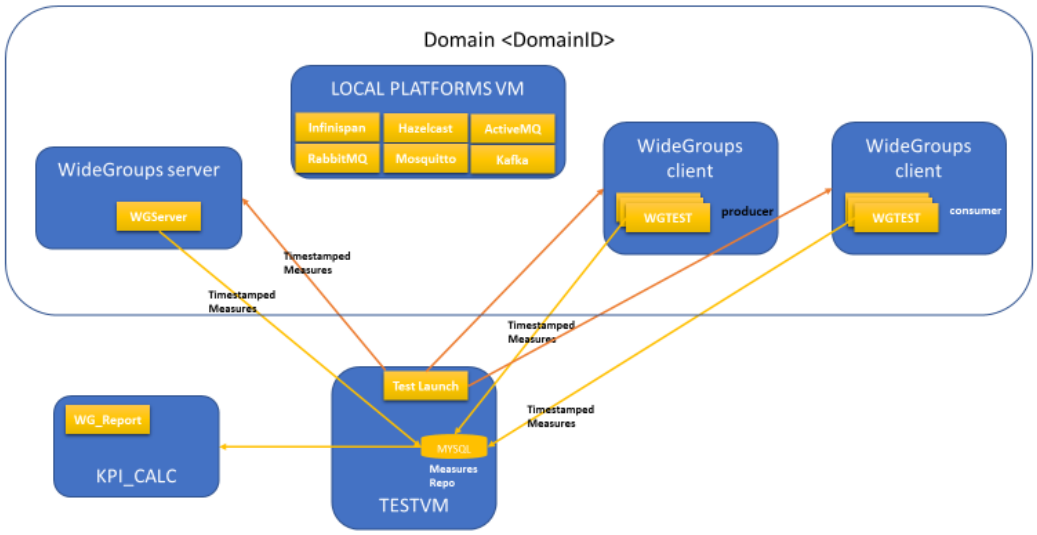


Figure 29 WideGroups test environment detailed

Table 27 Test environment dimensioning

Test Environment Dimensioning	
HOST	VMWare Esxi 5.5.0 Intel Xeon 4core @3 Ghz 64 GB RAM

For each domain 4xVM 1. Local Platforms VM 2. WideGroups Server 3. WideGroups client 1 4. WideGroups client 2	Linux x86 Fedora 4GB RAM 1x Quad-core CPU
3 virtual routers	Freesco OS 512MB RAM 1X Single Core CPU

6.2 MEASURE POINTS AND KEY PERFORMANCE INDICATORS

Measure points (M1 to M11) are indicated inside purple dots in the following schema.

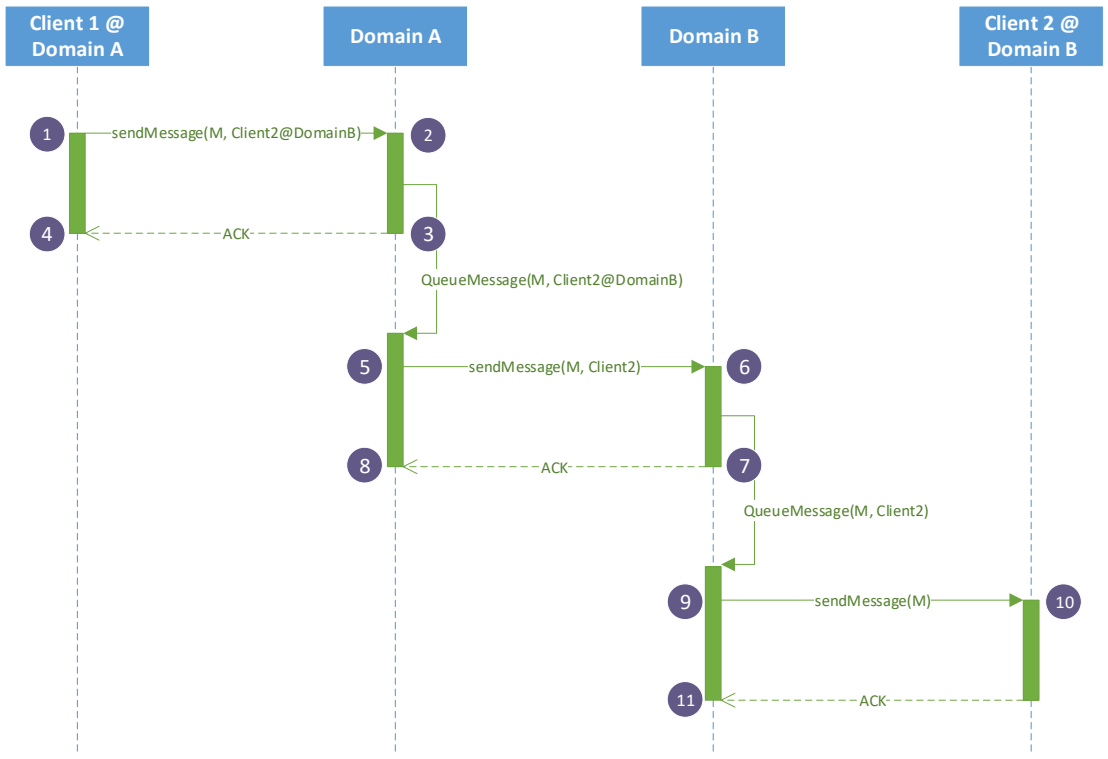


Figure 30 Test environment measure points for send message functionality

I defined the six generic key performance indicators listed in the table below.

Table 28 Performance test KPIs

KPI 1	Total time to complete the dispatching of all the messages in a test	MAX(M10) – MIN(M1)
KPI 2	Average processing time of a message between publisher and subscribers.	AVG(M10-M1)
KPI 3	Processing time of the first message between publisher and a subscriber belonging to the same domain.	MIN(<publisher.domainID>.M10) – MIN(<publisher.domainID>M1)]
KPI 4	Processing time of the first message between the publisher and the first external domain's subscriber.	MIN(<subscriber.domainID != publisher.domainID>.M10) – MIN(M1)]
KPI 5	Average of KPI3 measures for multiple messages test	AVG(KPI3) for all messages
KPI 6	Average of KPI4 measures for multiple messages test	AVG(KPI4) for all messages

6.3 PERFORMANCE TEST OF MULTIPLE MESSAGES DISPATCHING

Tests are identified with unique IDs inside the test DB. The format is Test-<P><N><S>@<I>_run-<R>, where:

- <P> is the platform identifier (e.g. W for WideGroups, K for Apache Kafka, A for Apache ActiveMQ Artemis)
- <N> is the number of messages sent in a sequence
- <S> is the size of the single message.
- <I> is the interval between two messages.
- <R> is the run identifier

For example Test-K100S@0,1_run-8 is the eighth run of the multiple messages test performed on the Apache Kafka platform with a sequence of 100 small messages (1KB) sent using an interval between two subsequent messages of 100ms.

I tested WideGroups with basic messages, Apache Kafka and Apache ActiveMQ in multiple messages dispatching at different message rates. I sent messages of three dimensions S (small size messages- 1KB), M (medium size messages- 500KB) and L (large size messages- 10MB) at different rates (0,1 seconds; 0,5 seconds and 1 second). 3 domains

The test environment setup has 4 clients:

- 1 Publisher connected to domainB,
- 3 Subscribers (1 for each domain domainA, domainB, domainC)

Publisher publish 100 messages of the same size at different intervals on a group/topic. Subscribers subscribe to that group/topic to receive published messages.

Each client is connected to its domain server. Each server/broker is federated to exchange messages with other domains.

Table 29 Test parameters

Test Parameter	**Value**
Test IDs	W100S@0,1; W100M@0,1; W100L@0,1; W100S@0,5; W100M@0,5; W100L@0,5; W100S@1; W100M@1; W100L@1; A100S@0,1; A100M@0,1; A100L@0,1; A100S@0,5; A100M@0,5; A100L@0,5; A100S@1; A100M@1; A100L@1; K100S@0,1; K100M@0,1; K100L@0,1; K100S@0,5; K100M@0,5; K100L@0,5; K100S@1; K100M@1; K100L@1;
Message size	S=1KB; M=500KB; L=10MB
Number of messages	100
Message interval	0,1s; 0,5s and 1s
KPIs analyzed	KPI1, KPI2, KPI3, KPI4, KPI5, KPI6

6.3.1 Results KPI1

KPI 1 measures the total time required to complete the test. In the tables below I report the test results (I executed different runs of the same test and analyzed the results, here and in the next chapters I report the fifth run). All the tests successfully terminate. The total time required depends on the interval among messages. For large messages, buffering is required and processing time makes the overhead time bigger than in other tests.

