Collaborative Open Market to Place Objects at your Service

SEVENTH FRAMEWORK PROGRAMME

D3.2.2.1

Design of the service monitoring tools

Project Acronym	COMPOSE	
Project Title	Collaborative Open Market to Place Objects at your Service	
Project Number	317862	
Work Package	WP3.2 Services deployment	
Lead Beneficiary	IBM	
Editor	Benny Mandler	IBM
Reviewer	Guy Laden	IBM
Reviewer	Iacopo Carreras	UH
Dissemination Level	PU	
Contractual Delivery Date	31/07/2014	
Actual Delivery Date	31/07/2014	
Version	V1.0	

Abstract

The design of a monitoring infrastructure for the COMPOSE platform is detailed. The design is broken into three pieces, namely the monitoring communication infrastructure, the producers of monitoring information and the consumers of monitoring information.

The monitoring infrastructure nicely fits into the effort of producing a unified technical approach to communication related components within COMPOSE. Thus, the platform makes use of the scalable communication infrastructure already deployed as a part of the COMPOSE platform, for the transport of the monitoring information. Moreover the producers and consumers of monitoring information make use of extensions of the communication infrastructure for their proper operation.

Monitoring information producers come in different flavours. First, the built-in communication system membership information sharing scheme provides an indication as to the liveness of its own components which can represent external entities they are engulfed with. In addition specific local agents may be operating in specific parts of the platform providing information stemming from their own running environments. Such agents can be made a part of the cloud infrastructure by monitoring liveness of VMs for example, or can be more specific to COMPOSE components by providing information on COMPOSE applications.

There can be multiple monitoring information consumers, chief among them is the COMPOSE cloud controller which is in charge of deployment and lifecycle management, which needs this information for its own proper functioning.

Document History

Version	Date	Comments
V0.1	01/07/2014	Initial version
V0.2	10/07/2014	Add figures and comments from Guy
V0.3	29/07/2014	Integrate comments from Iacopo
V1.0	30/07/2104	Add final touches and explanations

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
API	Application Programming Interface
P2P	Peer-to-Peer
Pub/Sub	Publish/Subscribe
VM	Virtual Machine

1 Introduction

The monitoring infrastructure is meant for the scalable gathering and dissemination of information from throughout the COMPOSE platform, concerning the liveness of the various entities. The monitoring component is comprised of three parts, namely (i) components gathering and publishing the relevant information, (ii) components subscribed to obtain previously published monitoring related information (iii) the information dissemination backbone.

The COMPOSE monitoring infrastructure comes to complement the built-in cloud health monitoring system provided by the infrastructure. That health monitoring system communicates with the Cloud Foundry cloud controller and provides information on the state of VMs. That information in turn is used to help the cloud controller with its management decisions. That infrastructure can be used as one of the sources of information for the COMPOSE monitoring scheme, but in addition we want to make sure to add the flexibility to monitor the specific elements which are important to the COMPOSE platform, and further ensure that the interested parties have easy access to this information.

The basic aspect which is being monitored is the liveness of COMPOSE components. After this functionality is in place the project will assess whether there exists a need to use the same infrastructure for monitoring other system aspects, such as resources consumption.

The monitoring infrastructure will support, among other aspects, self-management capabilities within the platform. Such capabilities will be manifested by the platform being informed as to the updated state of various components, and thus have the ability to act upon changing state of affairs in an autonomic fashion.

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 (such as direct links and pub / sub), as well as membership and monitoring components.

The monitoring component's main expected use is to serve as an infrastructure providing support for resiliency aware applications, providing fault tolerance of COMPOSE applications and in the future possibly also composite applications.

The main general interaction with COMPOSE components can be viewed in Figure 1. As can be seen in the figure, the monitoring component is spread over the internal platform scalable communication infrastructure component. On the one hand it obtains its information from the different kinds of components that comprise the platform, for example the data management and the service management component. Thus, information on different kinds of COMPOSE objects, such as Web Objects, and COMPOSE applications and services can be made available through the monitoring infrastructure. On the other hand potential monitoring information consumers can be reached via the monitoring system as well, such as the service management component which includes the COMPSOE cloud controller.

