Imtech

GAIUS

Realization of a
Generic Automated Integrated Universal System for wind tunnels
and a pilot implementation

For
DNW

Architecture Description

Editor: Evert van de Waal

Version: 1.0

Date: 13 Oct 2007

Version Management

Version	1.0
Status	Final
Date	13 Oct 2007
Classification	Public
Author	Imtech ICT

Version History

Version	Date	Status	Author	Comments
0.2	1-10-2007	Draft	E. vd Waal	Major changes after discussion
0.3	8-10-2007	Proposal	E. vd Waal	Textual overhaul after review
1.0	13-10-2007	Final	E. vd Waal	Minor modifications after review

This document was created with contributions from:

Marcel Smit
Pieter Goudzwaard
Bas Snelders
Kristian Kievith
Gerard Morsink
John van Orsouw
Jack het Lam
Evert van de Waal

Edited by:

Evert van de Waal

Contents

Summary

Goal of the GAIUS project was to create a new wind tunnel automation system in which the various subsystems of a wind tunnel can be integrated and automatically controlled. Here, the core choices made in the design of GAIUS are described and justified.

The interactions between DNW personnel and the GAIUS systems are categorized. The following people (roles) have most interactions with GAIUS:

- DNW Management
- DNW Project Engineers
- DNW System Installers
- Tunnel Operators
- DNW software developers

Their needs are summarized, and from them architecture principles are extracted, which would lead to a successful system:

1. Use of Open Software
2. Use of Python as the core programming language
3. Use of industry standards
4. Use of the GAIUS bus
5. Use a standard module interface
6. GUI's are separate components
7. All wind tunnel hardware is simulated

These principles are used to form the actual architectural design, which is presented.

References

[1]	Titel:	FIASCO Request For Proposal
	Author:	DNW
	Versie:	February 2006
[2]	Title:	FIASCO Functional Requirements Document
	Author:	DNW
	Version:	February 2006
[3]	Title:	Financial Proposal GAIUS
	Author:	V. van Swinderen
	Version:	April 2006
[4]	Title:	Introduction to FIASCO
	Author:	DNW
	Version:	May 2006
[5]	Title:	UI Style guide for DNW
	Author:	Okko Willeboordse
	Version:	29 sept 2007
[6]	Title:	GAIUS development documentation
	Author:	Imtech
	Version:	Oct 2007

Please note that these documents are confidential and will not be available for the NK Architecture.

List of abbreviations

DNW	Duits Nederlandse Windtunnels (German-Dutch wind tunnels)
GAIUS	Generic Automated Integrated Universal System for wind tunnels
NLR	Nationaal Lucht- en Ruimtevaartlaboratorium (National Air & Space lab)
DLR	Deutsches Zentrum fur Lüft- un Raumfahrt (German center for air and space travel)
SST	Super-Sonic Tunnel
KRG	Kryo-Rohr-Windkanal Göttingen (Cyro wind tunnel Göttingen)
BOM	Bill Of Materials
GAS	General Automation System
TOPAC	Test Object Position and Attitude Control
MMI	Man Machine Interface
WDF	Wind tunnel Data Format
APROPOS	Aerodynamics Processing and Presentation Open-ended System, DNW data presentation system
RTPS	Real-Time Publish Subscribe protocol
SOAP	Simple Object Access Protocol: A protocol often used in middle-ware layers.
CORBA	Common Object Request Broker Architecture: A middle-ware layer.
DCOM	Distributed Component Object Model: A proprietary Microsoft middle-ware layer.
(G)UI	(Graphical) User Interface

1 Introduction

1.1 DNW

Duits Nederlandse Windtunnels (German-Dutch wind tunnels, DNW) is an organization that owns and operates wind tunnels. It is a joint venture by the Dutch and German research institutes NLR & DLR.

DNW is responsible for the maintenance of the wind tunnels, the acquisition of wind tunnel research contracts, the preparation and execution of wind tunnel experiments and the processing of experiment data.

Wind tunnels are extremely powerful, sophisticated machines. To illustrate:

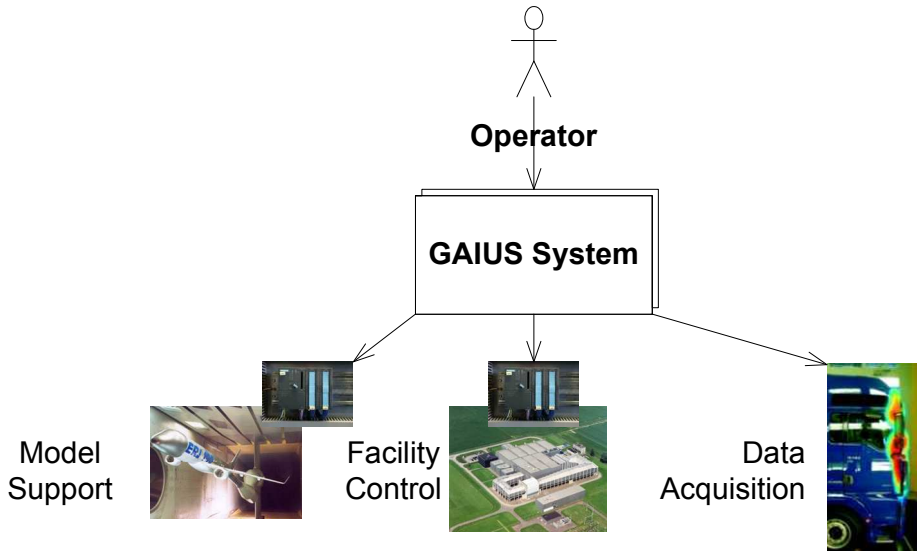
> The Super-Sonic Tunnel (SST) facility has a pressure container of 600 m^3 at 150Bar pressure. This pressure is released in 20 seconds, reaching an airspeed of Mach 4. It generates one of the strongest sound in Europe, with oscillations of 0.8 Bar.
>
> Some wind tunnels are capable of power changes in excess of 4MW/s. These tunnels are capable of causing a major blackout if actions are not coordinated properly with power companies.
>
> The Kryo-Rohr-Windkanal Göttingen KRG (cryo facility) consumes one truckload of liquid N^2 every few minutes at full operation.
>
> Models being tested are positioned with an accuracy of 0.01 degree, at loads of e.g. 70kNm.
>
> Some experiments are conducted with high power lasers that charge the air so that pressure variations cause light to be emitted. In other experiment, droplets in aerosols are tracked to determine airflows.

Priorities for DNW during measurements are:

1. Safety of the people involved.
2. Safety of the facility itself.
3. Safety of the model under test.
4. Quality of the measurement results.
5. Being time-efficient.

The technology needed to perform experiments constantly changes. Thus, the tunnel systems need to be constantly modified and updated to accommodate new experiments. Often specialized equipment is needed to perform specific tests.

1.2 Goals for the GAIUS project



The Generic Automated Integrated Universal System for wind tunnels (GAIUS) project was initiated by DNW in order to develop a new automation system that integrates all the various systems needed to operate a wind tunnel. GAIUS is intended to be used in all DNW wind tunnels, to achieve the following goals:

- **Have a unified approach to operating a wind tunnel**, to better facilitate exchanges of people and instruments.
- **Improving wind tunnel efficiency** by allowing experiments to be automated through a scripting language.
- **Improving wind tunnel accuracy**: by interconnecting the various systems in a wind tunnel, higher-order control loops can be made than when the systems are not connected.

1.3 People interacting with GAIUS

In the DNW organization, people will interact with GAIUS in various ways. To better understand the GAIUS system, the people interacting with it will be grouped in *roles*. Each role views the GAIUS system in a particular way.

The DNW roles that have most interaction with the GAIUS system are:

Management. This role is responsible for directing the people in DNW, and allocating funds. Also, they are responsible for the profitability of a wind tunnel, and the short and long term operation of the facilities.

Project Engineer. This role is responsible that the right experiments are executed. Together with the DNW customer, they will prepare a test series, supervise the execution of the tests, and present the measurement results.