Table 30 KPI1 results (ms) WideGroups

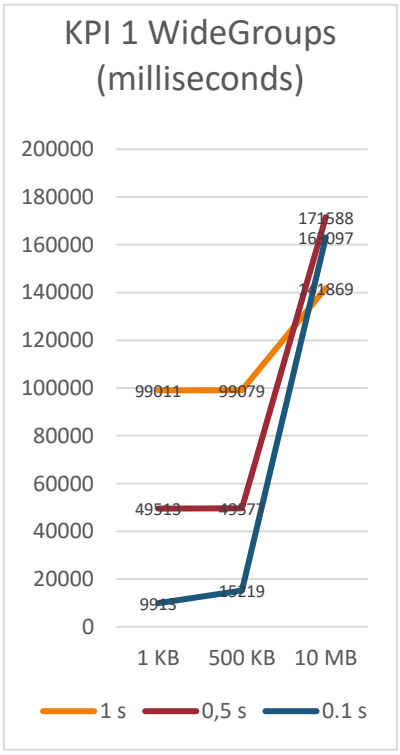
Size / Interval	1 KB	500KB	10 MB
1 s	99011	99079	141869
0,5 s	49513	49577	171588
0.1 s	9913	15219	163097

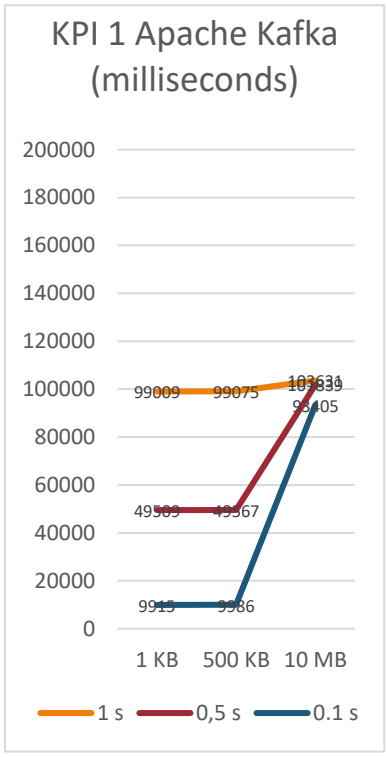
Table 31 KPI1 results (ms) Kafka

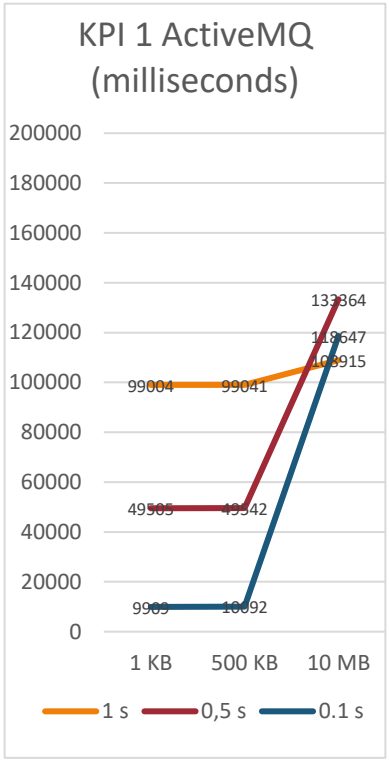
Size / Interval	1 KB	500KB	10 MB
1 s	99009	99075	103631
0,5 s	49509	49567	101839
0.1 s	9915	9986	93405

Table 32 KPI1 results (ms) ActiveMQ

Size / Interval	1 KB	500KB	10 MB
1 s	99004	99041	108915
0,5 s	49505	49542	133364
0.1 s	9909	10092	118647







6.3.2 Results KPI2

KPI 2 measures the average processing times for each message. WideGroups has better performances compared to Apache Kafka for small and medium messages. The performance degrades for large messages as WideGroups does not split messages into multiple smaller messages to provide better performances reconstructing them after processing.

Table 33 KPI2 results(ms) WideGroups

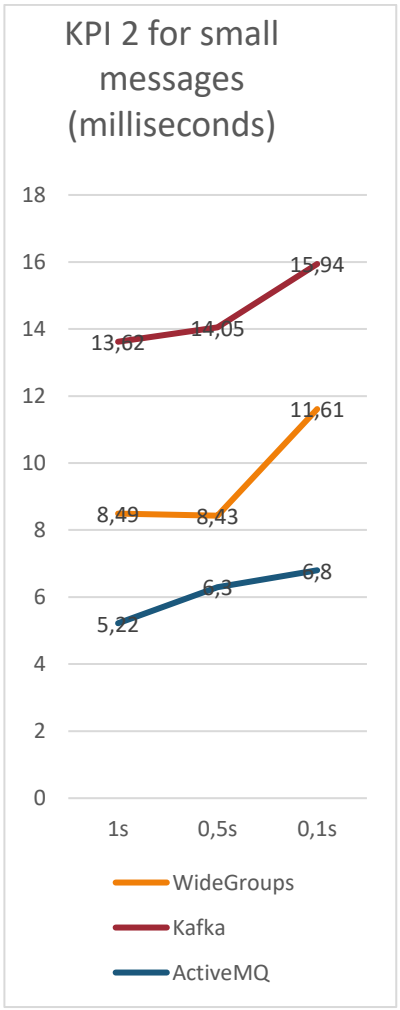
Size / Interval	1 KB	500KB	10 MB
1 s	8,49	59,86	129
0,5 s	8,43	57,4	136
0.1 s	11,61	193,92	197

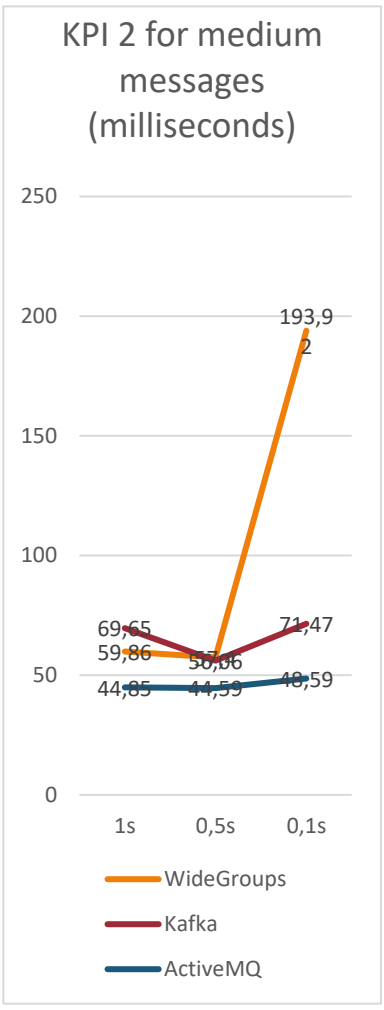
Table 34 KPI1 results (ms) Kafka

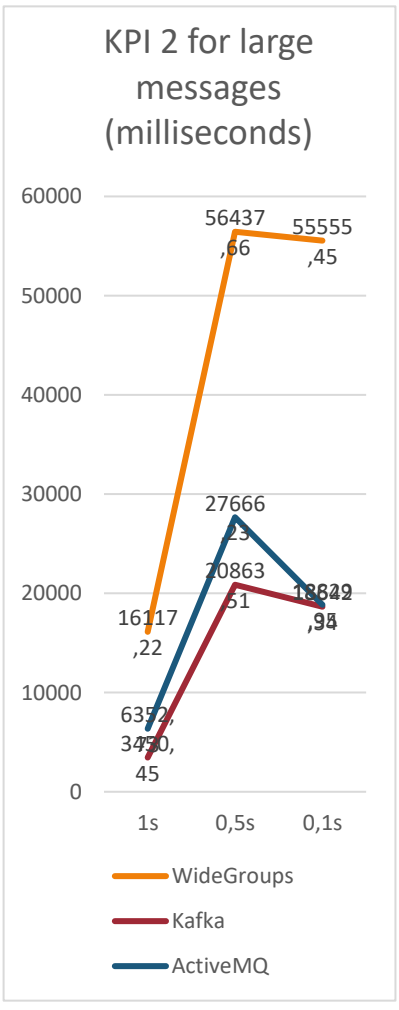
Size / Interval	1 KB	500KB	10 MB
1 s	13,62	69,65	3450,45
0,5 s	14,05	56,06	20863,51
0.1 s	15,94	71,47	18642,34

Table 35 KPI1 results (ms) ActiveMQ

Size / Interval	1 KB	500KB	10 MB
1 s	5,22	44,85	6352,73
0,5 s	6,3	44,59	27666,23
0.1 s	6,8	48,59	18829,95







6.3.3 Results KPI3

KPI 3 measures the processing time of the first message between publisher and a subscriber belonging to the same domain. Message dispatching to clients connected to the same domain does not require the server to server connection. The first message processing time is usually bigger than the processing time of the subsequent messages as the system must start thread pools and initialize queues. Kafka is slower that WideGroups and ActiveMQ. WideGroups has better performances for small messages.