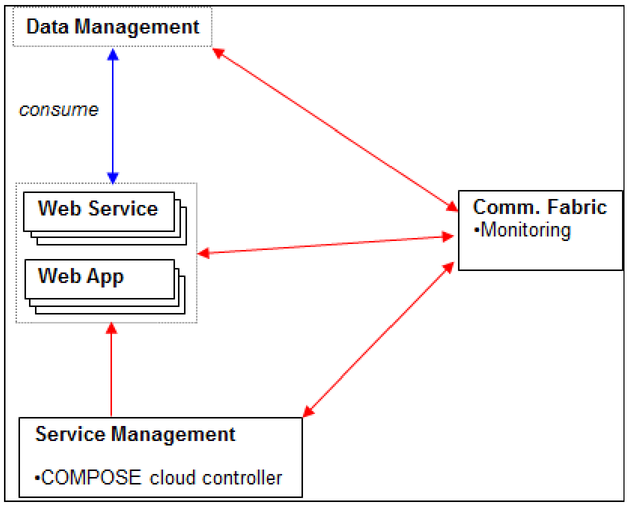

Figure 1: Interactions with major components

2 High Level Design

The main aim of the COMPOSE monitoring component is to support a distributed and scalable collection and dissemination infrastructure, providing the ground for multiple kinds of elements to be monitored and to gain access to monitoring information. Thus, the monitoring infrastructure is based on the COMPOSE scalable communication infrastructure, which at the moment provides membership and pub / sub services for the COMPOSE platform.

The monitoring infrastructure is fed by local agents collecting specific information. There may be several kinds of such local agents, each tailored for the collection of information about a specific kind of component. There shall be local cloud clients which will collect mostly liveness information about cloud hosted components, such as COMPOSE run-time applications. In addition there may be such a client targeted towards COMPOSE cloud services which may be hosted outside the cloud itself. Agents operating within the COMPOSE cloud platform can provide liveness information on infrastructure components and may collect and distribute also resource consumption information. In addition there shall be an agent which will provide monitoring liveness information on the Web objects infrastructure of COMPOSE. This agent will co-exist and co-reside with the data management layer of COMPOSE.

These local agents will communicate with the monitoring infrastructure via a thin client that will be provided for both Java and JavaScript. This thin client will establish a communication link with the COMPOSE scalable communication overlay via one of its servers and will be able to publish the monitoring information via that thin client as well as receive notifications from the global monitoring infrastructure. Thus, each local agent will contain, configure, and instantiate such a thin communication client to be used for publishing monitoring information as well as receiving feedback information from the infrastructure itself.

The main information consumer of the monitoring information that is being collected is envisioned to be the COMPOSE controller, which is in charge of the deployment and lifecycle aspects of COMPOSE applications. Liveness information is required in order to drive the correct lifecycle actions. In addition, the deployment portion relies on the lifecycle component to provide updated information as to the state of COMPOSE applications, such that correct actions can be taken upon applications deployment. In an advanced scenario the deployment component may use this information to indicate to the orchestration engine about certain components that have failed, and thus have either a new composite application being deployed or have the orchestration engine bypass the failed component.