System Installer. This role is responsible for maintenance of tunnel systems, and also for the preparation / modification of the models to be tested.

Tunnel Operator. This role is responsible for the actual operation of the wind tunnel. The experiments must be performed as desired by the Project Engineer, whilst maintaining high standards in safety, quality and efficiency.

DNW Developer. As the types of experiments and the technology used to perform them constantly change, software changes such as interfaces to new systems need to be made constantly. DNW has developers capable of making these modifications.

1.4 Approach of this document

In this document, first the manner in which the roles interact with the GAIUS system are studied. Then, the core principles underpinning the GAIUS system will be explained and justified from the ways the roles interact with GAIUS, and their specific needs. Finally, the technical decomposition of the GAIUS system will be given.

2 System Roles

In this chapter, the ways in which DNW interacts with the GAIUS system will be further explained. From the interactions, the main requirements of GAIUS are derived. In the next chapter, this analysis will be used to determine the core principles underpinning the GAIUS system.

The interactions with GAIUS were investigated using the different views of GAIUS. The views of the following roles will be investigated:

- Management
- Project Engineer
- System Installer
- Tunnel Operator
- DNW Developer

2.1 Role: Management

2.1.1 Description

DNW management is responsible for the long term and short term running of the wind tunnels operated by DNW. It is responsible for short-term planning & optimization of operations, and the long-term maintenance & improvement of DNW services.

2.1.2 Activities

- Assigning personnel
- Scheduling of wind tunnel experiments
- Financial management of wind tunnels
- Training & development of personnel
- Improve co-operation between wind tunnels
- Marketing of wind tunnel services

2.1.3 Interaction with GAIUS

Management interaction with GAIUS is largely indirect, except for expenses due to GAIUS-related projects.

Indirectly, management interacts with GAIUS in the following ways:

- Management needs to take care that people have the required skills & knowledge to work with GAIUS.
- The efficiency of a wind tunnel is improved by using GAIUS.
- When deploying GAIUS in a wind tunnel, time needs to be allocated for testing.

- Due to the capabilities of GAIUS, new possibilities are created which influence the market value of the wind tunnels.
- When maintaining a wind tunnel over a long period of time, systems become obsolete and need to be replaced with updated systems. The GAIUS system must facilitate this.

Especially for wind tunnels that are in high demand, care must be taken that the down-time due to installation or maintenance of GAIUS-related systems is as small as possible. Step-wise implementation is essential to achieve this, as well as verification of the GAIUS system in a simulated environment.

2.2 Role: Project Engineer

2.2.1 Description

The project engineer is responsible for the design of experiment execution procedures. Because these experiments are very costly, they need to run with little or no delays and yield the maximum amount of information. The project engineer designs a recipe for the experiments so the wind tunnel is configured correctly to measure and record the parameters of interest , along with all the process parameters.

After experiment completion, the results will be available in a format that can be used by the customer for modeling and presentation of the information. The project engineer sees to it that the customers can use this data as input for further analysis.

2.2.2 Activities

The project engineer will perform the following tasks before the execution of the experiment:

- Verify that the necessary data is available at the start of the experiment.
- Verify that the data is correct.
- Create the program for the wind tunnel parameters to be applied during the experiment.
- Verify that the execution of the experiment is physically possible.
- Assure the experiment does not damage the equipment or the testing materials.
- Check that during the experiment, all necessary data can be measured.
- Make an estimate of the results that can be expected.

When the experiment has finished, he should be able to perform the following tasks:

- obtain the measurement data
- obtain the data recordings of process
- convert the data to a format that can be used for analysis and presentation

2.2.3 Interaction with GAIUS

The GAIUS system contains a preparation module that enables the project engineer to create a recipe for experiment execution. This module runs in a safe environment, not directly linked to the wind tunnel. Here the input parameters for the wind tunnel are determined that will applied during the experiment. The geometry of the object being tested is entered into this module. Also the positions of the sensors on the surface of the model are entered.

The preparation module automatically determines the limits for the input parameters for the wind tunnel. The positions and movements that are planned for the execution are checked to assure that these are physically possible. The preparation module saves the parameters and data in a format readable for the wind tunnel.

The GAIUS system contains a simulation module that uses the recipe as input and performs the test in an environment with simulated hardware. This module tests that the data is correct and that the test can be executed.

The GAIUS system contains the “GAIUS bus” for the controlling of the test and the acquisition of the data. The GAIUS bus records the data during the execution of the test. The data is stored in “WDF” format – the standard format for wind tunnel data.

In the near future, the GAIUS system will be extended with a new presentation module. The requirements for this presentation module are still under discussion. For the moment this module only contains an interface to the presentation system APROPOS. The GAIUS presentation module is designed in such a way that this extension can be added easily.

2.3 Role: System Installer

2.3.1 Description

The installation crew is mainly occupied with the configuration as delivered by the project engineer. The required configuration is handed over to the installation crew, who then adapt the systems accordingly. The configuration entails the model to be tested, the required wind tunnel facilities, the required sensors and measurement equipment, the necessary computer systems, and the way the software must be configured.

When problems occur during a test, the installation crew may need to make further adjustments to solve them.

Often, the customer has additional requirements concerning the way the model is treated, site security, computer systems, and the handling of measurement results. The installation crews needs to accommodate these requirements.

2.3.2 Activities

The installation crew performs the following activities:

- Mechanically installing the model to be tested.
- Installing and connecting the various sensors in the model
- Mechanically and electronically preparing the wind tunnel systems
- Installing and configuring the necessary computer systems

- Modify the computer infrastructure (e.g. the routing of IP packets)
- Replace GAIUS hardware in case of breakdowns.

2.3.3 Interaction with GAIUS

The main interaction with GAIUS for the installation crew is the deployment of the various computer systems. For instance, the GAIUS system needs to access the file system where the test data is stored. Often, the customer brings his own file storage to prevent leakage of sensitive information. The GAIUS system then needs to be reconfigured correspondingly.

The installation crew needs to have a good understanding of the way GAIUS communicates with other systems to do this properly.

The installation crew needs to check if the connections of the measurement equipment are configured correctly in the GAIUS system.

When problems occur, the installation crew may need to co-operate with the operator to diagnose and solve them. The information provided by the GAIUS system will then be vital.

2.4 Role: Operator

2.4.1 Description

The responsibility of the operator is to operate all systems in a wind tunnel, such that the required tests are performed, and correct results are obtained. He co-operates closely with the Project Engineer.

The operator is also responsible for wind tunnel safety.

2.4.2 Activities

An operator performs the following activities:

- Check that the system is ready to perform a test.
- Perform the procedures necessary to initiate, perform and complete a test.
- Monitor the systems.
- Bring the systems to a safe situation if safety is compromised.
- Diagnose any problems that may occur and solve them.

2.4.3 Interaction with GAIUS

The operator of a GAIUS-equipped wind tunnel will heavily interact with GAIUS.

- The GAIUS Man Machine Interfaces (MMI's) are used to check the status of the systems, issue commands to the system and monitor the events in the system. These MMI's are essential for the efficient operation of the system.

- The log messages that the GAIUS system generates are vital for the operator to quickly diagnose and solve problems.
- With the GAIUS automation system, recurring tasks can be automated to improve wind tunnel efficiency and decrease operator workload. Sometimes these require parameter input, for which the operator may be queried.
- The GAS system allows some status checking and exception handling to be automated.

As the MMI's are essential to operators, these are carefully designed. A style-guide [5] has been defined which gives the accepted guidelines for MMI implementation.

2.5 Role: Developer

2.5.1 Description

The developer creates modules which will run on the computer-systems. The most important issues for the developer are:

- Create modules.
- Test modules.
- Trace and debug the modules.

For creating modules, the developer wants tooling to create modules properly. The programming-language should be easy accessible and maintained in an environment which has debug-facilities.

Created modules must be tested and during these tests logging should be produced by the system. The logging will help tracing why errors came up.