Table 36 KPI3 results (ms) WideGroups

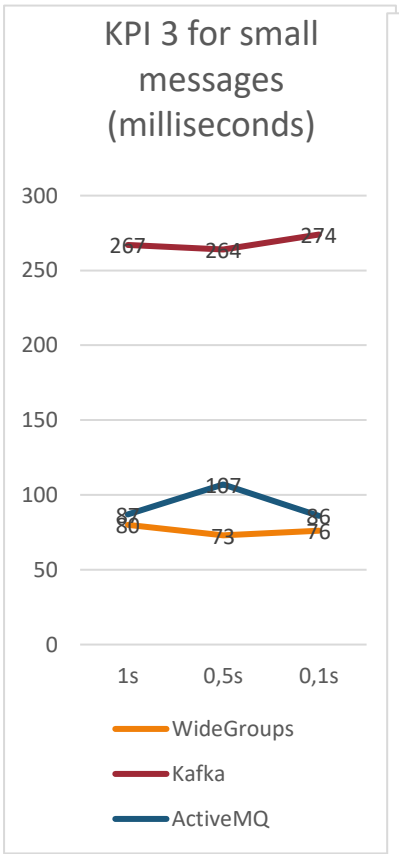
Size / Interval	1 KB	500KB	10 MB
1 s	80	59,86	1077
0,5 s	73	57,4	820
0.1 s	76	193,92	1445

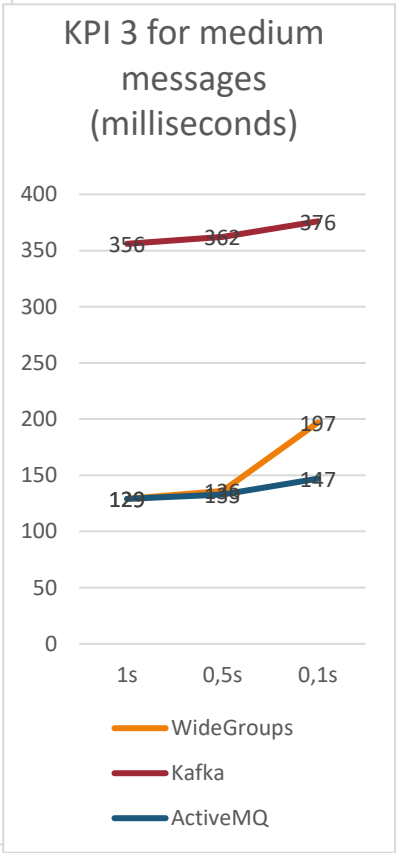
Table 37 KPI3 results (ms) Kafka

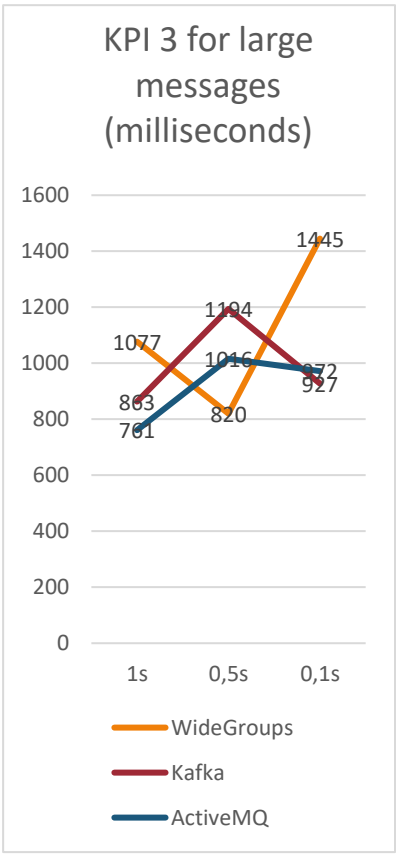
Size / Interval	1 KB	500KB	10 MB
1 s	267	356	863
0,5 s	264	362	1194
0.1 s	274	376	927

Table 38 KPI3 results (ms) ActiveMQ

Size / Interval	1 KB	500KB	10 MB
1 s	87	129	761
0,5 s	107	133	1016
0.1 s	86	147	972







6.3.4 Results KPI4

KPI 4 measures processing time of the first message between the publisher and the first external domain's subscriber. In this test you can see how WideGroups send large messages faster that the other two platforms. This test involves the message federation and results depend on the methods used. Kafka uses an external component that slows down message dispatching.

Table 39 KPI4 results (ms) WideGroups

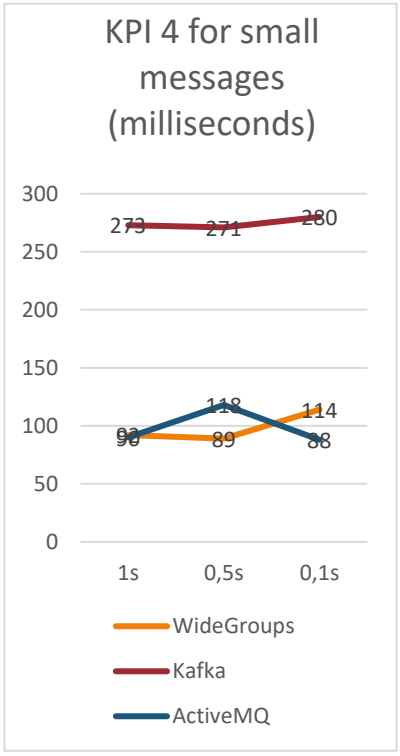
Size / Interval	1 KB	500KB	10 MB
1 s	92	183	1500
0,5 s	89	209	1632
0.1 s	114	257	1782

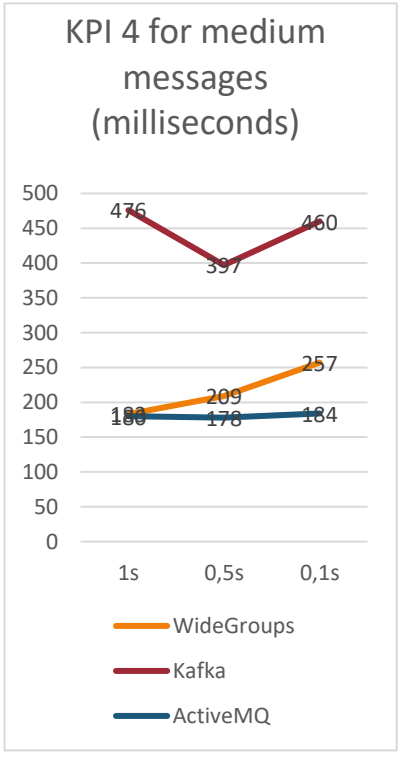
Table 40 KPI1 results (ms) Kafka

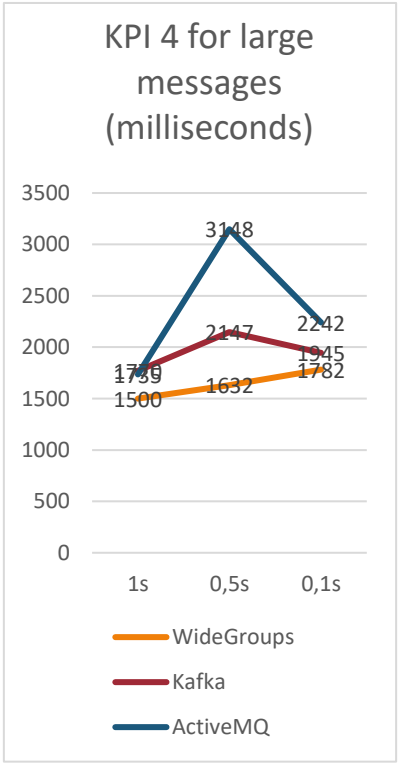
Size / Interval	1 KB	500KB	10 MB
1 s	273	476	1770
0,5 s	271	397	2147
0.1 s	280	460	1945

Table 41 KPI1 results (ms) ActiveMQ

Size / Interval	1 KB	500KB	10 MB
1 s	90	180	1735
0,5 s	118	178	3148
0.1 s	88	184	2242







6.3.5 Results KPI5

KPI 3 measured the processing time of the first message between publisher and a subscriber belonging to the same domain.. Average of KPI3 measures for multiple messages test First messages' results are worse than other messages that arrive when systems' threads are up and running and do not require initialization processes that slow down the message dispatching operations. KPI 5 measures the average processing time of messages sent to a client connected to the same publisher's domain. WideGroups results are biased as I saw that in different runs some large messages caused some buffering issues.

Table 42 KPI5 results (ms) WideGroups

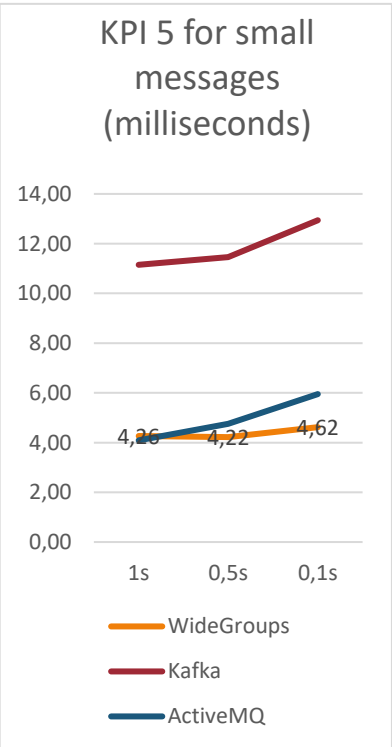
Size / Interval	1 KB	500KB	10 MB
1 s	4.26	25.38	450.88
0,5 s	4.22	25.08	15585.52
0.1 s	4.62	34.14	11736.06