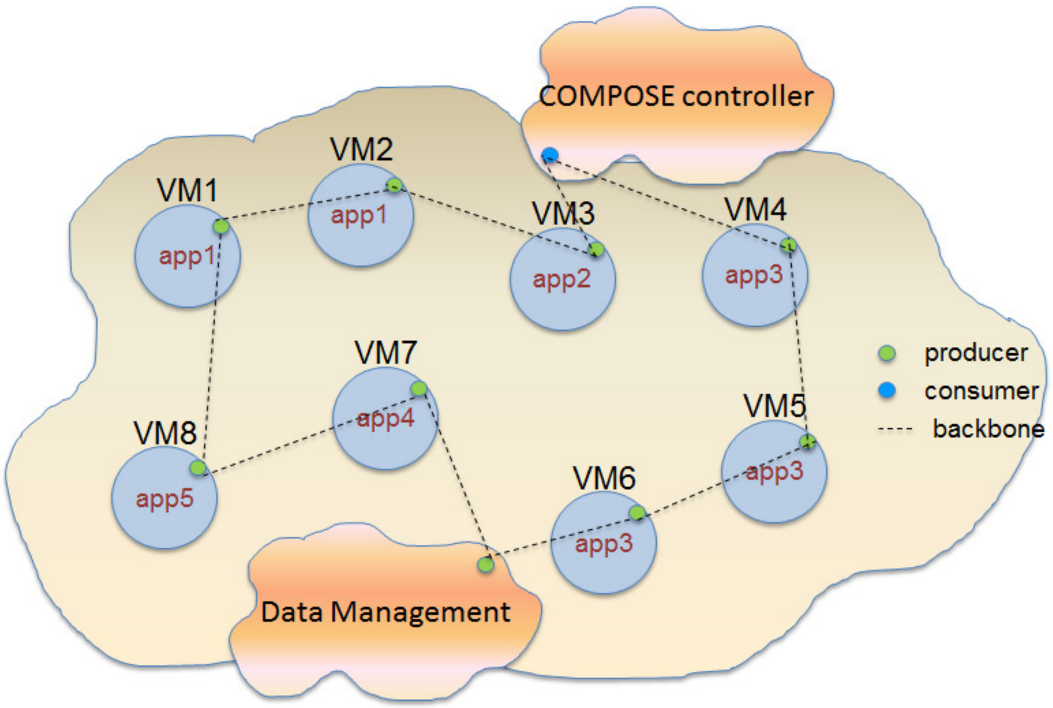

Figure 2: Monitoring infrastructure main components

Additional potential consumers of the monitoring information may include a dashboard, serving as an operator monitoring hub, via which updated information about the state of components within the COMPOSE platform can be viewed. In addition, in the future additional capabilities can be built on top of the monitoring infrastructure by which some historical data can be collected and maintained, potentially using a cyclic DB to keep the amount of resources consumed by this component in check. This information may be used by a recommendation engine that keeps liveness statistics, or by a reputation engine that takes such information into account for assigning relative reputation values.

2.1 Dissemination backbone

The dissemination backbone design is based on the existing scalable communication component integrated within the COMPOSE platform[1], serving membership and pub / sub capabilities

[1] See D4.2.2 "Initial prototype of the COMPOSE communication infrastructure" for more detailed information.

within the cloud platform. The communication service is being deployed as a part of the core COMPOSE platform, and is made available as a generic platform service supporting, among other capabilities, self-management properties.

The scalable communication component is based on a self-organizing, structured peer-to-peer (P2P) overlay for scalability purposes. The overlay is constructed such that all peers are reachable by all other peers in the group, but connections are maintained only to a subset of the group nodes.

The communication infrastructure enables distributing information among peers in various manners using various mechanisms. A single overlay group can scale efficiently up to a couple of thousand members. This group will consist of communication servers that will be installed within the platform. To further scale the solution and enable the participation of various kinds of smaller components for which it is not feasible to run a complete server, we introduce the notion of a thin communication client. The thin communication client will provide access to all the capabilities offered by the overlay while carrying a small and light footprint. Most notable the thin client will enable establishing a connection to a communication server, use the monitoring API to publish monitoring data, and use the monitoring API to register for the reception of previously published monitoring data.

The communication servers forming the backbone of the communication system are written in Java, and thus require a JVM and the availability of the TCP networking stack and free ports. The thin client will be made available both in Java as well as in JavaScript to enable the easy integration with the COMPOSE platform in which is the main programming language being supported is JavaScript.

The thin client will be configured with the identity of a list of coordinating communication servers. Upon start up the client will establish a connection to one of the coordination servers and will obtain from it the identity of the communication server to which it needs to bind. Once the connection is established with the designated communication server it is maintained until the communication server stops for some reason. At that time the client will turn back to one of the coordinator services and obtain the identity of a new server to bind to.