2.5.2 Activities

The developer will need to perform the following activities with the GAIUS system:

- Test the software in a safe environment, where no damage can be done the tunnel hardware.
- Trace and recording events in the GAIUS system.
- Debug running software.
- Statically check the code.

2.5.3 Interaction with GAIUS

Developers interact with the source-code of the GAIUS system. They add modules, modify existing modules, and then run these to improve and debug them.

A common task will be to build a bridge between GAIUS and a new system. Thus GAIUS is a very open system, which is easy to interface with.

For efficient development, reuse of existing code is important. The developer will make use of the standard frameworks used to build GAIUS modules to increase productivity.

After or during development, the developer wants to test and maintain the created modules. A tools are included in the system which help the developers:

- Trace, logging and recording tools.
- Debug-environment.
- Tools to check code. (PyLint)
- Simulation environment.

The possibility to simulate the module in the system enables detection of errors before the module is loaded in the real-live-system.

The documentation [6] defines interfaces and command, enabling developers to quickly learn how to program in the GAIUS system. Also, Imtech will train DNW developers.

2.6 Summary: Key GAIUS characteristics

From the views presented above, the following key characteristics can be determined:

- GAIUS is an open system, with open interfaces to which new systems can be attached easily.
- The data available in parts of the system is easy to share with other parts of the system.
- It is easy to use ‘scripts’ to automate the control and measure systems.
- It is easy to test & improve scripts in a simulated environment, without using the actual wind tunnel.
- The GAIUS system is adaptable to accommodate the differences between various wind tunnels, and also changes in a wind tunnel system.
- The GAIUS system is cost-effective.
- The GAIUS system runs on readily available hardware (PC’s).
- The GAIUS system is supported for a long period, in the order of 10 years.
- It is possible to deploy the GAIUS system gradually, and with as little interference with tunnel operations as possible.
- The configuration of GAIUS is easy to read and edit.

3 Architecture Principles

In the previous chapter, the ways in which the different roles operating within DNW view the GAIUS system, and their key drivers were analyzed. In this chapter, the core principles for the GAIUS are presented. Alternatives to the choices made in this chapter are also discussed. The analysis of the previous chapter will be used to justify the choices made.

3.1 Principle: Use Open Source software

In the GAIUS system, a large amount of Open-Source software is used as building blocks. This is one of the core principles of the GAIUS system. Due to the use of Open Source software, the system can be developed quickly. Open Source software libraries exist to perform many different functions that are useful for the control of a wind tunnel. Also, Open Source software is usually designed to interface well with other Open Source projects, encouraging a programming style where chunks of Open Software are glued together to create a working system.

Two alternatives exist: developing all blocks specifically for the GAIUS project, and the use of closed-source building blocks. To develop everything specifically will be prohibitively expensive, and is thus reject. The use of closed-source software has several drawbacks:

- With closed-source software, it is more difficult to experiment and see if a possible solution gives the desired functionality.
- Closed-source software can be difficult to interface with. If interfacing problems arise, they can be very hard to solve. Open Source software is usually easy to interface with, because Open Source software is usually designed to interconnect with software developed in other Open Source projects.
- With closed-source software, a dependency is created to the supplier. Some suppliers don't support their software for long periods of time, creating a liability for wind tunnels. With Open Source software, we have complete control over the software.

An advantage of closed-source software is that sometimes its quality is higher than for open-source software. However, in advanced applications usually problems are found even in closed-source software. It is often very hard to have these problems solved, due to the closed nature of the software. In open-source software, problems are relatively easy to solve.

With Open Source software, and esp. the Linux platform, there are very powerful options to tune performance and Real-Time behavior. The standard Linux kernel has good soft real-time behavior, which can even be improved upon by applying well-known patches. The amount of control and ease of tuning is superior to that offered by e.g. the Windows platforms.

3.2 Principle: Use Python

In GAIUS, the Python programming language is used as much as possible. This is an important principle of the system.

Python is a programming language developed by Guido van Rossum in 1991. It is compiled to byte code, which is then interpreted.

The main reason to use Python is that it is a language that is very easy to program. Typically, a program written in Python is significantly smaller than if it were written in e.g. C# or Java, and much smaller than the same program written in C++. Also, Python is a very open language, with a simple yet powerful interface to code written in other languages. Thus it is the ideal language to 'glue' different components together. As GAIUS uses much open source software, most GAIUS programming is to glue components together.

Viable alternatives to Python would have been:

C++, an established, extremely capable language. However, C++ is also a very difficult programming language, reducing the ease with which DNW can modify the system. Still, C++ is used to create some parts of the GAIUS system where performance is critical.

C#. However, C# is not platform independent. Also, it is harder to program than Python.

Java. Java is platform independent, but it is harder to interface to C libraries in Java.

Ruby. Ruby is another interesting language, but less mature than Python. Also, Ruby is a pure interpreted language, thus its performance is much worse than Python's. It has a very abstract syntax, which is less easy to learn than Python's syntax.

3.3 Principle: Use Industry Standards

In GAIUS, industry standards are used for communication as much as possible. The main advantage is that by using existing standards, GAIUS is easier to connect to than if interfaces were dedicated. Also, it is possible to use existing tools to support the system. And finally, industry standards are being actively developed. Thus it will be easier to incorporate future developments into GAIUS when industry standards are used.

3.4 Principle: Use the GAIUS bus

In GAIUS, information is shared between the modules using a publish-subscribe bus. This means that information is published, regardless of which modules are using this information. Thus it is very easy to share information in ways that have not been foreseen during development, making it easy to extend the GAIUS system. New control loops are simple to create using this bus. By using this bus, in effect an implementation of the Blackboard architecture pattern is created, where all information is available to all modules of the system. The information is used by the modules at their discretion.

The GAIUS bus will be more fully described in the next chapter. However, at this point it is relevant to mention that it is based on an existing open source implementation of the Real Time Publish Subscribe (RTPS) protocol, following the Use Open Source and Use Industry Standards principles.

Alternatives to the GAIUS bus are:

- Using an alternative middleware layer, e.g. SOAP, CORBA or DCOM. However, in these systems the focus is not on information being shared, but on services. The GAIUS system is intended as a measurement and control system, in which the focus is on the data being generated by the various subsystems, not on services. Thus these middleware layers are not suitable.
- Using dedicated point-to-point connections. In such a system, the dataflow is very clearly defined. However, in the GAIUS system, it is not possible to foresee all ways in which data will need to be routed through the system. Point-to-point connections would severely restrict the GAIUS system.

3.5 Principle: Use a standard module interface

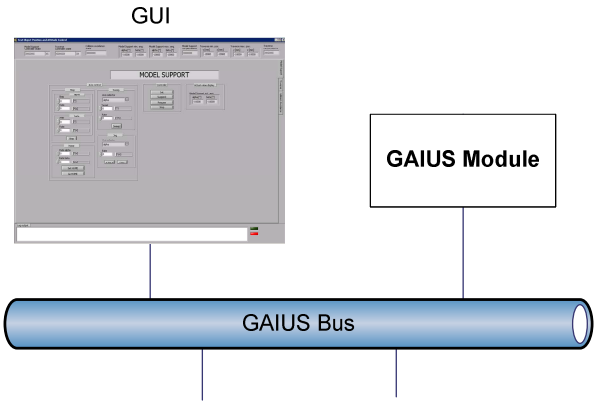
As modules connect to the GAIUS bus, it is important that they expose a standard interface. Due to this interface, it becomes much easier to control these modules using automatic scripts etc. Also, it becomes easier to route information through the system in ways currently not foreseen. As a side effect of standard interfaces, the modules have a highly standardized structure, where much code can be re-used. This is advantageous both in development costs and ease of maintenance.

The interfaces to modules can not be completely identical, however. As each module performs specific tasks, they must produce specific types of information. To interpret this information, knowledge about the specific module is required. This is allowed in the GAIUS system.

Alternatives to this solution are:

- Introspective Interfaces. It is possible to create a method of introspection, through which it is possible to automatically query modules to discover their interface. However this solution is very complex. It was judged that it is not cost-effective for the GAIUS system to pursue it. Also, GAIUS will be maintained by people who are very knowledgeable about the exact information produces & consumed by modules. Thus advanced introspection is not necessary.

