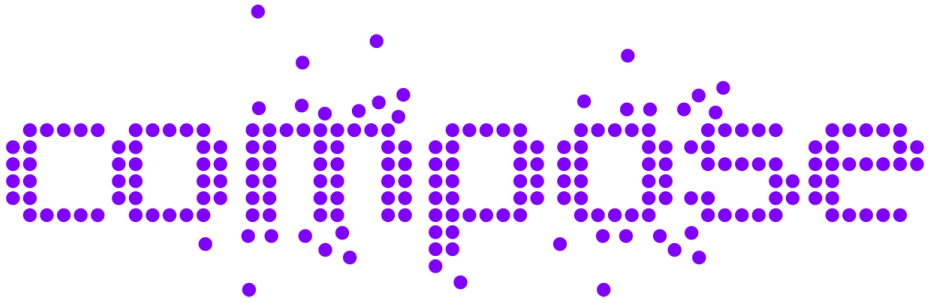

Collaborative Open Market to Place Objects at your Service

D4.2.2

Initial prototype of the COMPOSE communication infrastructure

Project Acronym	COMPOSE	
Project Title	Collaborative Open Market to Place Objects at your Service	
Project Number	317862	
Work Package	WP4 Infrastructure Development and Prototyping	
Lead Beneficiary	IBM	
Editor	Benny Mandler	IBM
Reviewer	Daniel Schreckling	UNI PASSAU
Dissemination Level	PU	
Contractual Delivery Date	31/10/2013	
Actual Delivery Date	31/10/2013	
Version	V1.0	

Abstract

The communication infrastructure aims to provide membership and communication patterns of various kinds to the different entities running within the COMPOSE platform. This document aims to accompany the initial prototype of the communication system rather than provide a detailed design document of the demonstrated system. A full fledged detailed design document is scheduled for the end of the project.

The developed communication infrastructure is based on the creation of a peer-to-peer overlay network, in which each node connects and guards a small amount of peer nodes, and regularly exchanges information with them. That construct identifies and disseminates information as to nodes joining or leaving the overlay. In addition, a structured topology is maintained within the overlay which enables efficient key-based routing, which most notably is used for an efficient implementation of a pub / sub communication paradigm.

Document History

Version	Date	Comments
V0.1	15/09/2013	Initial version
V0.2	24/10/2013	Comments from Daniel
V0.3	28/10/2013	Incorporate test suite information

Table of Contents

List of Figures

List of Tables

Acronyms

Table 1: Acronyms table

Acronym	Meaning
COMPOSE	Collaborative Open Market to Place Objects at your Service
DHT	Distributed Hash Table
JVM	Java Virtual Machine
NACK	Negative Acknowledgment
P2P	Peer-to-Peer
Pub/Sub	Publish/Subscribe

1 Introduction

The communication infrastructure is meant for the scalable dissemination of membership information as well as several communication paradigms to the different users encompassing the COMPOSE platform. This service will be available as a generic platform service that can be used by the platform administration itself for management purposes, by the different COMPOSE entities embedded within the platform for communication and for achieving self management properties and may be available to external entities as well (mainly in the form of notifications). We opted for a peer-to-peer (P2P) based overlay as the underlying structure of the communication system. An overlay network is made of a group of processes that connect to each other, whereby each process connects only to a small number of its peers. This technology was mainly chosen for its contribution to the overall scalability of the system.

Services provided by this component at this time are membership and publish / subscribe (pub / sub). Later releases of this component may add (i) point-to-point messaging either over the overlay links or directly between the peers by opening a specific connection between the source and the target, (ii) broadcast, for enabling the submission of a message to all participating entities; a capability mainly used for management purposes (iii) DHT - for a scalable and distributed information location service and (iv) convergecast as an advanced monitoring infrastructure element. A central design point is to have a unified technology that will cover the entire spectrum of communication capabilities required for the proper and efficient running of the platform. That includes pure communication capabilities, as well as membership and monitoring components.

