Triumvir

Content

[Introduction 2](#_Toc293942793)

[Guidelines 2](#_Toc293942794)

[server and Client 2](#_Toc293942795)

[Initialization - Interactions 2](#_Toc293942796)

[Types of Interactions 2](#_Toc293942797)

[From Triumvir to Civis component inside the same instance 3](#_Toc293942798)

[Security Design 4](#_Toc293942799)

[Contact procedures 4](#_Toc293942800)

[Knowing who is the Triumvir Server 4](#_Toc293942801)

[Getting data from the local Civis 4](#_Toc293942802)

[Questions 5](#_Toc293942803)

[Work distribution 6](#_Toc293942804)

[Work resolutions 6](#_Toc293942805)

[Importance of the “Hello” procedure for the triumvir 6](#_Toc293942806)

# Introduction

Triumvir is the direct Latin translation for “three men” and is used to designate a group of three persons (in these case machines) that perform an important role inside the group where they are integrated.

A triumvir is the main communication bridge between each civis and the rest of the network.

This role is responsible for ensuring that all cliens assigned to him get timely knowledge about the status of the clan or castrum. It is also responsible for warning other triumvirs about status of his own cliens.

In case of fail or mistrust, another triumvir is elected from the list of civis under his charge. The details for elections and interactions with other roles are described in the section entitled “Triumvirate” of the Remedium Structure documentation.

# Guidelines

On this chapter we detail how the triumvir should work from a conceptual point of view.

## server and Client

We divide the Triumvir component in two separate roles: server and client.

The server is intended to provide update messages for each client in the network whenever requested. The client has the responsibility of requesting these updates messages at periodic time intervals.

The updates messages are described on their own section at this document.

## Initialization - Interactions

### Types of Interactions

The server is activated on each instance and will automatically synchronize itself with the data gathered by Civis (if available).

The purpose of this design decision is to prepare each instance to become Triumvir under the shortest amount of time possible. Otherwise we would have to wait until each newly assigned Triumvir is ready and prepared to handle clients.

The server interacts with three distinct entities:

* From Triumvir to Civis component inside the same instance
* From Triumvir to Triumvir in external instance
* From Clients in external instances to Triumvir

Over the next sub sections we detail these interactions.

## From Triumvir to Civis component inside the same instance

A Triumvir needs to gather knowledge that is generated on its own instance. This task is presently carried forth by the Civis component and we define in this sub section a protocol for both entities to communicate and exchange data.

At this particular moment of implementation, we define that Civis component should supplying new information to the Triumvir whenever it becomes available.

On Figure 1 we introduce an illustration of the proposed interaction and provide an explanation of the protocol between both entities.

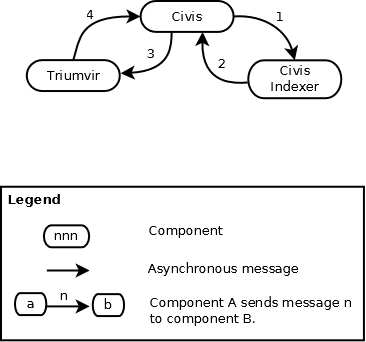


Figure - Interaction between Civis and Triumvir

The Civis component initiates the action through message #1; this message is addressed to the Civis Indexer component running on the same instance as the Triumvir. The purpose of this message is to request an update of knowledge if available.

If the Centrum indexer holds no knowledge in the repository, the date for requested update is zero. This value represents a request for complete synchronization of information between the two entities.

Using the value provided by the Triumvir as the date of previous update, the Centrum component will relay this information to the Civis Indexer.

It would have been possible for the Triumvir to contact directly Civis Indexer and prevent an intermediary in this process, but we opted as a design decision to keep the Centrum component as contact point to ensure that in future changes to the architecture of Civis, we maintain the main Civis component as the pivot which knows where these messages should be relayed.

This design decision also allows centralizing the requests made to Civis and handles them efficiently in case of authorization management and load balancing.

## Security Design

We define that each Triumvir should only allow registration for 1000 clients. This restriction provides a safety mechanism to ensure that attackers do not overrun the resources from the machine running the Centrum service.

An attacker might still fake the registration of Triumvir clients up to the maximum number of available slots; however, this action should trigger an alarm message that is dispatched to alert the system administrator.

# Contact procedures

On this section we detail the contact interactions between the server and client roles.

## Knowing who is the Triumvir Server

A client triumvir needs to know where the server triumvir can be found.