3.6 Principle: GUI's are separate components



GUI as a separate component

In GAIUS, there is a strict separation between modules and the GUI's through which operators interact with these modules. The Graphical User Interface (GUI) communicates with the module solely through the GAIUS bus. This has the following advantages:

- No vendor lock-in. This solution allows the use of highly advanced GUI-building environments without the fear of e.g. vendor lock-in, as changing GUI technology has a small impact on the complete system. For example, it is easy to gradually change GUI technology, one GUI at a time, once a GUI needs to be updated. No big-bang change is necessary.
- No difference between scripted or manual control. As far as the modules are concerned, there is no difference between manual or scripted control. This means that their implementation becomes very clean, and ensures that both ways of controlling the module are capable of all necessary functions. Also, it becomes easy to perform automatic tests on the modules.

- Support for multiple interfaces. It becomes trivial to have the modules controlled in different ways. A GAIUS module can be controlled just as easily from a terminal as from a GUI or from a script. Also, redundancy is supported. Multiple GUI's can be instantiated, where the operation of the module is monitored in different locations.

The alternative to this is to build GUI's within the modules. This would lead to a slightly simpler system. However, it would not be as easy to control from a script as in the current situation. Also, applications with built-in GUI's are notoriously difficult to design, often degrading to spaghetti where the GUI code is interwoven with application logic. A properly designed application uses the command design pattern to separate application logic from the GUI code. By making the GUI a separate component, use of the command pattern is automatically enforced.

3.7 Principle: Simulate all hardware

In order to minimize interference with tunnel operations, it must be possible to test the GAIUS system separately from the actual wind tunnel hardware. This is achieved by making a software simulator of this hardware. This simulator is then used for testing the GAIUS system before it is deployed. Also, it is used during the preparation phase of a tunnel test, to check if the configuration is complete and correct.

Alternatives are not viable. As a wind tunnel system is extremely expensive, it is impossible to sufficiently test GAIUS purely on the wind tunnel. Instead of a software simulation, a hardware simulation could have been made, but this is also very expensive and insufficiently flexible.

4 Modules & Interactions

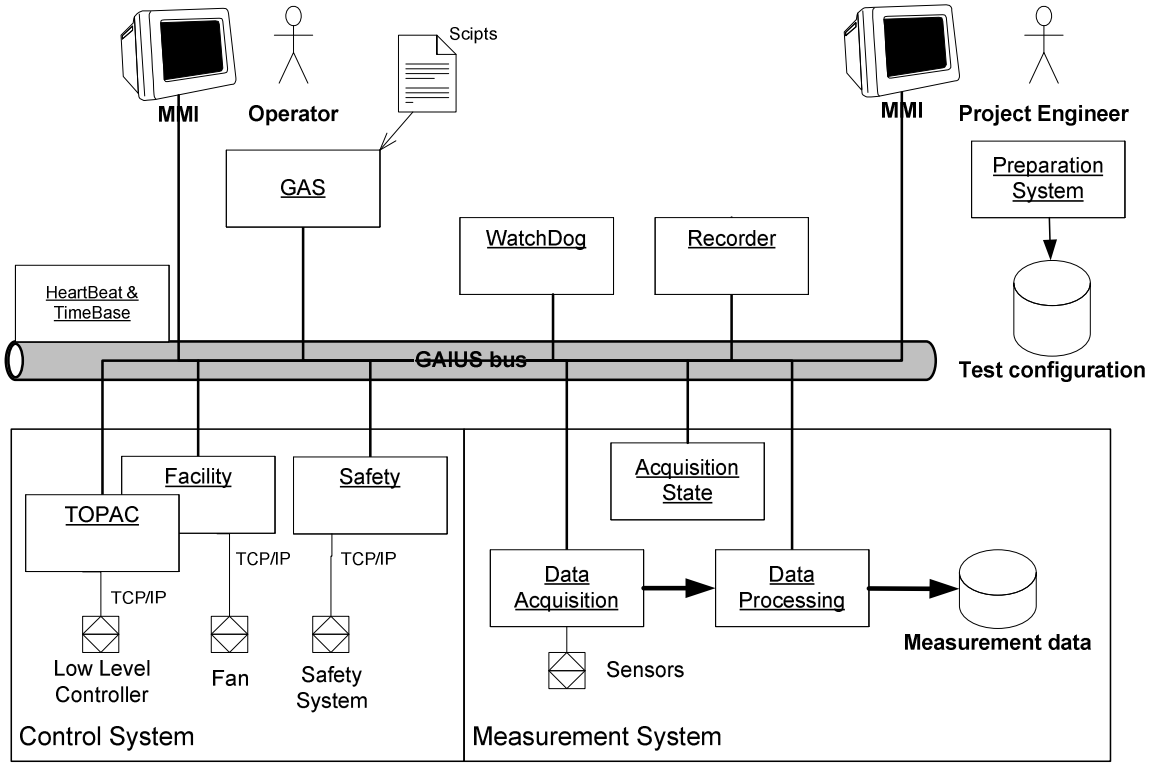
After determining the architecture principles, the architectural design can be determined. In this chapter, various aspects of the architecture are described. Most attention will be given to the decomposition of the GAIUS system into interconnected modules, which of course is the heart of an architecture description. At this time, it is not appropriate to give all the details of the modules; these can be found in the requirements [2] and [4]. Most modules behave very similarly, and this common behavior will be described. Attention is also given to the ways the modules communicate, and the underlying principles of the GAIUS bus.

Finally, several system-wide issues are discussed: the configuration of the GAIUS system, the deployment of GAIUS, and running GAIUS in a simulated environment.

4.1 Module Interconnections

In this section, the main concepts of GAIUS will be briefly explained. The core modules are given, and the main concepts & philosophies are explained.

To help understand the place each module or system has in GAIUS, the following figure gives an overview:



GAIUS System Overview

In section 4.1.1, a short description of these parts will be given. In section 4.1.2, the concepts of Automation Level and Run Permissive will be explained. In section 4.1.3 the safety concept of GAIUS will be defined.

4.1.1 Description of modules

The modules in GAIUS can be divided in three groups:

> **Infrastructure**: These modules are the heart of the GAIUS system. These are the GAIUS bus, and the heartbeat generator. Without these, GAIUS can not function. As these parts are so essential, they will be described further in section 4.3.
>
> **Controlling modules**: These modules control the various sub-systems of the wind tunnel. These are Facility, TOPAC, Safety, Data Acquisition and Data processing. The latter from the Measurement System.
>
> **Supervisory modules**: These modules allow the control of wind tunnel as a single unit. These are: GAS and Acquisition State. The Preparation system is used off-line to prepare future tests.
>
> **Supporting modules**: These modules support the other modules. Examples are: Watchdog and Recorder, and the data storage.

The responsibilities of these modules will be briefly described below.

GAS module: The General Automation System (GAS) module is the environment in which user-developed scripts can be run to control the GAIUS system. All other modules can be commanded from the GAS module.

Acquisition State module: This module knows which test is being performed by GAIUS, and broadcasts this information so that modules can retrieve test-dependent data.

Facility module: controls the various facilities offered by the wind tunnel, which provide the air flow and condition it. These include e.g. the main fan, facilities to control temperature, humidity, pressure etc.

TOPAC module: The Test Object Positioning And Control (TOPAC) module controls all moving objects in the measuring location, including the positioning of the model being tested, measurement probes etc. TOPAC communicates with the underlying controllers to co-ordinate and correct all movements. It also performs collision avoidance.

Data Acquisition module: collects and stores all measurement data. Data is stored in a specific file system.

Data Processing module: filters the measured data and presents them.

Safety module: the interface to the safety features of the tunnel, insofar these are not handled in other modules.

GAIUS bus: the vehicle through which the GAS module communicates with the other modules. Also, the MMI's connect to the modules through the GAIUS bus.