The design followed is of a scalable, fully distributed, messaging, membership and monitoring infrastructure. This scalable and distributed infrastructure will utilize peer-to-peer and overlay networking technologies to perform its operations at the scale and performance envisioned, as can be seen in Figure 1. In traditional small scale systems, typically a full connection mesh is used in which each node is connected to each other node in the system. These are typically centralized solutions in which either IP multicast is used or in contrast multicast by unicast methods are used. Such a scheme is not feasible anymore when moving to a large scale, in which case we opt for a self organizing, structured overlay network providing overlay-based communication services. In addition, strong consistency semantics which are feasible in small scales become a hinder to scalability, thus in this large scale we opt for an eventual consistency semantics for the membership information. Individual group communication sub-systems can implement a variety of messaging semantics, which in turn can be used by different clients based on the needs of their own system. In the current pub / sub implementation both bets effort and a NACK based semantics are supported.

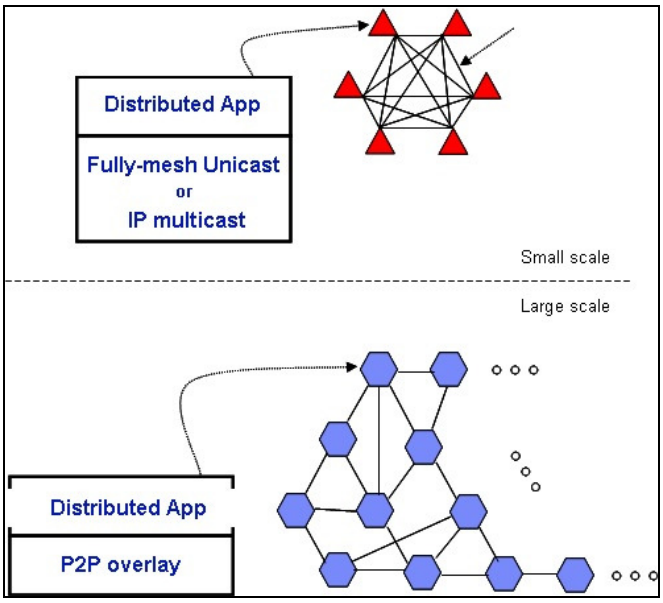

Figure 1: Communication structure

The system demonstrated at this time consists of a single overlay group, which based on the underlying architecture can scale efficiently up to a couple of thousand members. Later releases of this component will devise a hierarchy built on top of the group concept presented at this time to achieve higher scalability. One aspect of the hierarchy will consist of thin clients connecting to a server which belongs to the group. That will enable thin clients to participate in the global communication scheme without incurring a large overhead. Thus, the current intention is to install the communication sub-system on multiple servers running inside the platform, and in the future have smaller COMPOSE entities communicate with this communication servers based on their needs.

The main services envisioned by this component shall be membership, scalable group communication, and in the future a monitoring, and a DHT services may be added. The actual services to be implemented and their priorities will be determined by the needs of additional platform components. This set of services will enable (i) the operation and orchestration of resiliency aware applications, providing fault tolerance – may be used by the platform controller component (ii) Monitoring, load balancing, resource management, and efficient components scheduling - may be used by the platform controller component (iii) Support distributed resource location and discovery both by applications and the platform management, and (iv) Application integration and cooperation, by using the pub / sub service.

The current prototype demonstrates a tier of communication servers forming a single overlay group. Future releases will contemplate whether a thin-clients tier connected to the servers tier is enough to efficiently support the platform or whether the servers tier itself needs to be devised in a hierarchical fashion as well. We have given some thought to that option and have a solid design in case it shall be required. A sneak preview can be seen in Figure 4.

This prototype is written purely in Java, so that a proper JVM is needed wherever this is to be deployed. In addition, as the prototype is of a networking nature, it relies on the existence of the TCP stack and the availability of free ports, one per each participating node. Moreover, at the moment the assumption is that all nodes are in the same administrative domain and do not have to cross firewalls.

Communication between the nodes is performed via TCP, and the information is transmitted as is. Thus, if stronger security requirements are in place, higher layers need to take care of this. For example, encryption and decryption of messages can take place at the applications using this infrastructure for group communication.

The main purpose of this document is to accompany the technical demonstration and provide the necessary background information to be able to put into the right perspective within the COMPOSE platform. It is not intended to be a full-fledged design document of this component. Some more details can be found in the initial architecture deliverable, and a full fledged design document is expected at the end of the project.

2 Prototype overview - short description

The prototype is centred on demonstrating the membership and pub / sub capabilities of a single overlay group.