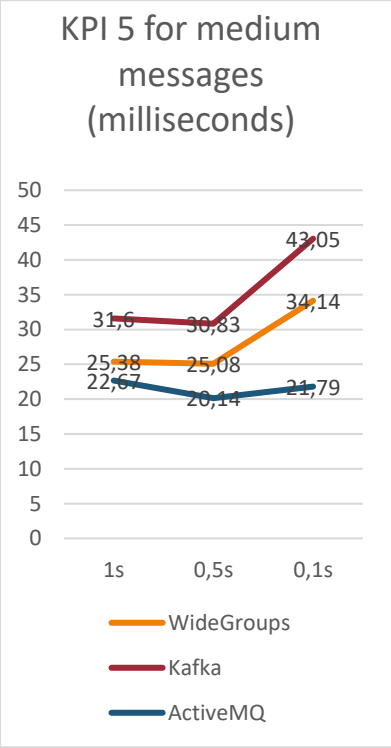
Table 43 KPI5 results (ms) Kafka

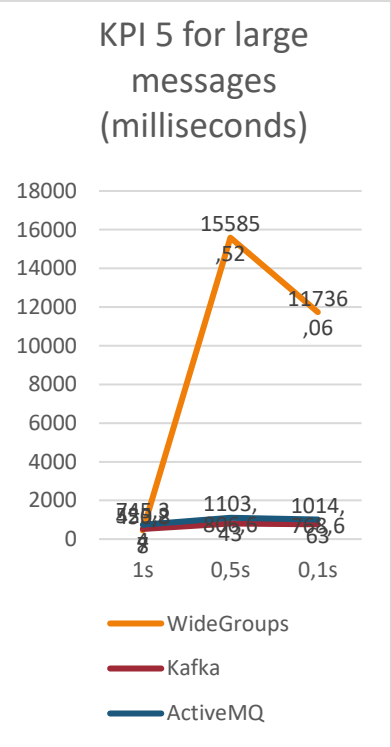
Size / Interval	1 KB	500KB	10 MB
1 s	11.15	31.6	525.27
0,5 s	11.46	30.83	806.6
0.1 s	12.94	43.05	768.6

Table 44 KPI5 results (ms) ActiveMQ

Size / Interval	1 KB	500KB	10 MB
1 s	4.09	22.67	745.34
0,5 s	4.75	20.14	1103.43
0.1 s	5.95	21.79	1014.63







6.3.6 Results KPI6

KPI6 measures the average of KPI4 measures for multiple messages test as KPI5 does for KPI3 measures.

Table 45 KPI6 results (ms) WideGroups

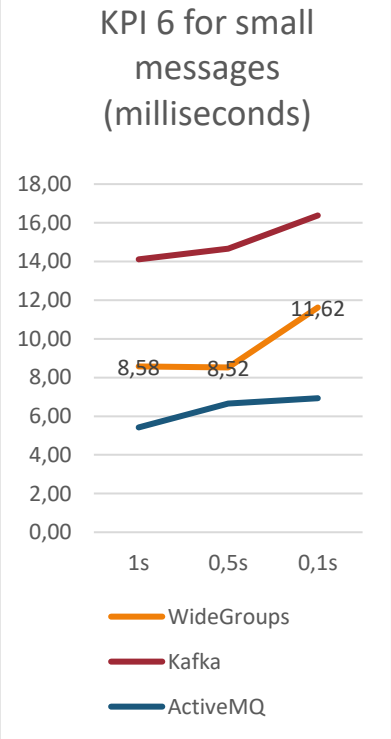
Size / Interval	1 KB	500KB	10 MB
1 s	8,58	61,6	19530,23
0,5 s	8,52	58,09	63149,89
0.1 s	11,62	92,36	51717,82

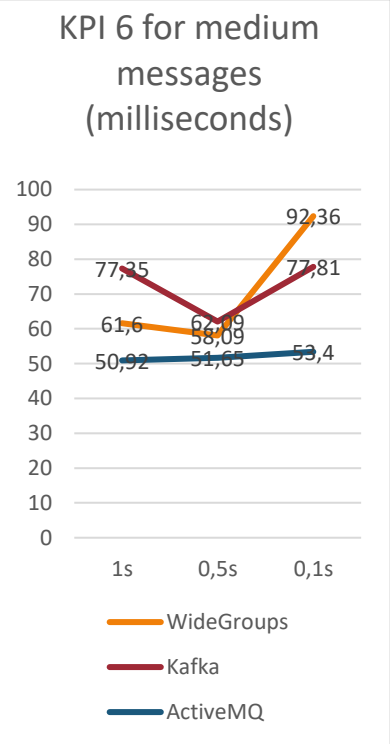
Table 46 KPI6 results (ms) Kafka

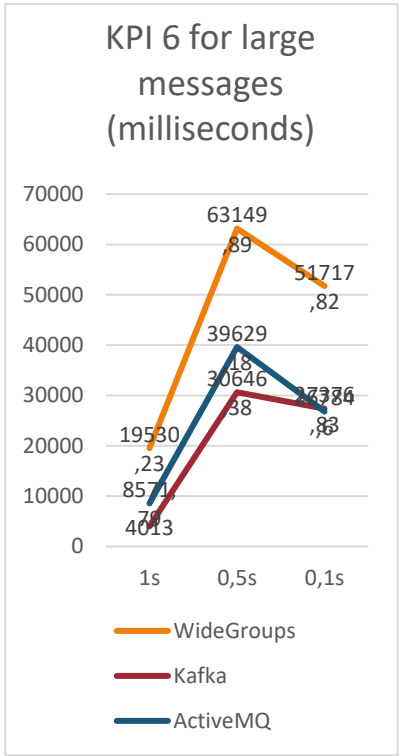
Size / Interval	1 KB	500KB	10 MB
1 s	14,11	77,35	4013
0,5 s	14,66	62,09	30646,38
0.1 s	16,38	77,81	27376,83

Table 47 KPI6 results (ms) ActiveMQ

Size / Interval	1 KB	500KB	10 MB
1 s	5,42	50,92	8571,79
0,5 s	6,66	51,65	39629,18
0.1 s	6,93	53,4	26784,6







Chapter 7

7 WIDEGROUPS USE CASES

To test WideGroups applicability I studied and designed its integration in two use cases. In both use cases cross datacenter and cross administrative domain data sharing with flexible and decentralized configurations are primary concerns. The first one is about cross datacenter information sharing among railway automation systems, the second one focuses decentralized spreadsheet elements sharing among datacenters in two different applications: spreadsheet to spreadsheet data sharing in a large consulting company and decentralized platforms to spreadsheet data sharing in port community systems and logistics.

7.1 CROSS SITE DATA SHARING AMONG RAILWAY AUTOMATION SYSTEMS

The first use case is the development of a solution for the Hitachi Rail STS integration layer cross site replication module. It designs and implements the model described in the deliverable 8.3 of the In2Rail European Project. “Each Control Center can synchronize, exchange information with another via the gateway and WAN (Wide Area Network). In this way the information that are not only specific to one particular region can be exchanged and available by other systems, components from different centers”. The Integration Layer component described in the section 2.2.3 of this thesis has a modular architecture that integrates components developed with Hitachi Rail STS

and modules from external platforms. In order to avoid vendor lock-in for specific platforms external platforms' modules are decoupled from Integration Layer modules through "connectors" and "connector APIs". The outcome of the research described in 2.2 are standard API and connectors for two platforms' categories (key value stores and message oriented middleware). Standard connectors can interact with a cluster of platforms deployed on a certain administrative domain. Clustering functionalities are platform specific and require a specific setup for each domain. This requirement is easy to accomplish as a specific administrative domain is usually maintained by a single small group of administrators. Federation of platforms is usually enabled to exchange data and messages among different administrative domains. The setup of federations can be not as easy as setting up clusters for a single administrative domain as distant sites can be administered by more than one group requiring additional management efforts. As I illustrated in previous chapters methods for platforms' federation require to touch different domains' configuration accordingly, to synchronize configuration edit operations among different sites, to open firewall ports on servers depending on the platforms' requirements and to reboot software platforms to update federation information. Moreover, different federation mechanisms could lead to potential problems because each multicast routing protocol uses different network level protocols that should not be enabled on certain networks or require to open protocol specific ports on servers causing security issues.

In order to make the federation creation and maintenance process easier I used WideGroups servers and clients to exchange standard connector APIs messages among platforms deployed in different administrative domains. I integrated the WideGroups client in platform connectors and added a cluster of WideGroups servers for each administrative domain to filter messages sent to software platforms and create groups of external platforms' nodes.