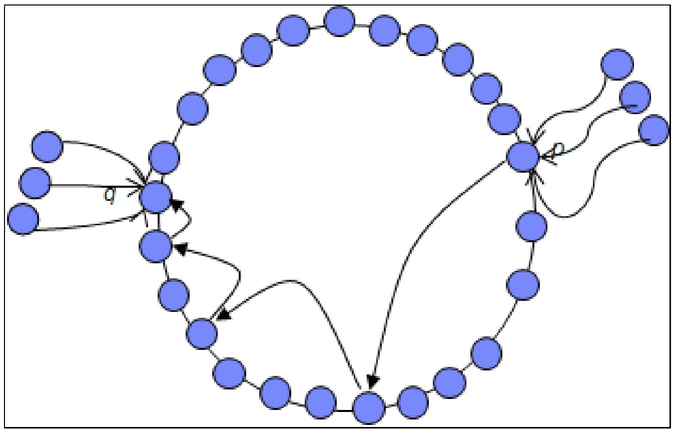

Figure 3: Scalable dissemination backbone - servers and clients

The API of the communication infrastructure was extended to easily accommodate entities providing and receiving monitoring data. The proposed API is described in section 3. This API will be supported both by the communication servers as well as the thin clients. Its realization shall be completely independent of the actual implementation of the information dissemination within the communication cluster.

2.1.1 Design alternatives

Various implementation alternatives for the efficient dissemination of monitoring data using the existing scalable communication infrastructure were considered during the design phase. The first one took advantage of the built-in attribute replication mechanism. The attribute replication mechanism was built for slowly changing data that is not very large. The attribute replication service enables a node to set attributes within its own map and have the information replicated throughout the system using the internal gossip mechanism.

The second alternative was to use the already existing publish / subscribe infrastructure. In this design choice specific monitoring topics will be created internally by the communication system. A call to disseminate a piece of monitoring information will be passed to the communication infrastructure via the monitoring API and will be translated internally to a publish operation. On the other hand, components who wish to be notified of newly posted monitoring information will use the monitoring API to express that intention. Internally a subscription to a monitoring topic will take place to make sure that the information is propagated to the designated communication entity.

A third alternative would be to implement a convergecast algorithm within the communication infrastructure. Convergecast can be viewed as a bi-directional broadcast in which a spanning tree is created on the outgoing path and the results are propagated back on the revers path. As can be seen in Figure 4, the tree can be created dynamically in real-time and there can be many roots to such trees. It is possible also to build such a tree that would not necessarily involve all the nodes in the overlay. In addition, computation or aggregation functions may be deposited in the intermediate nodes while creating the tree. These pieces of computation may be used on the reverse path while collecting the results in order to reduce the amount of traffic flowing in the network.

At the initial implementation stage it is anticipated that the implementation will be based on the built-in pub / sub mechanism. Once the initial implementation is in place there will be an evaluation phase to ensure that this approach fits the intended purpose.

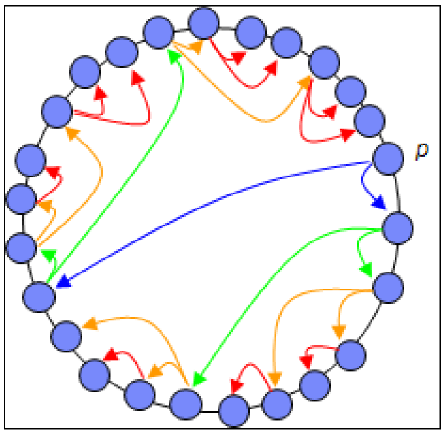

Figure 4: An on demand broadcast tree

2.2 Information producers

The first line of monitoring information producers consists of the elements of the communication infrastructure components themselves, via their built-in membership information gathering mechanisms. The main aim for maintaining membership information is to supply members of the group with updated information as to which components are currently connected, and which others have left the group. Information about nodes that have joined, left, or crashed is gossiped periodically by each member to its connected peers, and thus the information is distributed eventually to all group members. This information can be used by the cloud controller and its sub-components to obtain a view of live and failed components.