Heartbeat & Time base: Part of the GAIUS bus, which synchronizes all modules.

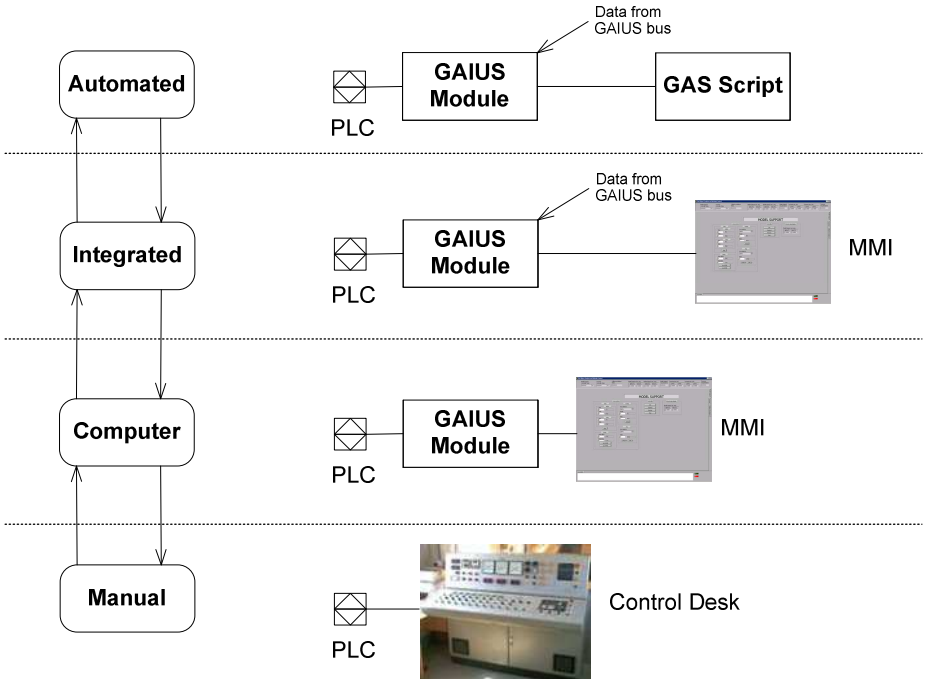
Watchdog: checks the other modules, and restarts modules that are not responsive. Also the Watchdog boots the system.

Two roles are direct users of the GAIUS system: the wind tunnel operator and the Project Engineer. The wind tunnel operator is responsible for executing the experiments and guarding safe operation; the project engineer is responsible that the right experiments are performed, and that correct results are obtained. The project engineer is also responsible for providing the test configuration, which is used by the other modules to get the right parameters for the experiments.

Finally, there is a **recorder module** that logs all events on the GAIUS bus.

4.1.2 Automation Levels & Run Permissives

As GAIUS is a distributed system, a scheme is necessary to determine how the various modules should interact. For this, the automation level concept is used.



The GAIUS Automation Levels

The system can operate on different levels of automation: Manual, Computer, Integrated, Automated, as described in the specifications [2]. The interactions between GAIUS components differ for each level:

> **Manual**: The GAIUS modules are not performing any control; they only monitor the current state of the sub systems. The wind tunnel systems are controlled directly by the operator.
>
> **Computer**: The GAIUS modules perform simple control tasks. They are controlled directly by the operator. The modules do not communicate with each other. They use only inputs from the sub systems, no data from the GAIUS bus is used.
>
> **Integrated**: The GAIUS modules are controlled directly by the operator, but they use data from other GAIUS modules through the GAIUS bus. GAS is not operational.
>
> **Automated**: The GAIUS modules can be controlled by GAS.

Transitions occur as follows:

- At start-up, a module will be in the lowest automation level: manual. Using a switch, the operator can release a sub-system for automatic control. This means that GAIUS will never suddenly perform actions, without explicit clearance by the operator. At any time, the operator can take a sub-system back to the 'manual' level by revoking the clearance for automatic control.
- If there is a clearance for automatic control, the module will switch immediately to the 'computer' level. It will then remain in this level, unless another party requests a higher level.

- The ‘integrated’ level is entered when the operator starts a control function in which a module uses information from the GAIUS bus to implement high-level control loops. It will monitor the validity of the received information, and will fall back to ‘computer’ mode if necessary.
- When a GAS script is started, it will request that all involved modules go to the automated level. If one of the modules involved does not switch to automated level, or falls back to a lower automation level during script execution, an error will be generated by the GAS module.

Run Permissives

Run Permissives are a simple scheme with which it can be checked if a wind tunnel is ready to go. Each module produces a state, the Run Permissive, which indicates if it is ready for a run. A module can perform a number of checks to see if a run is permitted or not. Three possible states exist: GO, NOGO or Unknown. The Unknown state is used when a check needs to be performed explicitly, i.e. when the run permissive is not continuously monitored by the module. Using a command, the module can be asked to check for run permission.

The run permissives will be checked at the start of a run by GAS.

4.1.3 Safety concepts

Safety in GAIUS is very important, due to the extremes of power, wind speed, pressure or temperature that can be involved in wind tunnel operation. Thus a clear and simple safety concept is needed. The main concept is as follows:

> *Each part of the wind tunnel is responsible for its own safety.*

As GAIUS is a supervisory system, where sub-systems are combined to form a larger system, the main safety issues need to be handled by the lower level sub-systems. Because GAIUS is a distributed system with communication links that are not guaranteed to be safe, this is even more the case.

However, by adding sub-systems, more advanced safety features can be implemented that are not possible with the low-level systems. A clear example is collision avoidance. To implement collision avoidance, the current configuration of wind tunnel, models, probes and armatures involved in a test must be known, as well as the current and future position of every moving item. This is known by the GAIUS system, but not by the low-level systems. Thus GAIUS can improve safety and reduce operator workload by using its information.

If an error occurs, this error must be propagated to any module or system that is dependent on the system in error. This can be done in two ways:

1. Enter the error state.
2. Stop transmitting information that is no longer reliable.

Modules at a higher hierarchical level commanding the module / system at fault will monitor the error state, and handle errors appropriately. Modules at the same level of hierarchy using information published by the module / system at fault will monitor the rate at which information is published. If items have not been updated for a certain period, the subscriber will interpret this as an error and handle it suitably.

Low level modules / systems need not monitor the state of a higher level module, as the low level module it responsible for its own safety. However, it is advised that they do monitor activity of the higher level module using a watchdog-like scheme. For example, when in ‘automated’ mode, and a module should revert to ‘integrated’ mode if no activity is seen from the higher level module for a certain period.

4.2 Standard module interface

Much of the functionality and interfacing of GAIUS modules is generic, so that they fit readily into the GAIUS system. In this section, the generic interface and functionality will be described.

4.2.1 Standard inputs and outputs

All GAIUS modules connect to the GAIUS bus, and behave in a common pattern. Each module has an unique name, which is used as a prefix for the data ID when publishing and subscribing. The following standard subscriptions and publications are made by a module:

Identifier	Type	Description
<Name>	Command subscription	For receiving all commands to the module. Commands can be issued by GUI's, by GAS, or an other way.
<Name>auto_level	Publication	Shows the current automation level mode of the module (Manual, Computer, Integrated or Automated)
<Name> task_stat	Publication	Shows the current Task state (Off, Initialize, Idle, Starting, Changing, Stopping, Keep)
<Name>comm._stat	Publication	Shows the current command state (Off, No Command, Unable, Executing, Suspend, Achieved)
<Name>_RUN_PERMISSIVE	Publication	Shows the current run permissive (UNDEFINED, GO or NOGO)

4.2.2 Standard Commands & Events

The following commands can be issued to a module:

Init: When handling this command, the module must initialize communication with all sub-systems, and ensure they are in a safe mode.

Start: Issued to start active control, and to initiate movement. Several types of movement exist: keep (maintain current position in active control), sweep (change one parameter), step (go to a defined position), track (follow an external variable) and jog (change one parameter a bit). Together with the start command, all parameters necessary to define the movement are sent.