7.1.1 Primary - Backup architecture with asymmetric groups

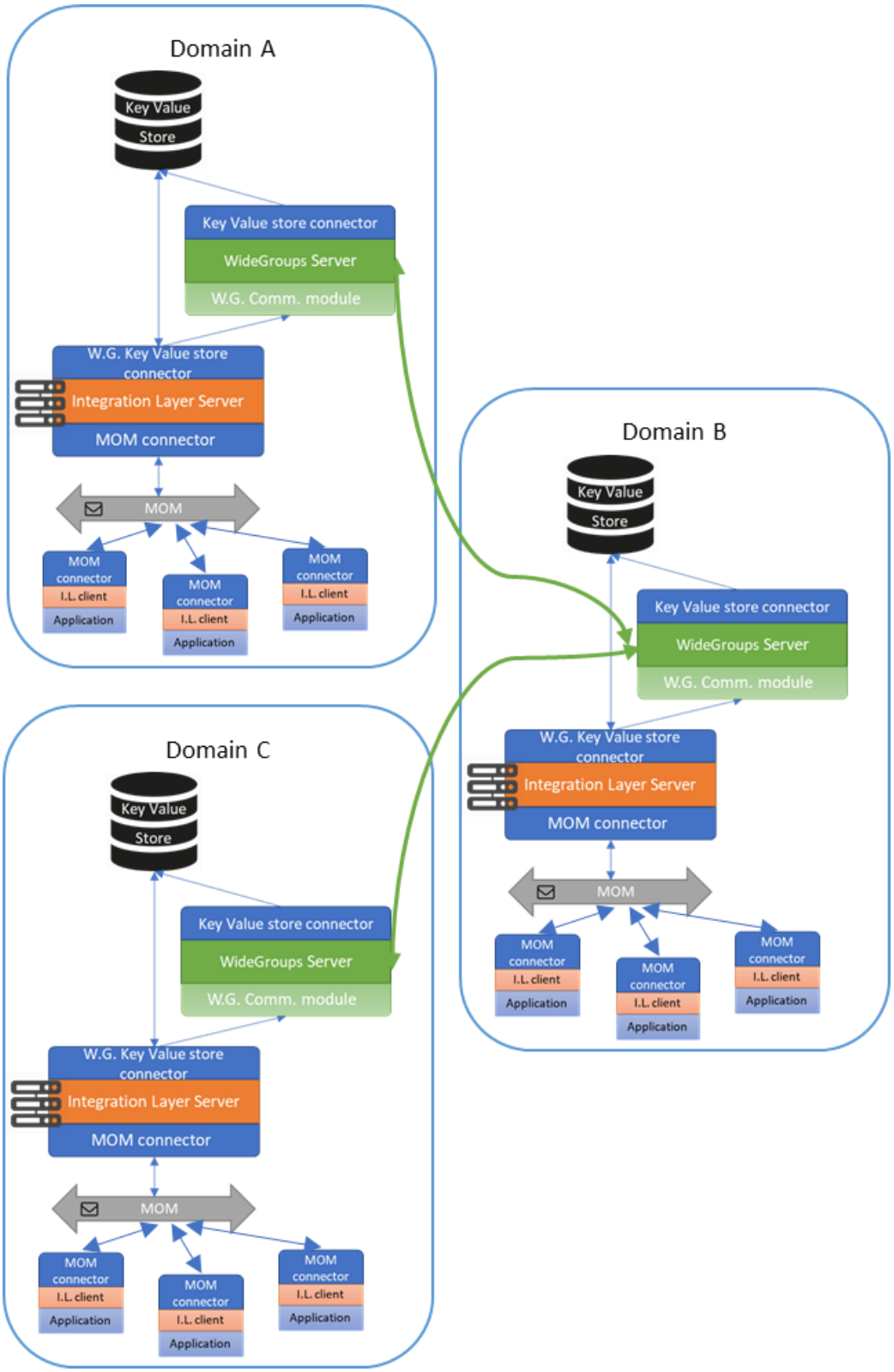


Figure 31 Use case 1 primary backup architecture

The objective of this use case is twofold:

1. to test the federation of “key value store platform messages”
2. to test asymmetric groups.

Platform messages exchange standard key value store connector APIs commands on an asymmetric group to backup data among three remote sites. In this configuration there is always one site acting as the master cluster and two sites acting as backup sites. In the case of failure of the master cluster, the slave cluster with the smaller IP address takes over and becomes the new master. The slave clusters must maintain all copies backed up for fault tolerance inside their repositories in order to start working faster in case of main datacenter failures. The Integration Layer uses components off the shelf to store data used by its servers. I integrated WideGroups clients inside the Integration Layer’s Key Value Store connector. At KeyValue store delegate startup a group of servers belonging to different domains is set using an asymmetric group. Platform messages regarding these group are replicated on the remote cluster that join the group at startup.

7.1.1 Load Balancing architecture with asymmetric groups

In the second setup I tested symmetric groups interconnecting three sites. The messages are published on dynamic groups that change their participants based on the train position. The objective of the second use case is threefold:

1. to test the federation of “mom platform messages”
2. to test symmetric groups.
3. To test dynamic groups

Different application can publish messages on the same group. Groups can change participants based on the train position in order to let different participants start receiving updates regarding a train position. This use case demonstrated how WideGroups supports hot reconfiguration and service continuity.

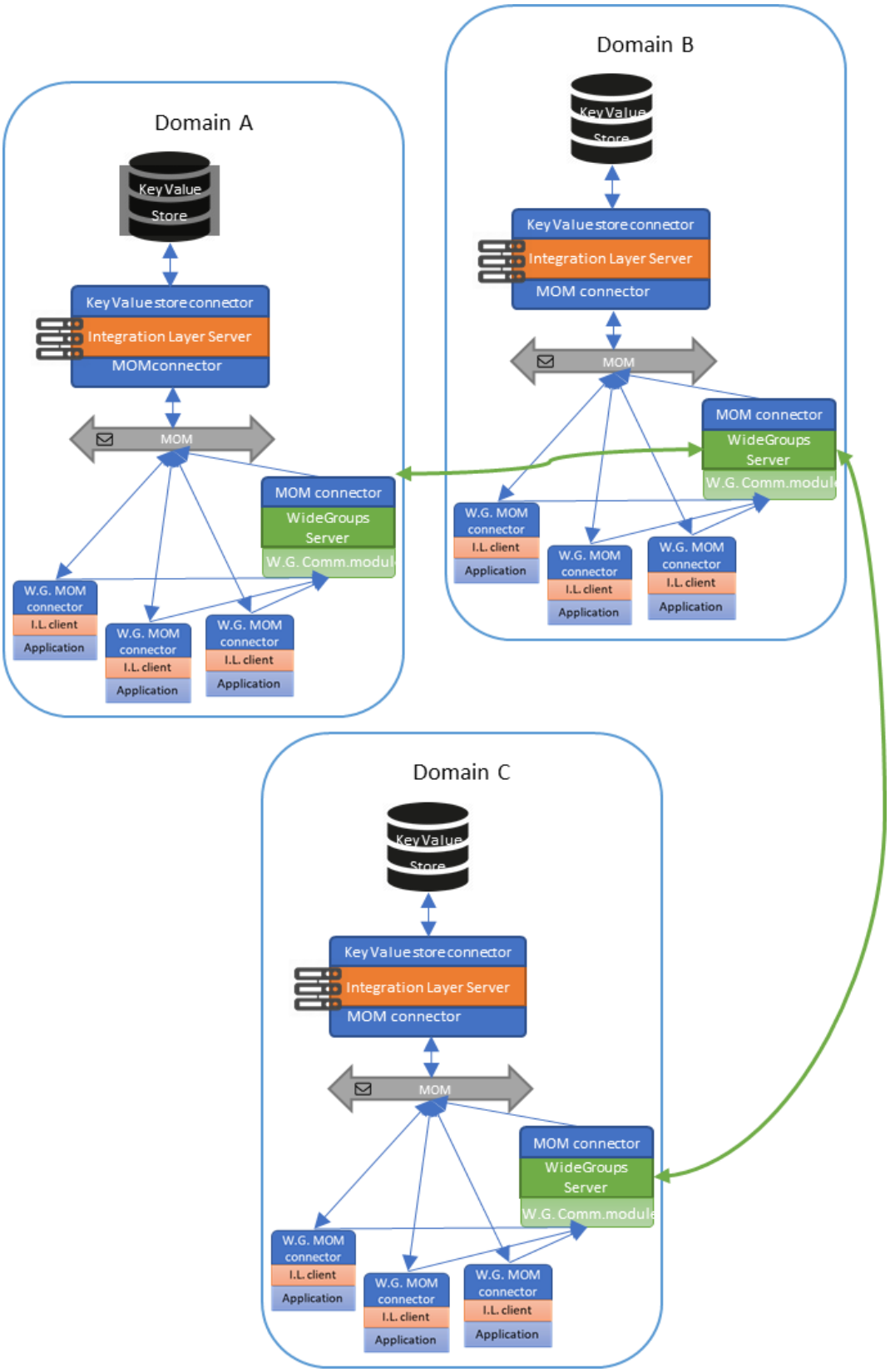


Figure 32 Use case 2 load balancing architecture

7.2 CROSS SITE SPREADSHEET DATA SHARING

The second use case is direct towards the decentralization of Spreadsheet Space's control plane. Spreadsheet Space control plane server is a group of off-cluster nodes (nodes whose owners are never involved in a collaboration). Their owner cannot look at data shared for privacy and security reasons. Therefore it never receives spreadsheet elements' data even if they are encrypted. Spreadsheet Space Server component implements four modules: an event manager, a sharing manager, a collaborators manager and security module. I use WideGroups to decentralize this component distributing it in different administrative domains. The Spreadsheet Space control plane is handled by the Spreadsheet Space server. Thanks to WideGroups I designed a new version of the cluster of zero knowledge cloud servers to distribute them among the different end users' administrative domains, thus distributing not only the data storage part and the co-editable spreadsheet element reconstruction, but also the core services. Spreadsheet Space evolves from a single space to a federation of distributed private Spreadsheet Spaces. I used Asymmetric groups for view image exposition and symmetric groups for co-authoring operations.

7.2.1 Spreadsheet Space use cases and user acceptance evaluation

With the support of the Joint Research Center on Computing Platforms (CIPI) I carried on user acceptance evaluations [7] and tests of the Spreadsheet Space platform in various application domains which together involved nearly 100 users. These tests has been useful to understand what are user requirements and what behavior and performance they expect from a completely decentralized system for Spreadsheet data sharing. In this section I summarize these experiences made using Spreadsheet Space to speed up and optimize enterprise processes based on the interaction patterns identified in section 2.1.1.1. More specifically I describe two use cases: A. human resources management in a consulting company, B. data collection and sharing about custom export declarations from the Port of Genoa. For both use cases I describe the process, the interaction patterns used for collaborative analytics within these processes and how I applied Spreadsheet Space and its functionalities to implement such interaction patterns. In all the use cases each user used his own Microsoft Excel

installation (2010, 2013 and 2016 versions) running on his own PC (Windows 7 or 10). In order to make his Excel installation "Spreadsheet Space enabled" each user autonomously retrieved and executed the Spreadsheet Space installation wizard to install both the Spreadsheet Space addin inside Microsoft Excel and the "notifier" client application. For use case A CIPI installed an "on premises" version of the Spreadsheet Space server in the company's private cloud. For use case B CIPI and I used the public Spreadsheet Space server. CIPI installed a view server for each use case. Additional view servers are installable on demand by users and each user can register his personal view server on the network. For use case B I developed a Java application to filter data from an external platform and I used the Spreadsheet Space Java SDK to expose data to selected users.