- The prototype will demonstrate the manner in which nodes join the group, and leave the group in an orderly manner.
- Nodes crashing and their effect on the group as well as the dissemination of the relevant information will be demonstrated.
- Laying the ground for scalability measures of the membership components will be demonstrated.
- Pub / sub capabilities will be demonstrated.
- Laying the ground for scalability measures of pub / sub will be demonstrated.

2.1 Architectural responsibilities and interactions

This component will reside at the lowest layers of the COMPOSE platform run-time and will serve as a generic service to be used by other COMPOSE components, as is illustrated in Figure 2. The platform in which the COMPOSE platform will run contains its own messaging infrastructure, which is used for the proper functioning of the platform itself. On the contrary, the richer functionality we provide will be used by COMPOSE entities such as Services and Service Objects.

First and foremost, the platform run-time management will make use of this component to maintain information on entities which are alive and those that have left or crashed. In future releases more versatile monitoring schemes will be made available. The main interaction of this component is with the platform run-time management system that can figure out which components are alive. Other entities can register as well to obtain that information.

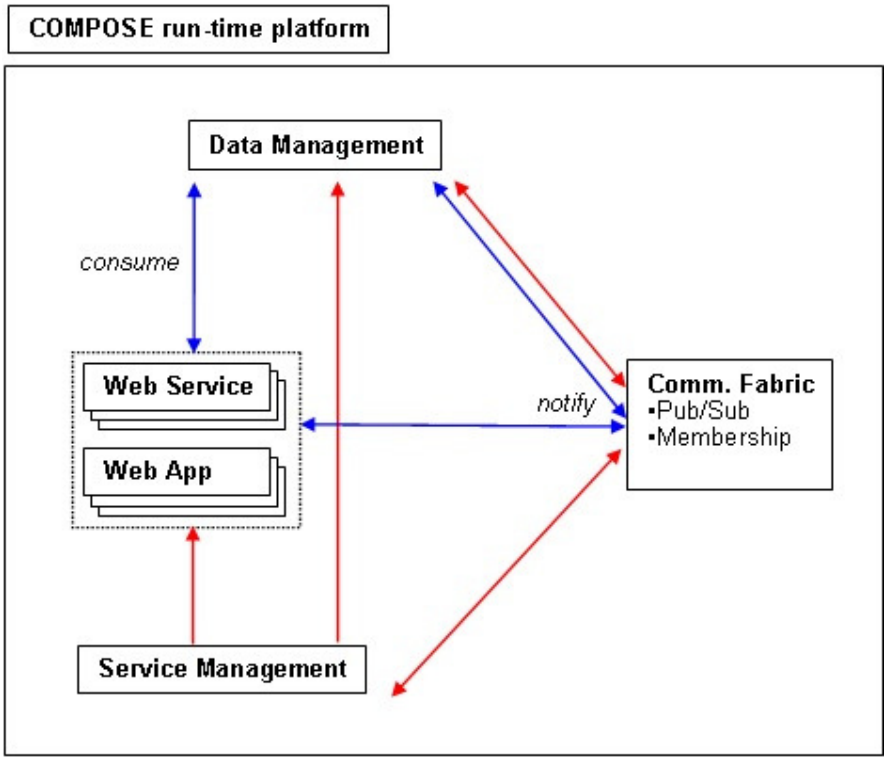

Figure 2: Internal platform interactions

Pub / sub communication can be used by platform entities as well. It can serve as a channel through which de-coupled services or applications get to cooperate. Using such a capability enables the communication and coordination between distinct entities which do not have to be aware of each other, but only need to agree upon the topic (or channel) through which they will be communicating. On the other hand it can serve as a notification mechanism to enable registered entities to find out when certain events have finished and potentially obtain the results of the preceding operation. That can serve as a basis for a lightweight orchestration mechanism of services running within the COMPOSE platform.