Stop: Issued when movement (changes) must stop. During this command, all necessary action is taken to stop the current movement and place the sub-systems in a safe mode.

Check permissive: command to explicitly check the run permissive.

Suspend: Abruptly stop changes, but allow resumption of the current movement.

Resume: Resume the movement that was suspended earlier.

Exit: Terminate the process.

A module can support other commands as well, for example to set the current position as the Home position, set the rate of change, etc.

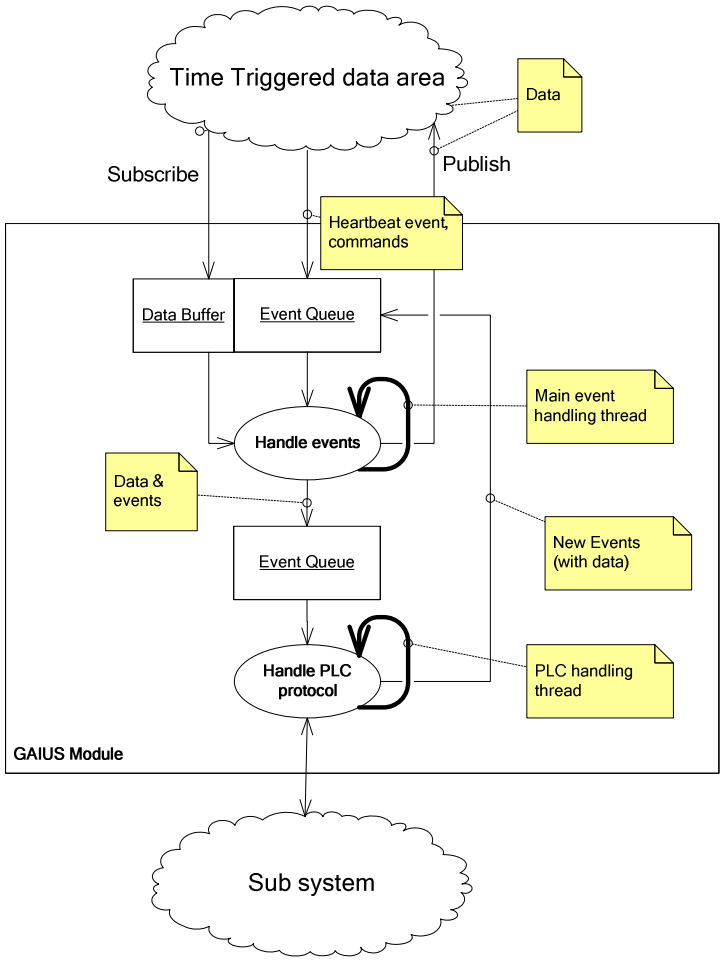
A module also received the Heartbeat event from the GAIUS bus. This event is used as the time base in the object.

Commands and the heartbeat event are a-synchronous. To prevent race conditions, commands and the heartbeat events are queued and then handled in a common thread.

4.2.3 Interface to sub-systems

The purpose of the modules is to control sub-systems. The communication with the GAIUS Bus must never be locked because of the communication with the sub-system, thus separate threads must be used to implement the GAIUS bus interface and the sub-system interface.

To make a clean division and prevent race conditions, communication between threads is through queues, as shown below. Through this set-up, a module can publish all relevant local data (PLC statuses, readings etc) on the GAIUS bus.



GAIUS Module structure

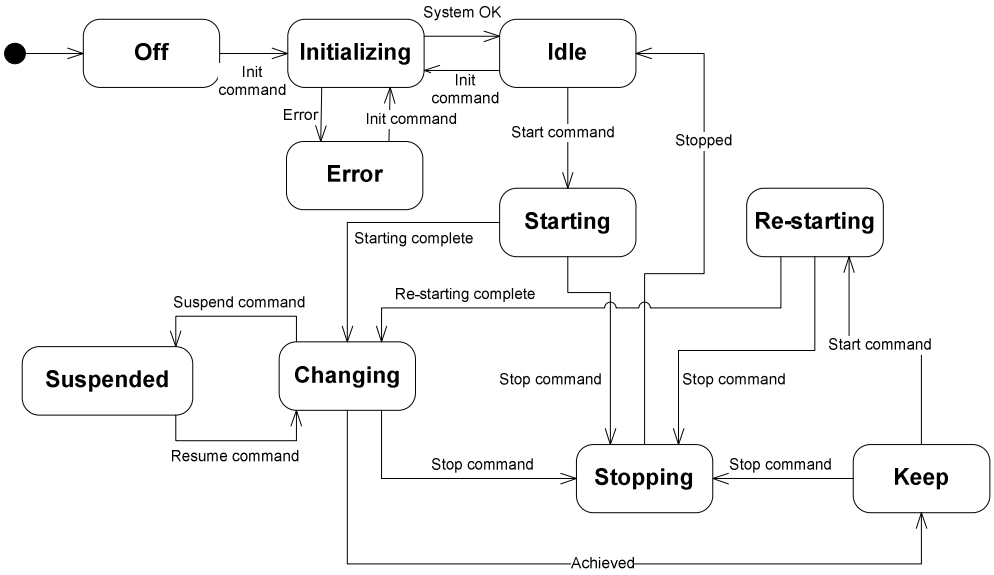
4.2.4 Internal behavior

It was shown that several commands are handled by a GAIUS module. These commands modify the internal state of the module, the 'Task State'. Not every command is valid in every task state. The state diagram for a standard module is shown below.

Please note that to improve the clarity of the figure, not all transitions to the Error state have been drawn. In reality, the Error state can be reached from almost all other states. The only way to get out of the Error state is through the Init command. Also, the transitions due to the 'Exit' command have not been shown. This command can be received in every Task State, and will lead to the abrupt termination of the module.

Several transitions in the figure occur due to commands, e.g. from Off to Initializing, from Idle to Starting, from Error to Idle, and from many states to Stopping.

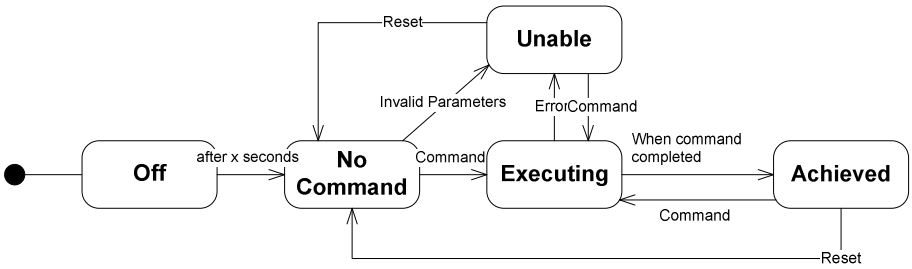
Other transitions occur due to internal events. These usually are events from the PLC worker thread. Examples are the transition from Initializing to Idle and from Starting to Changing. In these cases, a temporary state was entered due to a command, which is left if the sub system is brought in the right mode.



Task State Diagram

A module has a second state associated with command handling: the Command State. Because commands are a-synchronous events, multiple commands can arrive at the same time. The Command State helps the module decide which commands to execute.

In most cases, a command is queued in the internal event queue shown in the previous section. There is thus a (possibly substantial) delay between command reception and command execution. During that delay, the module should reject further commands. This is done by switching to the Executing state upon receiving a command, and going out of Executing mode when the command has been handled.

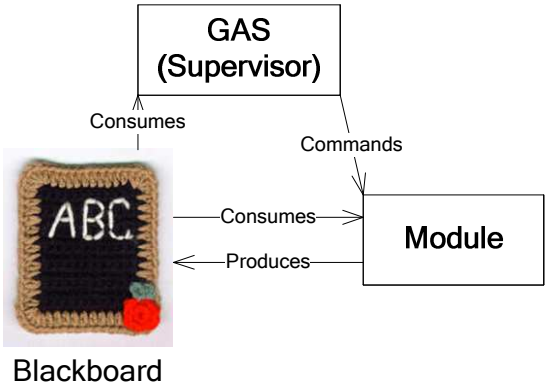


Command State Diagram

4.3 GAIUS bus

4.3.1 GAIUS as an implementation of the Blackboard pattern

The GAIUS system incorporates some ideas of the Blackboard pattern.