2.2.1 Monitoring agents

The main producers of monitoring information will consist of local agents that will be collecting specific information within their respective domain and entities. These agents are software programs which on the one hand connect to the communication infrastructure via a thin client, and on the other hand collect monitoring information for a specific entity and share that information with the rest of the interested parties in the cloud via the backbone of the monitoring infrastructure. For example there can be an agent providing information on the state of COMPOSE applications, in which case they will be running on the VM hosting the specific application that is being monitored; another can provide information on the state of the data management layer in which case it will run within the VM hosting the web container associated with the data management layer, and so on. Such agents will be developed to run within the COMPOSE cloud environment and provide liveness information for COMPOSE entities such as applications hosted within the cloud and possibly services that are connected to the cloud and can be used by COMPOSE applications (such DB, messaging, etc.). Agents which are connected to cloud infrastructure components may provide additional kind of information, such as resource consumption within the respective entities. In addition we would like to monitor as well the liveness of Web Objects connected to the COMPOSE platform. This task will require an agent of its own. It is anticipated that the frequency of providing such information will be configurable with a reasonable default to be set to 1 second, in addition to on demand responses to specific requests.

The monitoring agents producing information are well integrated into the cloud based COMPOSE platform by virtue of running within its internal components, and thus scale along well with the cloud infrastructure. It is anticipated that there will be at most a single agent per VM running within cloud. The exact amount will change accordingly to established cloud management policies, and thus will scale along with the rest of the infrastructure.

There are various potential manners in which the monitoring agents can obtain their liveness related information. In one extreme a thin monitoring communication client can be incorporated within every running COMPOSE application, and thus its mere existence carries the liveness information of the engulfing application. A second approach is to have a monitoring agent in each VM, and thus the disappearance of the monitoring agent signifies the crash of all applications running on that VM. In addition, the agent can monitor specific applications by keeping track of the processes table, or by establishing a heart-beating mechanism between both entities. In general, schemes for monitoring entities range from merely observing the existence and disappearance of TCP connection through general purpose per process health checks to application-specific health checks via a REST interface to be integrated into running applications.

The monitoring traffic is not expected to be high. Every agent will provide a small liveness related piece of information every configurable amount of time, with the default being set to one second. Other types of information will be published in lower rates than liveness information.

2.3 Information consumers

The main consumers of monitoring information are expected to be within the COMPOSE platform run-time management, most notably the COMPOSE controller which is in charge of the deployment and lifecycle components. As an extension, the orchestration capability may use monitoring information in the future to improve its operations. The main kind of information is expected to consist of liveness of different kinds of components. Thus, the platform management modules can figure out which entities are alive, and on the contrary which ones have crashed and react accordingly. A secondary kind of information may consist of resource consumption of infrastructure elements.

Nevertheless, other entities can register as well to obtain that information and make good use of it. Note, that the communication infrastructure is used as a means for disseminating information from the right sources to the interested targets, but the payload itself is opaque to this transport layer. The producers and consumers of monitoring data need to agree on the proper format to be used on both ends.

Overall, the amount of monitoring information consumers is expected to be low.

2.3.1 COMPOSE cloud controller: Services deployment and lifecycle manager

The COMPOSE cloud controller is in charge of connecting between COMPOSE developers and the COMPOSE cloud run-time. This component receives the applications created by the developers and passes them through the entire process which will make them COMPOE-ready. One of the controller's sub-components is a lifecycle manager which tracks the updated state of COMPOSE applications. The state of the applications dictates which actions can be taken on behalf of these applications. Thus, the lifecycle manager is a prime candidate for being registered to obtain monitoring information which will help in keeping its internal data structures up to date. That information can be used by the COMPOSE controller to take the correct action when certain actions need to be taken, such as the deployment of a simple or a composite application. The exact set of steps that need to be taken, and the entire feasibility of a developer request depends on the current state of all the components involved.