2.2 High level design

The design is of a fully distributed, self organizing, overlay network that does not rely on IP multicast. The group members are organized in a ring based on their virtual ID and each node is connected to its successor and its predecessor. Whenever a node fails or a new node joins the overlay the connections to the successor and predecessor are updates or established if necessary. That ensures connectivity and reachability within the virtual ring. Furthermore, in order to enable efficient key-based routing and information dissemination within the group a structured overlay topology is created and maintained, in which each node connects to a logarithmic number of its peers. These connections enable the efficient dissemination of data, since a correct construction of such a topology will enable each hop in the message route to at least half the distance to the ultimate message target. Several separate topologies can be supported and co-exist, such as adding a random topology layer which may handle the gossip traffic which is used for information dissemination purposes, such as membership and attributes replication. A random topology does not exist within the current prototype.

In order for a new node to join the group it has to know only the address of a single group node which is alive. Once the node is connected to the group, it can determine its location within the ring and all the peer connections are created. Thus, each node is guarded by a small number of peer nodes. Upon a node leaving or failing, the information will be captured by its peers and transmitted to the rest of the group members via a gossip mechanism. Additional options for fast failure detection exist, such as use message flooding for that information or designate a single node to collect that information and have the information transmitted to it directly in addition to the gossip mechanism.

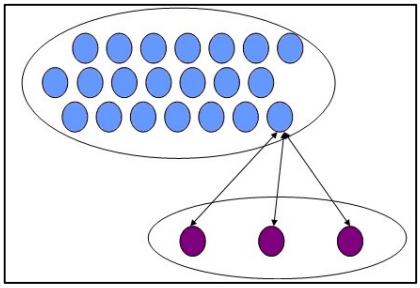

Figure 3: Thin client-server hierarchy

The external API, discussed in section 4, enables the registration of listeners to obtain the information on nodes leaving or joining the group. In addition, the API enables the creation of pub / sub topics, and publishers and subscribers. The underlying infrastructure ensures that published messages will be delivered to all interested subscribers in an efficient manner.

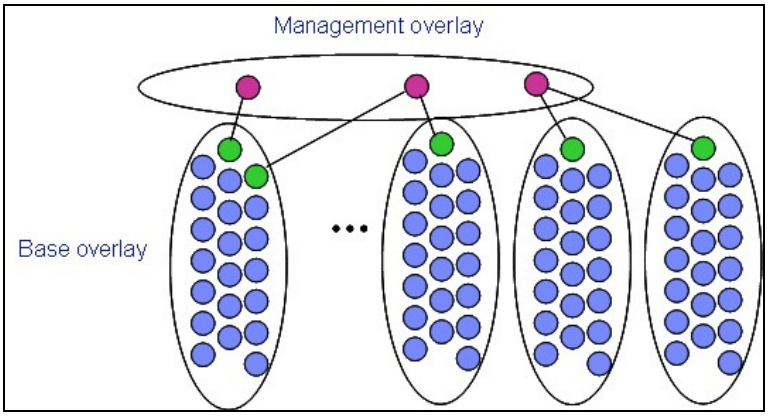

Figure 4: Possible future hierarchy

The design calls for the future introduction of another type of entity, namely thin clients. The main reason for the introduction of a thin client notion in addition to the existing communication server is that many of the entities running in COMPOSE that need to be connected to the network are light weight and having them bear the full fledged communication stack would be excessive. Thus, the servers will form a group of their own in which each member is connected only to a limited amount of its peers. The servers group will have full membership knowledge, providing an eventually consistent semantic. The servers' overlay is the part that has been implemented so far and is currently being demonstrated. The thin client part has been partially implemented at the moment, but is still not ready for prime time demonstration. The introduction of thin clients will enhance the scalability of the system and will enable smaller COMPOSE entities to participate in the communication scheme without bearing the complete burden of the full-fledged communication module. The planned API will enable these clients to participate in the group communication and monitoring activities as participants, able to obtain and broadcast information. This design aims to balance the desire on the one hand to have different entities participate in the communication bus, while keeping in mind that some of the participating entities are small and thus cannot be required to embed within them a full fledged communication server. Monitoring of all the different kinds of

components in the platform will be made possible and the servers' group will have a complete view of the state of the system and all its components.

This scheme, by design, scales up to a couple of thousands in the servers tier. In higher numbers delays in the information propagation may appear as the amount of content that needs to be transmitted by gossip throughout the group becomes rather large. Each server can support numerous smaller scale devices, thus reaching the envisioned scale of supporting hundreds of thousands of participants. As a possible future refinement of this scheme we shall contemplate a full fledged hierarchical structure at the servers tier as well, in which individual groups will be federated by a management group, thus reaching a scale of up-to a million in the servers tier itself. The hierarchical design relies on the same notion of a group mentioned above; with many such groups operating individually and having a representative of them connect to a management group that federates all the individual groups.