Blackboard Architecture Pattern

As with any blackboard, the method in which components access the blackboard is crucial. In the GAIUS system, the blackboard is accessed through the GAIUS bus. In fact, there is no single 'blackboard' entity, it will be shown below that the blackboard is distributed, and is represented by the data storage in the time-triggered data area.

4.3.2 Requirements for the GAIUS bus

The GAIUS system must have an robust communication layer. Without a robust communication layer, a number of problems can arise in a distributed system like GAIUS. Dead-lock can occur due to modules waiting for data items produced by others, who are themselves waiting etc. It may also be difficult for modules to locate the data they need, especially when the network topology is changed. Some of these problems can be solved by installing a centralized 'proxy' object through which all communication flows, but this suffers from a single point of failure and limited scalability. Also, as in a proper blackboard pattern, it is not known at design time which produces communicate with which consumers. GAIUS must allow flexible use of information by modules.

To prevent these problems, a number of requirements were defined for the GAIUS bus:

- There must be no single point of failure.
- There must be no required sequence of start-up for sub systems. This means that the protocol must allow for subscribers to start before publishers do.
- The information that is produced can be used by multiple processes.
- It must be possible to re-deploy the system without re-configuring the communications.

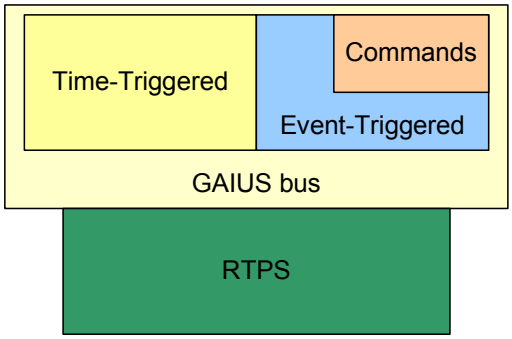
These requirements are satisfied by basing the communication on a publish-subscriber protocol like RTPS. As RTPS also satisfies architecture principle 3 by being an industry standard, it was selected.

4.3.3 Semantics of the GAIUS bus

In a control system like GAIUS, at least three different semantics for communication are required:

- For real-time control tasks, it is required to have timely communication. Values and states are produces at regular intervals (Time Triggered). Often, the flow of information in the system is complex, with many loops and branches. Care must be taken that dead-lock and live-lock are avoided, and that the information that is used is always consistent.
- Usually, there is a top-down hierarchy, where higher-level processes command the lower-level processes to change their mode of operation. This communication must be highly reliable, and the higher-level processes need confirmation whether the command was successful or not.
- An event channel is useful for sporadic events that have a short duration. Also, events are useful for reporting error conditions.

To meet these varying requirements, a layer is constructed on top of RTPS that supports these three semantics. It consists of the Time-Triggered area, the Command Area, and the Event-Triggered Area. This is the GAIUS bus.



GAIUS bus layers

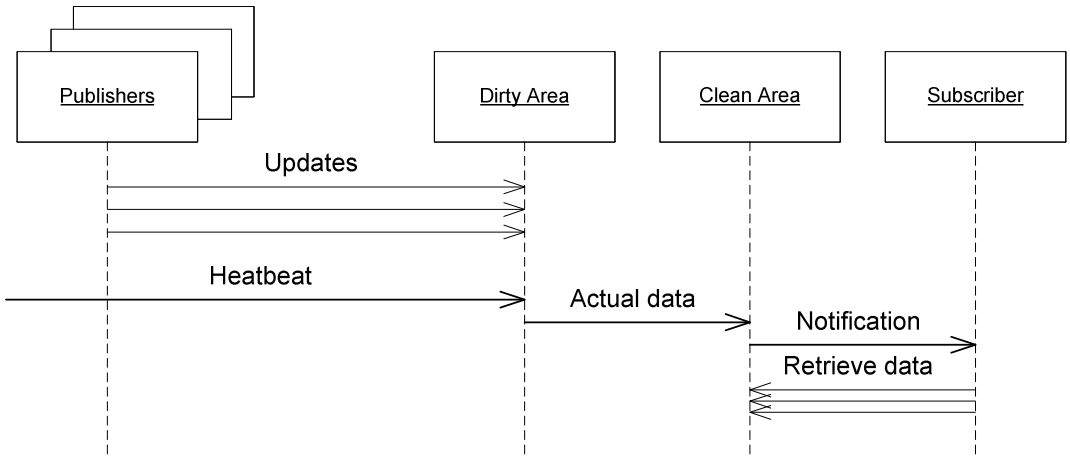
As a correct understanding of the semantics used for communication between the system components is essential for the correct implementation of these sub-systems, the semantics of the three methods of communication will be described below.

4.3.4 GAIUS Time-Triggered data

The Time-Triggered area is intended for information that is regularly produced, and combined with data from other sources to produce new information. Each subscriber receives regular updates from the producers. However,

to keep the information consistent, the subscriber is not notified of changes immediately. At a defined moment in time, the changes are made actual, and the subscriber is then notified.

To synchronize different subscribers, the changes are made actual at the same moment in time. This is done by a heartbeat message that is broadcast through the whole system.



Update sequence of Time Triggered Data Area

It is possible for a subscriber to be instantiated before a publisher. To prevent deadlock, it was decided to let the subscriber specify a default value, which will keep the subscriber in a safe condition, instead of generating an error if the publication is not yet available. Where necessary, a module can explicitly check the age of a publication to check for error conditions.

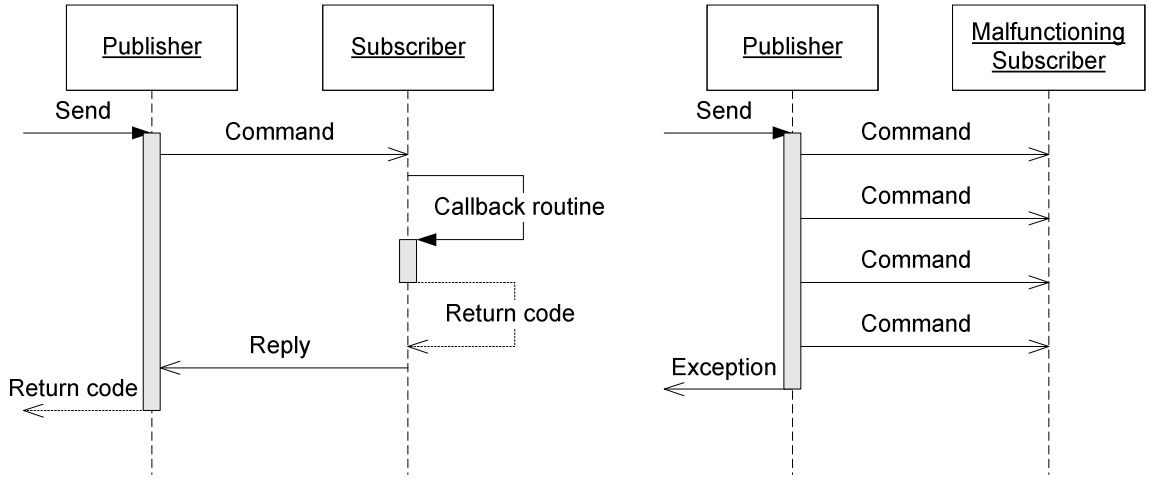
The clean data area can be seen as a local copy of the ‘blackboard’.

4.3.5 GAIUS Commands

The GAIUS commands are what the supervisor in the Blackboard pattern uses to tell the other modules what to do. Either GAS or the operator can be the supervisor. It is essential that the communication between supervisor and module is reliable.

When a command is issued, a callback routine is called in the subscriber. When the callback is finished, a reply is sent to the publisher of the command, as shown below. The GAIUS command mechanism contains a retry mechanism in case the reply is not received, making command transmission reliable.