2.3.2 Additional potential consumers

Recommendation system

A recommendation system can be deployed by the COMPOSE platform to help developers choose between multiple viable building blocks options while developing a new application. There can be many criteria by which a recommendation system may operate. One of these criteria may be based on the monitoring information. In this case the recommendation system may register itself to obtain monitoring information and may choose to keep that information as a historical data collection, potentially using a cyclic DB in order to ensure that the size of the information does not get out of hands. Such a scheme will enable the recommendation system to determine the availability levels of different applications, and thus be able to provide proper recommendations based on those criteria.

Composition engine

A composition engine provides advice to the developer as to combinations of existing building blocks which will provide the desired outcome. Thus, a composition engine may be interested in obtaining monitoring information and take it into account while producing its output. This can take one of two options, either pipe the composition output via the recommendation engine which takes availability information into consideration, or post-filter the results of a composition request to discard or at least mark compositions which include elements which are currently marked as failed.

Orchestration engine

The following link in the chain after the composition engine is the orchestration engine. The main goal of an orchestration engine is to take the desired composition and make a run-time artefact out of it. Thus, an orchestration engine may be interested in obtaining updated monitoring information to be used at run-time to alter the existing composition and replace any failed components by alternative ones that provide similar results, but which happens to be alive. This is currently viewed as an advanced stretch goal of the project.

Cloud Management Dashboard

Finally, monitoring information can be of help to a cloud management dashboard which needs to display updated information concerning the various entities which are hosted within the cloud. Such a dashboard can be tied indirectly to the monitoring system, for example via the lifecycle management component, or have a direct contact to absorb monitoring information at first hand. Such a dashboard will be interested not only in pure liveness information but also potentially in resource consumption matrix.