2.2.1 Main functionalities introduced

Membership

The main goal of the Membership service is to supply members of the group with the information regarding connected processes. This is needed for creating the view of which processes are currently alive, and obtain some information they want to share about themselves. Such information may aid higher layers in several aspects of the management of the system, such as supporting fault tolerance. In order for the system to scale better we opted for an eventual consistency semantic in which after a certain amount of time has elapsed with no processes joining or leaving the system all nodes will have the same membership information. In addition, to support pub / sub, the interests of the different nodes is maintained and propagated by the same membership mechanism. The interests of a node include the topics on which it wishes to publish as well as the topics from which it wants to receive messages.

A joining node needs to communicate with at least one member node which is alive. Once such a connection has been established, both nodes exchange information and based upon that the joining node starts establishing connections to additional nodes in the group, and may cause changes in the connectivity of existing nodes.

Every node in the group maintains open connections to only a small amount of its peers and guards them via a configurable heartbeats mechanism. Once a node ceases to receive heartbeats for a configurable amount of time it declared the peer to be suspect. Information concerning nodes that have joined, left, or crashed is transmitted periodically by each member to its connected peers, and thus the information is spread via a gossip mechanism throughout the cluster in a logarithmic time scale.

Such a capability is important for the platform administration to get a view of which entities are alive, and which others have disappeared. This information has implications for example on an orchestration engine, a recommendation and a composition engine.

Group communication

Efficient broadcast and publish / subscribe can be based on key-based routing. Once a structured topology has been established, efficient routing can be performed based on that topology.

Efficient pub / sub routing is achieved using information maintained by the interest aware membership service. Such a service was implemented and is demonstrated via the pub / sub capabilities. The interest aware membership calls for not only notifications on nodes joining or leaving the system, but also on nodes changing their interest, for example becoming a publisher or a subscriber of a specific topic. Such a service can be used to coordinate between different

entities interested in the same information distribution, without knowing all entities in advance or have direct connections between them.

Support can be provided as well for a Write / Subscribe semantics (a bulleting board) if needed. Such a semantic ensures that the last item published on a certain topic is received by interested subscribers. Such a service may serve as a building block for a workflow / orchestration engine, in which a process that finished its duties informs waiting processes via a specific topic. A process that needs to kick-start its operation grabs the information from the bulletin board and can start its respective flow. This capability is not demonstrated at the moment but can be added later if there will be an architectural need for it.

Monitoring

Monitoring functions can be supported by attribute replication based information spreading. Such a service is designed for slowly changing data that is not very large. The attribute replication service enables a node to set its own attributes and have the information replicated throughout the system using the internal gossip mechanism. In this scheme each node holds a read-write copy of a map of its own attributes and a read-only copy of the maps of all other nodes in the cluster. The information is propagated throughout the overlay using the gossip mechanism.

More efficient and versatile mechanisms can be out in place, and will be considered in later stages of the project in conjunction with the monitoring task taking place in WP3.2.

3 What is being demonstrated

This being the initial prototype of the communication technology the main aim was to produce a sound basis from which later, more advanced features and characteristics can be developed. Thus, a thorough design and rigorous testing were performed for this implementation to ensure that the basic capabilities are in place and work properly under various conditions. Specific aspects of scale and performance were taken into consideration during the design process, but at this stage these aspects were not put to the test. As a second phase we will initiate work on producing performance and scalability numbers, and take care of potential bottlenecks that may appear.

The tests were designed to test major functionalities of important classes, each layer separately, and a combination of layers working together. Examples of internal layers include (i) a communication layer, in charge of the low level network communication, (ii) topology layer which creates and maintains the desired topologies, (iii) gossip layer which is in charge of information dissemination and reconciliation.