This value is provided by the centrum client to the triumvir client.

When the centrum client in Remedium B gets in contact with the centrum server at Remedium A, the centrum server at Remedium A provide back an answer to remedium B that specifies where remedium B can find a his designated triumvir.

## Getting data from the local Civis

To avoid duplicating data, the Triumvir component has access to the database where all data containers are stored.

Access to these databases is not direct, it is provided by methods provided by the Civis which are kept as read-only by design.

# Questions

**Triumvir local**

*How do we know who is our triumvir server?*

- The centrum client is responsible for providing this answer during the “Hello” procedure

- The same procedure also activates and deactivates the Triumvir server

*How can a triumvir know that changes occurred locally?*

- We keep a log of data inserted/removed/modified inside each container. The triumvir client stores the data from his last ping and requests changes since the previous ping. If there is new data then it will be relayed to the Triumvir server

*How is this new data moved upstream (relayed to the triumvir server)?*

- The triumvir client will dispatch one message with 10 000 records per time (weak solution); the server provides an acknowledgement that this data was received.

Preferably, a checksum can ensure data integrity between transfers

- The envelope with data will also contain information about each container where data is placed (database name, number and type of fields)

**Triumvir external (interacting with clients)**

Receives data from authorized triumvir clients (needs to verify if they are registered or not).

*How do we know which remedium instances are registered to use us as triumvir?*

- The centrum is responsible for providing a list of remedium instances authorized to interact with us

- Any contact from remedium instances not present on our list is discarded and/or logged as suspicious (and possibly relayed back to the centrum as warning of suspicious action)

*How is this list composed?*

- The list must contain IP addresses and ports where remedium instance are running

- An expiry date is set and expired clients are removed from the list if they don’t say “hello”

- The list is updated using a message originated from our centrum client

*Where is new data stored?*

- The triumvir must be capable of creating containers that store the data coming from each remedium instance. This means that a triumvir should create these containers on the fly using the provided data on each envelope

*How do we decide which data to keep/share and which data to discard?*

- This is a task for the Quaestor role

- Our triumvir shall have a database container called *“external”*, containing the data intended for synchronization with clients

*How is new data from the triumvir server moved downstream to the triumvir client?*

- Data is distributed using envelopes that contain the data records and information about the required containers to store them

- New information is added, updates and/or deletes older information

- Clients only accept data envelopes from their designated triumvir servers

**Triumvir external (interacting with other triumvirs)**

As rule of principle, each triumvir only knows another triumvir in the circle. Our triumvir will interact with the other triumvir as if it was a triumvir client and not disclose the fact that it is a server. Using this principle we increase awareness to detect attempts of distributing malicious information.

# Work distribution

# Work resolutions

- Each database container should store and provide a log of changes that can be exported

- The exported log should allow adding data to a container located elsewhere

- Centrum client must check for the presence of messages for the triumvir (client and server) when proceeding with the “Hello” procedure

- Triumvir should be capable of creating a new database container for each authorized client

- Triumvir should create containers that accommodate the data from each instance

- Envelopes that contain records and information about each container must be made available

- Containers used by the Civis/Analyzer must be created/managed dynamically

## Importance of the “Hello” procedure for the triumvir

The “Hello” procedure from the Centrum is responsible for:

- Informing the triumvir client regarding who is the designated triumvir server

- Activate and deactivate the functioning as triumvir server

- List the authorized instances in case the local instance is a triumvir server

## Implementing the log of changes for each container

The log should provide import and export features for the container.

*How are we going to store changes?*

We need to compute all changes in the container since a given period of time (easy to say, difficult to implement)

- Added data

- Removed data

Where is the log of changes kept?

- Each container can add a special table for this purpose

- Proposed fields:

- Unique index

- Type of action (add or remove for the moment)

- Date when it was added

- Content (how can this handle multiple fields?

Can we store the SQL statement instead of a proprietary format?

- Yes, we can – but how do you handle the fields?

- Proposed fields:

- Unique index

- Date of action

- SQL statement

- Brings benefits:

- Already supports data being modified

- Clean and exportable

- Simple to implement, just record each statement at another table

- Disadvantages

- We have no control over SQL statements

- Uses a lot of text

- Additional processing overhead

- Text has to be encoded to base64 otherwise causes SQL injection

Methods to modify:

- delete()

- write()

**Others**

Is there need for keeping a log of changes on the database?