7.2.1.1 Human resources management in a consulting company

I tested spreadsheet to spreadsheet data sharing applied to the assignment process of human resources' working days to active projects. This test has been conducted within an international consulting company where human resources are organized in people units as usual in such companies. Each people unit has its own manager. Each people unit can have multiple active projects and each project has a manager that must employ a subset of consultants from the unit he belongs to. **People unit managers** and **project managers** have to keep track of the amount of working days their human resources spent in active projects and organize their future allocation. While the resources final allocation is registered on the company's ERP, the initial allocation proposal, as well as its negotiation and planning is a trial and error process always supported with spreadsheets. According to interviews and surveys carried on during the testing phase, spreadsheets give managers more flexibility than the company's ERP. They provide an easy-to-use and fast tool to take decisions and analyze tabular data within their own end user programmed spreadsheets. People unit managers and project managers register and analyze human resources involvement in active projects inside their own spreadsheets. During this process portions of managers' private spreadsheets are exchanged frequently to consolidate information and decisions. Project managers compile a table with standard fields including information about the employees and their skills, the geographical area, information of the project in which they are involved

and details about their specific work package and the current state assigning the amount of requested days for each month. Each unit's project managers send the table compiled with the human resources work hours report and the requests for the next month to their specific people unit manager. People unit managers collect tables from their unit's project managers and start proposing changes in the work hours planning based on the global view of project managers requests, analysis of idle human resources and other information coming from vacation plans and scheduled training activities. Once project managers and unit managers agree on the planning, unit managers send the final plan to the practice director that must approve the final allocation for loading into the company's ERP. Before Spreadsheet Space adoption this process was carried on exchanging emails among particiapants. This was time consuming and frequently lead to errors and misconceptions. I applied Spreadsheet Space to speed up the tables' exchange process and reduce errors during spreadsheet elements data sharing.

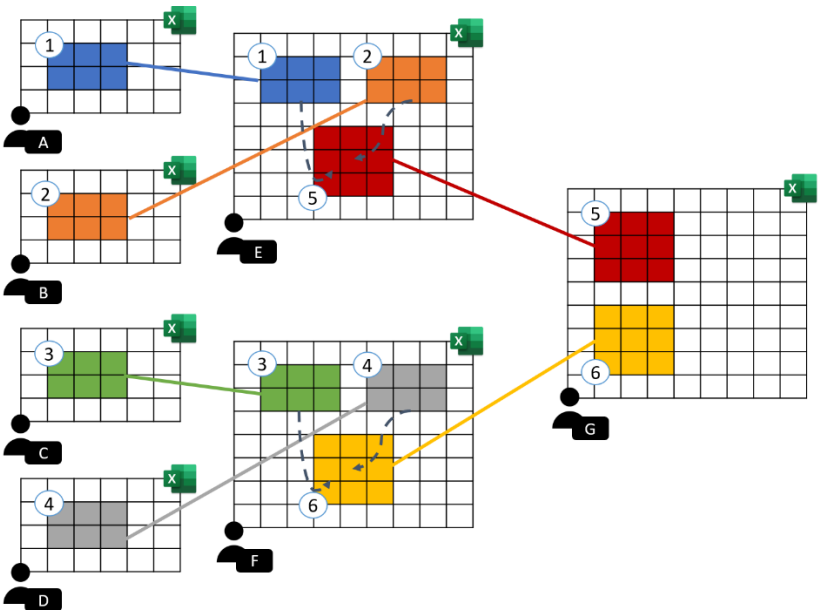


Figure 33 Spreadsheet Space enabled collaborative analytics process in the consulting company.

Figure 33 depicts an example of interactions in a process involving four project managers (A,B,C and D) divided in two people units, two people unit managers E and F and the practice director (G). Each grid represents a different spreadsheet owned by his creator and stored in his personal PC. Colored grids tagged with numbers are the

shared spreadsheet objects. I used three interaction patterns implemented in Spreadsheet Space: collaborative editing (grids number 1, 2, 3 and 4), data distribution (grid 5 and grid 6 on E and F) and data collection (grid 5 and grid 6 on G). Below I describe the creation, editing and sharing process of these objects.

- **Usage of collaborative editing**: I applied the Spreadsheet Space collaborative editing functionality to the process of exchanging human resources' working days planning requests among people unit managers and project managers. Grids number 1, 2, 3 and 4 are collaborative editable tables. The collaborative editing process is always initiated by a people unit manager. A people unit manager opens its Spreadsheet Space enabled Microsoft Excel and creates one table for each project manager in his unit. For example, people unit manager E creates the grid 1 and the grid 2. Each table has a column per information requested to the project manager. He subsequently creates a collaborative editing with the specific project manager for each table. For example, people unit manager E shares the grid 1 with project manager A and the grid 2 with project manager B selecting the collaborative editing mode. Each project manager imports the dedicated table in his spreadsheet and starts filling it with the required information. Unit managers collect information coming from different managers in the same spreadsheet and consolidate the final plan, exchanging edits with the project managers. In Fig. 6 the final consolidated plan is written in the grid 5 for people unit manager E and grid 6 for people unit manager F. The grid 5 is the consolidated plan for the people unit including managers A, B and E, while the grid 6 is for the people unit including managers C, D and F.
- **Usage of data distribution**: Unit managers expose a view of a table containing the collected information from the spreadsheet where they collaboratively analyzed data using the previously described collaborative editing sessions. The view is exposed to the practice director once the decision process is terminated. For example, people unit manager E creates the grid 5 where he stores the consolidated results, he then exposes it as a view to the people unit manager G.

- **Usage of data collection**: the practice director collects images of views from all the unit managers in its Microsoft Excel to analyze and visualize them before inserting the final plans in the company's ERP. In the figure above the practice director G imports the grids 5 and 6 that are the images of the human resources allocation plan consolidated by people unit managers E and F. Practice Director G will finally insert the consolidated plans in the company's ERP.

7.2.1.2 Statistics' analysis about custom export declarations from the Port of Genoa

The port industry involves different managers from carriers, shippers and agencies that rely on PCS (Port Community Systems) services to get information about import and export processes and to control transport activities. Managers perform data analysis with end user programs, the spreadsheets, and need live and constantly updated data from PCS for business analytics. In order to share these information for end user analytics, PCS and other platforms usually allow to extract and download data in spreadsheets format. Those data are usually shared via email without the possibility to have an up to date data source for end user programmers. Another frequently adopted solution is to share platform credentials with properly defined access control policies to limit uncontrolled access to all DB tables or storage platform's data. Once access is granted, external data access to data is possible using queries from spreadsheet applications, but at the cost of losing data consistency and reactivity (the capacity of a program to react to data changes) because data are not automatically updated as soon as they change. Another possible solution is to implement ad-hoc web services to satisfy each customer requirements. From the system administrators point of view the proposed solutions are time consuming and can create privacy and security issues. From the customers point of view are complex and not user friendly since they require to handle connections to multiple platforms or services with different formats and endpoints. I applied Spreadsheet Space to enable enterprise data integration of these data coming from port industry platforms like PCS in order to simplify the sharing process, to allow real-time updates and to create an always on connection among customers' spreadsheets and platforms' data. Spreadsheet Space's enterprise application data integration and exposition pattern can help both PCS administrators and users. Platforms' administrators expose data from their platforms using the

Spreadsheet Space APIs/SDK and end user programmers import read only data exposed to them.

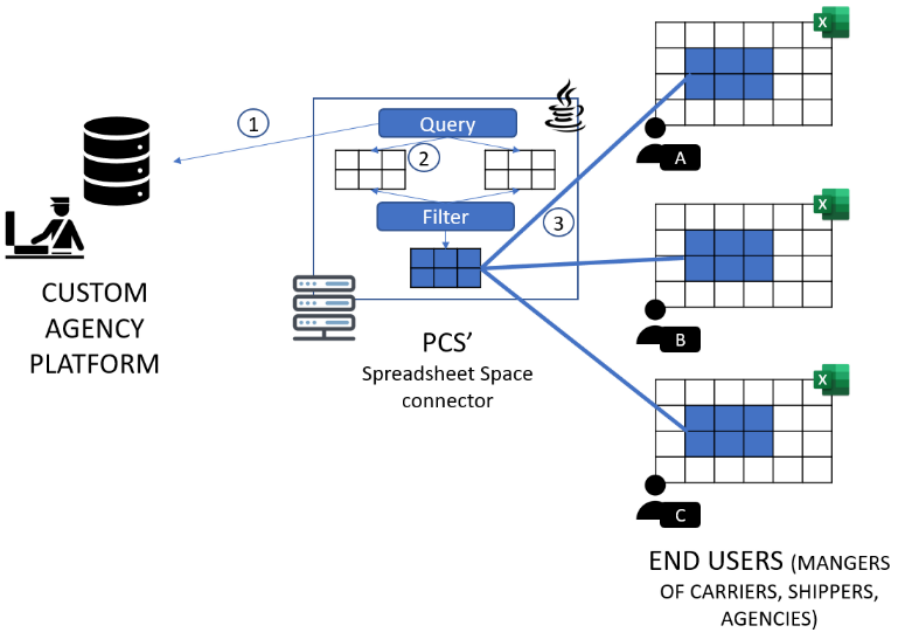


Figure 34 Spreadsheet Space enabled collaborative analytics process in the port community system.