Some examples of the test suite that was developed to test the system as it was implemented, as well as to serve as a regression test suite include:

- Verifying that the underlying communication module can be created
- Verifying that two such components can use the same port
- Verifying that two such components can send messages to each other
- Verifying that two such components can receive respective events
- Verifying that in a configuration of 2 when 1 dies the second one notices it
- Verifying that giving the right configuration many members can join a group

- Verify that once a group is established one can start having nodes leave and join and all the existing nodes will obtain the correct view of the current membership
- Verify that both subscribers and publishers can be created
- Verify that a published message arrives to all destination
- Verify that is a message is lost along the way (getting deliberately dropped) it will be re-sent and received eventually
- Verify that many publishers and many subscribers can exist and function well simultaneously
- Verify that various subscription and publishing patterns work correctly
- Verify that many topics can operate simultaneously
- Verify that no subscriber, single subscriber, many subscribers and all nodes as subscribers works ok
- Verify that no publisher, single publisher, many publishers and all nodes as publishers works ok
- Verify that publisher and subscriber on the same node works ok
- Verify that creating publisher first or subscriber first works ok
- Verify that processes on different nodes can form a group

4 External API - for use by 3rd parties

Enclosed is a very concise sub-set of the API that can be used by other components when interacting with the communication infrastructure.

- General node actions
  - create() – Will kick-start the process and will join the overlay.
  - close() – An orderly leave of the overlay.
- Membership
  - onMembershipEvent (MembershipEvent) - Obtain membership events by registering a specific call back with the communication sub-system. That callback will be invoked each time the node is made aware of a change in the membership of the group, or in the interests of a group member.
- Pub / Sub
  - createTopic (topicName, config) – Create a specific topic to be used later by publishers and subscribers
  - createTopicPublisher (config, EventListsner, Topic) – Creates a topic publisher instance for a particular topic.
  - createTopicSubscriber (config, EventListener, MessageListener, Topic) - Creates a topic subscriber instance for a particular topic. Messages will be received by the accompanying message listener callback.
  - *publishMessage (OutgoingMessage)* – Method used by the topic publisher to publish a message on its particular topic.
- Messages

- onMessage (IncomingMessage) – Callback for messages reception

5 Future directions

- Scalability – Scalability enhancements will be assessed along with the progress of the various parts of the project. Sound design plans are already available for a hierarchical approach to the overlays structure.
  Additional scalability enhancements may be introduced based on findings of specific issues with the currently demonstrated system. For example, enhancements to the attribute replication and gossip mechanisms, or considering NACK-based approaches for messages dissemination reliability.
- Point-to-point (overlay or direct) – Exposing the possibility to transmit messages to a particular destination. Upon need two flavours of a point-to-point mechanism can be supported. First, routing messages over the existing overlay links. Second, creating specifically tailored connections to a particular destination in addition to the existing overlay links, in case heavy traffic is expected between the two nodes (such as if there is a need to support data replication).
- Broadcast – Sometimes there's a need to transmit a message to all members of the group. This is mostly the case for administrative purposes. An efficient manner in which to perform broadcast at scale in an overlay system is important especially if this mechanism is to be used extensively and gets more important as the scale gets bigger. We have designed such a service that enable efficient broadcast by halving the target range at each hop of the message, and may implement and expose it upon need.
- DHT – a DHT can be a useful building block for information discovery, as a sort of a distributed repository. Within an overlay based DHT the entire space is divided between the participating DHT server nodes based on their virtual ID. Each key is mapped to one DHT server which it contacts in order to interact with the desired key. A scalable and elastic implementation can be achieved within the basis of the communication fabric; such that severs can be added or deleted to the DHT servers group based upon need and expected workload.
- Bulletin board – a Bulletin board can be thought of as an implementation of a shared state (or shared memory). Nodes get notified of the last value submitted on a specific topic but there's no guarantee that all values will be delivered (intermediate values may be skipped). This capability is often associated with maintaining distributed monitoring information in an efficient manner. This capability can be built using the existing pub / sub design and implementation and will be considered based upon the platform's needs.
- Convergecast – In essence it is a two stages process. The first stage is a broadcast message which builds an on demand tree along the way, publishing the piece(s) of data which are to be collected and potentially deposits an aggregation function. An aggregation function may be used for a more efficient computation, such as when trying to determine the load on the most loaded server, at each node the aggregation function will return the maximum load it knows about. The second stage is a reverse broadcast which traverses back the tree, passing along the local result, and potentially applying the aggregation function on the way back. Such a capability can efficiently aggregate query responses from many to one, on topic scope.