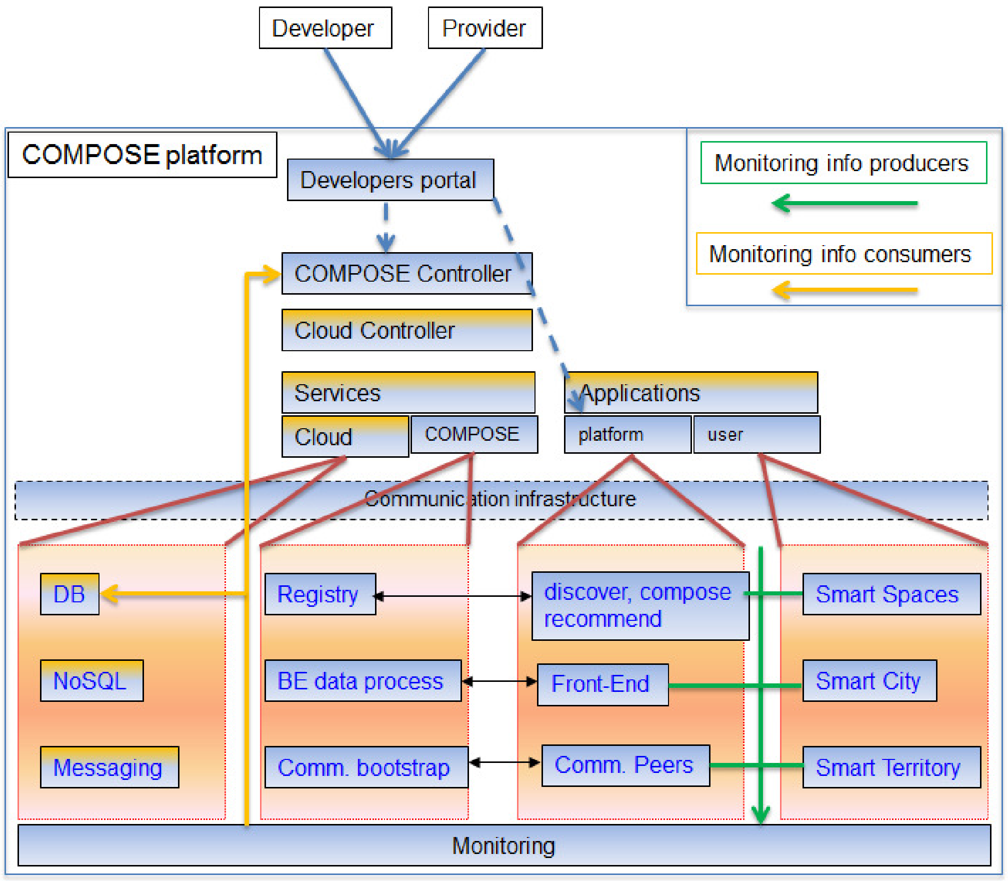

Figure 5: Monitoring information flow

In Figure 5 a pictorial summary of the previous section is provided. In it one can observe that the monitoring information will flow through the communication infrastructure from internal COMPOSE platform components and will be distributed to interested COMPOSE entities, such as the COMPOSE cloud controller. Along the way the monitoring information can be stored in a DB to be later used by additional components.

3 API

The API is provided in Java style, since the internal implementation of the communications servers is Java based.

3.1 Information Producers

- **Creation of a monitoring information producer**

A monitoring producer is the entity which is used to create monitoring information channel senders.

```
MonProducer mp = MonFactory.createProducer(com.client.Session session,
<String producer_id>,  <String producer_group>)
```

Session – refers to the communication client via which the monitoring infrastructure will operate

producer_id – provides an identification to this producer, such that consumers can direct requests specifically to a certain producer

producer_group – provides an identification to this producer as a part of a related group, such that consumers can direct requests specifically to a certain group

- **Creation of a channel for sending monitoring information**

A monitoring channel sender is the entity which is used to publish new monitoring information.

```
MonChannelSender mcs = mp.createChannelSender(String channel_name,
com.client.EventListener event_listener)
```

channel_name – provides the channel name that will be used by this producer to disseminate its monitoring information

event_listener – provides a callback mechanism which enables this channel sender to be made aware of important events that took place in the underlying communication client.

- **Send a piece of monitoring information**

```
mcs.sendMessage(com.client.Message msg)
```

msg – the message containing the monitoring information that this producer wishes to send to all the consumers

3.2 Information Consumers

- **Creation of a monitoring information consumer**

A monitoring consumer is the entity which is used to create monitoring information channel receivers.

```
MonConsumer mc = MonFactory.createConsumer(com.client.Session session)
```

Session – refers to the communication client via which the monitoring infrastructure will operate

- **Creation of a channel for receiving monitoring information**

```
MonChannelReceiver mcr = mc.createChannelReceiver(String channel_name,
com.client.MessageListener message_listener, com.client.EventListener
event_listener)
```

channel_name – provides the channel name that will be used by this consumer to receive its monitoring information

message_listener – provides a callback mechanism which will be invoked whenever a message is received on the channel receiver

event_listener – provides a callback mechanism which enables this channel consumer to be made aware of important events that took place in the underlying communication client.

3.3 Handling of monitoring requests

- **Requests listeners on the producer side**

    mp.setRequestListener(MonRequestListener mon_request_listener)

mon_request_listener – the entity on the producer side which is in charge of receiving and interpreting requests from the information consumers (for example, send now updated information)

- **Process a request for monitoring information**

    void MonRequestListener.onRequest(com.client.Message mon_request)

mon_request – the actual monitoring request received by the producer

- **Send a monitoring request to a producer**

    mc.sendMonitoringRequest(String producer_id, com.client.Message mon_request)

producer_id – the producer which should receive this message

mon_request – the actual monitoring request to be delivered to the producer

- **Send a monitoring request to a group of producers**

    mm.sendMonitoringRequestToGroup(String producer_group, com.client.Message mon_request)

producer_group – the producer group which should receive this message

mon_request – the actual monitoring request to be delivered to the producer