CIPI teamed up with PCS administrators to develop a Spreadsheet Space connector to let them easily share tabular data with selected users to export statistics about custom export declarations from the Port of Genoa (Fig. 7). PCS implements a continuous query (1) to get updated data from the custom agency platform. It subsequently stores and filters data (2) classifying goods, their weight and value. The application integrates the Spreadsheet Space SDK to expose a view (the colored grid) to selected users (3). Enabled users can import up to date information in their spreadsheets (the colored grid) to perform statistical analysis, integrate data with information imported from other sources and eventually share results from their analysis to other end users.

7.2.1.3 Lessons Learned from Use Cases.

In the first use case I analyzed the usage of spreadsheet *collaborative editing* and fine grained *data collection/exposition* functionalities inside an industrial use case. Then I compared our model with state of the art methods for spreadsheet coediting available in spreadsheet editing platforms. Recently, spreadsheet file editing platforms integrated native groupware applications that lets editors join co-authoring shared sessions.

Remote productivity suites were the first to implement the spreadsheet sharing model transforming the multiuser remote productivity suites to groupware applications. Google Sheets (https://www.google.com/sheets/) and EtherCalc (https://ethercalc.net/) lead the transformation with Excel Online (https://office.live.com/start/Excel.aspx) following their approach. Remote productivity suites do not use local resources and filesystems: browsers run a client side web application that remotely calls a server-side office suite application that exposes its methods through a web service. Remote productivity suites are freely available online, but at the cost of file content delocalization and less privacy: files are stored in the cloud and end users lose control on savings and versions. Local productivity suites' classic spreadsheet editing software like Microsoft Excel gained a native support to spreadsheet coauthoring only recently. Microsoft Excel enables coauthoring with file delocalization in the cloud. These platforms weakly support the industry use case I analyzed for two reasons: **data delocalization in public clouds** and **impossibility to share just a small portions of data inside a spreadsheet**. File and data delocalization is not suitable for collaboration on sensitive enterprise data. There are on premises versions of these platforms like the Office Online Server, but they enable collaboration only for local area networks. A wide area network connection relies on the centralized cloud platforms described before. All control plane and data plane modules are handled by off-cluster third party nodes for both remote and local productivity suites. Platforms analyzed implement only a spreadsheet file sharing model that shares data at file level granularity. Data distribution is possible only using a coarse grained approach sharing the entire spreadsheet in read only mode. Data collection is not possible as users cannot import data from different spreadsheets and integrate them in one single spreadsheet. A better choice would be spreadsheet references, reactive links to external data automatically updated that work as pointers to the content of another cell in the same spreadsheet or in another spreadsheet. References implement fine grained data access, but unfortunately they are possible only for spreadsheet elements stored in files located in the same file system and are spreadsheet editing software dependent. End users can exploit the spreadsheet reference functionality for fine grained data exposition and data collection, but it is not an easy and fast task. For data exposition, if the spreadsheet

contains data that must not be read by other sharing participants, the owner must copy and paste only the necessary data to a brand new spreadsheet externally referencing only the cells that she wants to share. Data collection requires to import spreadsheets separately and then reference selected cells from external spreadsheets. The import range solution in Google Sheets is another possible way for spreadsheet data sharing, but it is limited to Google sheets' files delocalized in the Google cloud storage platform.

I argue that the sharing process is not correctly modeled and left to end user initiatives limited by enterprises' restrictions necessary to stop uncontrolled sensitive data sharing (e.g.: public cloud platforms such as Google Sheets or Excel Online are often blocked to prevent third party data access to private data as data are never end to end encrypted). As a consequence, a common way to share spreadsheet data is to share the entire file sending a copy via email to collaborators or uploading it on a private shared repository. Both methods can generate data inconsistencies causing the so-called "multiple versions of the truth" as they make users responsible for sharing updates and people working on that file could not receive updates as soon as they occur.

In the second use case I analyzed *enterprise application data integration*. Spreadsheets implement technologies to fill spreadsheet data elements with external data without sharing the entire file to enable collaborative analytics. External data access is possible using queries, but at the cost of losing data consistency and reactivity (the capacity of a program to react to data changes) because data are not automatically updated as soon as they change. Queries are also read only connections and do not permit the sharing of local results after analysis to other end users and platforms. There are also proprietary add-ins to enable synchronization between a spreadsheet and external storage platforms like SQL and NoSQL DBs used to maintain data consistent across different spreadsheets and external software. They guarantee data external access even if the spreadsheets or software manipulating those data are temporarily offline. Typically those platform connectors are spreadsheet editing software independent, but are storage platform dependent. Platforms' proprietary add-in can implement data plane on collaborators' nodes, but they require to share platform credentials among all the collaborators giving complete and uncontrolled access to all DB tables and storage

platform's data. A complete add in that provides both read and write access to external platform's data is the MySQL extension for Microsoft Excel that provides a native support to connect readable and writable data to an external MySQL table in order to give external access to that spreadsheet element to other Microsoft Excel spreadsheets or to other platforms' software. This approach is fully centralized and it is possible to import data from only one database at a time reducing data collection possibilities. Data are stored in a central database and the synchronization protocol uses DB transactions to commit updates on the database thus giving non real time editing sessions but with strong consistency.

I collected feedback from three different types of actors during our tests: the end users, the developers and the companies' IT managers. I received good feedback from involved end users that appreciated the possibility to share spreadsheet elements from cell to sheet granularity, the possibility to control range updates and the native support for different versions of Microsoft Excel on Windows. On the other hand I observed an initial discomfort in adopting Spreadsheet Space functionalities in their processes and requests for a complete addin also for MacOS (there is only a beta version currently available with reduced functionalities). I surveyed the developers that built the Spreadsheet Space connector for use case B. They appreciated the variety of SDK Spreadsheet Space provides (Java, C#, Javascript and REST APIs) and found useful and easy to understand the available documentation. It was easy to integrate SDKs to develop connectors to read and write spreadsheet elements' data from different software and spreadsheet editing platforms. I report some negative comments about the necessity to develop ad hoc connectors for external platform from scratch and the requests for connectors' templates for the most used technologies from which to start (like MySQL servers). Companies' IT managers recognized the possibilities offered by a fully distributed architecture. They also appreciated the absence of a unique central storage platform and the possibility to save data encrypted into their administrative domains with the separation of the control plane form a private data plane that enables scenarios that are currently not supported by existing platforms. Both developers and IT managers understood the possibilities that Spreadsheet Space offers to synchronize spreadsheet elements' data among different platforms without the necessity to share

central platform credentials and give complete access to an external data storage platform.

Chapter 8

8 CONCLUSIONS

In this thesis I presented WideGroups, a decentralized framework to share data among different administrative domains without a-priori configuration of platforms' nodes federations. Dynamic configuration strongly relies on discovery techniques inspired to XMPP and SMTP protocols and methodologies. WideGroups exploits these techniques to discover completely decentralized servers and establish ad hoc connections to exchange data. My approach derives some of its concepts and techniques from application layer multicasting algorithms and systems. There is a correspondence between overlay networks used to connect servers running application layer multicasting algorithms and WideGroups servers' networks in the way they can be created at runtime and adapted to efficiently exchange messages. WideGroups servers run an application layer multicast protocol deployed at infrastructure-level and work as rendezvous points for WideGroups clients organizing themselves in overlay networks. WideGroups adopts multicasting techniques similar to the ones used by group communication frameworks, such as JGroups, adapted to work in cross administrative domain scenarios over a wide area networks. Overlay networks are formed to exchange messages in groups that can be asymmetric or symmetric. Asymmetric groups have one producer and multiple consumers, while symmetric groups let multiple producers be active at the same time. Groups exchange two types of messages: basic messages, used to let clients belonging to different administrative

domains communicate without a centrally administered platform, and platform messages, used to add functionalities to groups such as publish subscribe and key value storage. Publish Subscribe adds topic based filtering to groups with standard wildcarding. Key value storage provides the possibility to share portions of storage platforms or filesystems to host data exchanged with platform messages without deploying a central cluster. Interaction with external components happens through standard APIs and standard connectors that abstract away the vendor specific APIs implementations to interact with the locally installed platforms. Therefore, WideGroups can interconnect platforms of the same type implemented by different vendors with different APIs. Moreover, thanks to its decentralized nature it can connect them in a controlled and decentralized way even if they are running in different administrative domains. This approach is particularly useful when applications handle sensitive data and need to avoid third party data access, especially in enterprises. The framework has been tested and compared with federated message brokers used to exchange messages among different administrative domains. It provides message propagation times comparable to federated brokers. Finally I focused on use cases where the WideGroups usage proved advantages compared to other candidate enabling technologies and methods. The first use case focused on data sharing among railway automation systems belonging to different administrative domains. In these systems sensitive data regarding different areas of the railway networks are administered by separate entities. In the second use case I showed how I distributed the control plane module of the Spreadsheet Space platform to distribute Spreadsheet Space servers in enterprise use cases. I presented two use cases and the correspondent user acceptance evaluation reports that confirm how Spreadsheet Space is particularly interesting for end user programmers that use spreadsheets for collaborative analytics of sensitive industrial data. The WideGroups prototype deserves more work to increase its performance and security. I already started developing a faster version of the framework with a better parallelization of its internal threads. The framework designed during this research project has interesting characteristics that can make it a good choice in enterprises to avoid vendor lock-in on specific federated message/storage middleware. Moreover, it speeds up integration and configuration of cross

administrative domains data sharing providing an efficient, scalable and secure solution. This framework answers to the higher demand of administrative scalable distributed systems in these latest years that often guided enterprises to choose distributed ledger technologies (DLTs) trying to adapt them to fit efficiency, privacy and flexibility requirements with little success.