The semantics of commands allow many publishers of a command, but only one subscriber.



Command sent successfully **Command send failed**

A GAIUS command can contain parameters, as well as details used for authorization.

4.3.6 GAIUS Event Triggered data

Included in the GAIUS bus is a thin layer over the RTPS API that utilizes an object-oriented syntax. Also, it is implemented in such a way that it avoids race conditions in the receiver.

4.3.7 GAIUS notification

The GAIUS bus has a dedicated Event-Triggered data channel for notification: ‘notify’. Any process can publish to this channel, and also subscribe to this channel.

Notifications contain data, describing the sender of the notification, a code for the notification, a description of the notification and optional extra data.

4.3.8 Data formats

Data in the GAIUS bus is communicated using a Variant type. This type supports most common data types (strings, integers, floats etc). Also it supports complex types: dictionaries and lists. In dictionaries, strings are used as keys. Both dictionaries and lists store Variant types, thus allowing complex data types to be constructed and communicated.

4.4 Configuration

All GAIUS modules need configuration, for example the directories to use for data storage, details needed to communicate with the PLC sub-systems, etc. etc.. The configuration is divided into three parts:

- **Deployment configuration**: is specific for each computer in the GAIUS system. This configuration is stored in the file deploy.conf; the location of this file is determined by the environment variable GAIUS_DEPLOY_DIR. It contains e.g. the specific GAIUS modules that need to be started on this computer, and the location of the other configuration parts.

- **Control configuration**: This contains configuration for modules that control the wind tunnel, e.g. Safety, TOPAC and Facility. This configuration reflects the configuration of the wind tunnel, and not the specific settings for a test. This is intended for data that changes independently from the experiments, as a result of wind tunnel maintenance.
- **Experiment configuration**: This contains configuration that determines the exact experiment being performed, e.g. the configuration of the measurement system. This information is read during the (re)initialization of the acquisition system. The current Acquisition State is used to determine which experiment configuration to read.

As all configurations are important for interpreting test results, all configurations are copied into the file structure storing the measurement data.

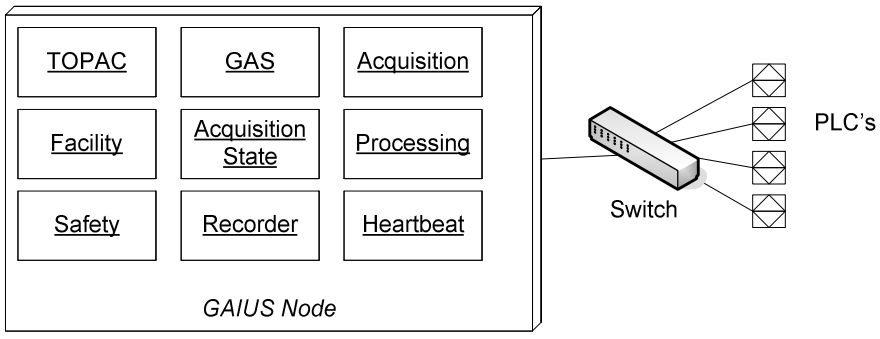
All configurations are stored in ASCII files using the INI-file format.

4.5 Deployment

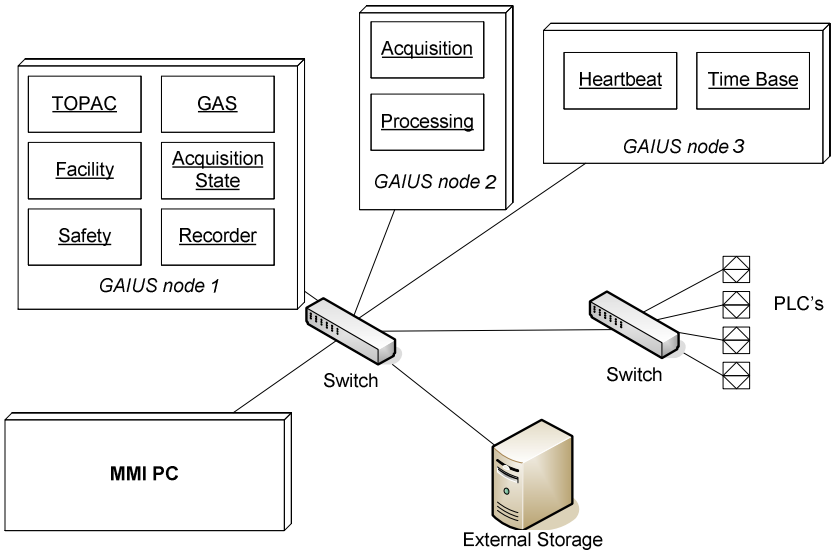
Deployment of the GAIUS system is quite straight-forward & flexible. Theoretically, the whole GAIUS system can be deployed on a single computer, which can have either a UNIX or Windows operating system; depending on the details of the specific wind tunnel that is being automated, other configurations can be chosen. Important aspects of the deployment are:

- TCP/IP connections between all GAIUS computers, and their routing.
- The test data file system, which is shared between all GAIUS computers. Often, the customer who ordered the tests provides external data storage for measurement results.
- The GAIUS installation, which is identical for all GAIUS computers sharing the same operating system.
- Every GAIUS computer must have a specific deploy.conf file, in a specific directory. The environment variables GAIUS_DEPLOY_DIR must be set correctly to point to this file.

Two example deployments are shown below. The minimal configuration can be used e.g. for testing or demonstration, whereas the typical configuration is advised for a typical wind tunnel.



Minimal GAIUS Configuration



Typical GAIUS Configuration

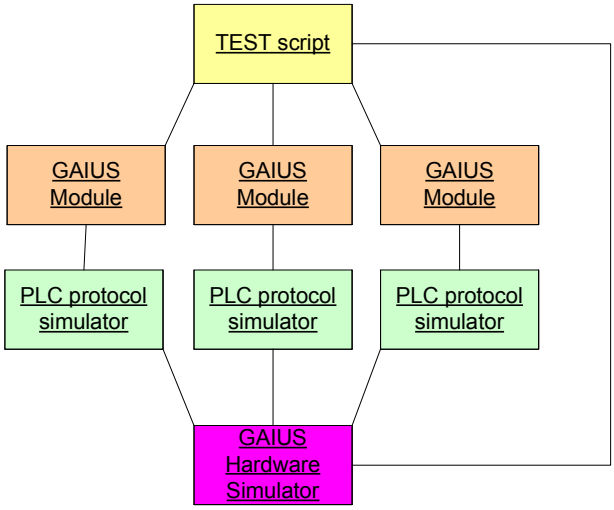
In the typical configuration, the modules Data Acquisition and Data Storage have been separated from other modules, because these two modules have a lot of interaction, and are processor intensive. To improve synchronization, the heartbeat generation has been deployed on a third node, which must have a light load to improve stability.

For systems with very high acquisition needs, it is possible to use several acquisition processes that can be deployed to different computers.

4.6 Testing & Simulation

Testing GAIUS on a simulator is one of the architecture principles of GAIUS. Because GAIUS usually communicates with its underlying systems using TCP/IP, it is possible to write modules that implement the same protocol as the actual hardware, and that are accessible to test scripts. Also, because all GAIUS modules use the GAIUS bus, it is easy to write scripts that control the modules. Thus it is possible to perform exhaustive black-box tests of individual GAIUS modules using test scripts. Also, the whole GAIUS system can be tested in this way.

To prevent interference between simulation tests and the GAIUS system controlling the wind tunnel, different TCP/IP ports are used when the system is running under test. This is controlled by the environment variable GAIUS_PORT_OFFSET.



GAIUS simulation set-up

The Hardware Simulator contains dynamical simulation of all sub-systems that are controlled by GAIUS, such that basic behavior is similar to reality. If necessary, highly advanced simulations can be included, but as the simulation is mostly used for functional tests of the system, only the functional behavior of the wind tunnel hardware needs to be included.