REFERENCES

[1] Bonnie A Nardi. 1993. A Small Matter of Programming: Perspectives on End User Computing. MIT Press, Cambridge, MA, USA.

[2] Andrew J Ko, Robin Abraham, Laura Beckwith, et al. 2011. The State of the Art in End-user Software Engineering. ACM Comput. Surv. 43, 3: 21:1–21:44.

[3] Christopher Scaffidi, Mary Shaw, and Brad Myers. 2005. Estimating the numbers of end users and end user programmers. Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on, 207–214.

[4] Brad A Myers, Margaret M Burnett, Andrew J Ko, Mary Beth Rosson, Christopher Scaffidi, and Susan Wiedenbeck. 2010. End User Software Engineering: CHI 2010 Special Interest Group Meeting. CHI '10 Extended Abstracts on Human Factors in Computing Systems, ACM, 3189–3192.

[5] S. Erwig, "Software Engineering for Spreadsheets," IEEE Software, vol. 26, no. 5, 2009, pp. 25−30.

[6] Gilbert, Seth, and Nancy Lynch. "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services." Acm Sigact News 33.2 (2002): 51-59.

[7] Maresca, Massimo. "The Spreadsheet Space: Eliminating the Boundaries of Data Cross-Referencing." Computer 49.9 (2016): 78-85.

[8] ChengZheng Sun "Operational Transformation and its Application to Microsoft Office Suite," Talk at Microsoft Research Asia (Beijing, China). April 15, 2010.

[9] Ellis, Clarence A., and Simon J. Gibbs. "Concurrency control in groupware systems." Acm Sigmod Record. Vol. 18. No. 2. ACM, 1989.

[10] Lamport, Leslie. "Time, clocks, and the ordering of events in a distributed system." Communications of the ACM 21.7 (1978): 558-565.

[11] Colin J. Fidge (February 1988). "Timestamps in Message-Passing Systems That Preserve the Partial Ordering" (PDF). In K. Raymond (Ed.). Proc.

of the 11th Australian Computer Science Conference (ACSC'88). pp. 56–66. Retrieved 2009-02-13.

[12] Mattern, F. (October 1988), "Virtual Time and Global States of Distributed Systems", in Cosnard, M., Proc. Workshop on Parallel and Distributed Algorithms, Chateau de Bonas, France: Elsevier, pp. 215–226

[13] Sun, Chengzheng, et al. "Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems." ACM Transactions on Computer-Human Interaction (TOCHI) 5.1 (1998): 63-108.

[14] https://drive.googleblog.com/2010/09/whats-different-about-new-google-docs.html

[15] Wang, David, Alex Mah, and Soren Lassen. "Google wave operational transformation." Whitepaper, Google Inc (2010).

[16] Ongaro, Diego, and John K. Ousterhout. "In search of an understandable consensus algorithm." *USENIX Annual Technical Conference*. 2014.

[17] Maresca, M., Parodi, A., Camera, G., & Baglietto, P. (2017). Environment Sensors Measures Processing: Integrating Real-Time and Spreadsheet-Based Data Analysis. In *Adaptive Mobile Computing* (pp. 29-46). Academic Press.

[18] Camera, G., Baglietto, P., & Maresca, M. (2018, July). Shared Access to Spreadsheet Elements for End User Programming. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)* (pp. 1571-1572). IEEE.

[19] Camera, G., Baglietto, P., & Maresca, M. (2019, October). A Platform for Private and Controlled Spreadsheet Objects Sharing. In *2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC)* (pp. 67-76). IEEE.

[20] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, *MIT Press, McGraw-Hill Book Company*, 1993.

[21] Y. K. Dalal and R. Metcalfe. Reverse Path Forwarding of Broadcast Packets. Communications of the ACM, 21(12):1040–1048, 1978.

[22] Ballardie, Tony, Paul Francis, and Jon Crowcroft. "Core based trees (CBT)." *ACM SIGCOMM Computer Communication Review*. Vol. 23. No. 4. ACM, 1993.

[23] Deering, S. (1989). RFC1112: Host extensions for IP multicasting.

[24] Fenner, W. (1997). RFC2236: Internet Group Management Protocol.

[25] Cain, B., Deering, S., Kouvelas, I., Fenner, B., & Thyagarajan, A. (2002). RFC3376: Internet Group Management Protocol. IETF Proposed standard.

[26] Fenner, B., He, H., Haberman, B. and Sandick, H. (2006). Internet group management protocol (IGMP)/multicast listener discovery (MLD)-based multicast forwarding ("IGMP/MLD Proxying"). IETF RFC 4605.

[27] Diot, C., Levine, B. N., Lyles, B., Kassem, H., & Balensiefen, D. (2000). Deployment issues for the IP multicast service and architecture. *IEEE network*, *14*(1), 78-88.

[28] Janic, M. (2005). Multicast in Network and Application Layer. Ph.d. thesis, Delft University of Technology,The Netherlands,

[29] Hosseini, M., Ahmed, D. T., Shirmohammadi, S., & Georganas, N. D. (2007). A survey of application-layer multicast protocols. *IEEE Communications Surveys and Tutorials*, *9*(1-4), 58-74.

[30] R. Zhang, Y.C. Hu, "Borg: A Hybrid Protocol for Scalable Application-Level Multicast in Peer-to-Peer Networks," ACM NOSSDAV, pp. 172-179, Monterey, CA, 2003.

[31] J. Liebeherr, M. Nahas, and W. Si. Application-layer multicasting with Delaunay triangulation overlays. IEEE Journal on Selected Areas in Communications (JSAC), 20(8):1472—1488, October 2002

[32] D. Helder and S. Jamin. End-host multicast communication using switch-trees protocols. Proc. of the 2nd IEEE/ACM International Symposium on Cluster Computing and Grid (CCGRID'02), May 2002.

[33] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. Proc. of ACM SIGCOMM'01, pages 161—172, August 2001

[34] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. Proc. of the 3rd International Workshop on Networked Group Communication (NGC '01), 2001.

[35] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole. Overcast:Reliable multicasting with an overlay network. Proc. of USENIX, October 2000

[36] Y. Chawathe, "Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service," Ph.D. Thesis, University of California, Berkeley, Dec. 2000

[37] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in Communication (JSAC), 20(8), October 2002.

[38] Y. Chawathe, S. McCanne, and E. Brewer, "RMX: Reliable Multicast for Heterogeneous Networks," IEEE INFOCOM, vol. 2, pp. 795-804, Tel Aviv, Israel, Mar. 2000.

[39] L. Mathy, R. Canonico, and D. Hutchinson. An overlay tree building control protocol. Proc. of 3.rd International COST264 Workshop on Networked Group Communication (NGC'01), pages 78—87, November 2001.

[40] P. Francis. Yoid: Extending the Internet multicast architecture. Unrefereed report, http://www.icir.org/yoid/docs/index.html, pages 1—38, April 2000.

[41] Jha, S., Behrens, J., Gkountouvas, T., Milano, M., Song, W., Tremel, E., ... & Birman, K. P. (2019). Derecho: Fast State Machine Replication for Cloud Services. *ACM Transactions on Computer Systems (TOCS)*, 36(2), 4.

[42] Ban, B. (1998). Design and implementation of a reliable group communication toolkit for java. *Cornell University*.

[43] Birman K. and Joseph T. Reliable Communication in the Presence of Failures. ACM Transactions on Computer Systems, 5(1):47–76, Feb. 1987.

[44] Towsley D., Kurose J., and Pingali S. A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols. IEEE Journal on Selected Areas in Communication, 15(3):398–407, Apr. 1997.

[45] A. Hornsby and R. Walsh, "From instant messaging to cloud computing, an XMPP review," *IEEE International Symposium on Consumer Electronics (ISCE 2010)*, Braunschweig, 2010, pp. 1-6. doi: 10.1109/ISCE.2010.5523293

[46] P. Saint-Andre, "XMPP: lessons learned from ten years of XML messaging," in *IEEE Communications Magazine*, vol. 47, no. 4, pp. 92-96, April 2009. doi: 10.1109/MCOM.2009.4907413

[47] P. Saint-Andre, "Streaming XML with Jabber/XMPP," in *IEEE Internet Computing*, vol. 9, no. 5, pp. 82-89, Sept.-Oct. 2005. doi: 10.1109/MIC.2005.110

[48] Andre, P. S. "IETF RFC 6120 Extensible Messaging and Presence Protocol (xmpp): Core." (2017).

[49] Belshe et al. “IETF RFC 7540.Hypertext Transfer Protocol Version 2 (HTTP/2)” https://tools.ietf.org/html/rfc7540

[50] D. A. Menasce, "MOM vs. RPC: communication models for distributed applications," in *IEEE Internet Computing*, vol. 9, no. 2, pp. 90-93, March-April 2005.

[51] SMTP, IETF. Simple Mail Transfer Protocol. Technical report, IETF, 2001. http://tools. ietf. org/html/rfc2821.