ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA184

# INSPECTOR, A ZERO CODE IDE FOR CONTROL SYSTEMS USER INTERFACE DEVELOPMENT

V. Costa, B. Lefort
CERN, European Organization for Nuclear Research, Geneva, Switzerland

*Abstract*

Developing operational User Interfaces (UI) can be challenging, especially during machine upgrade or commissioning where many changes can suddenly be required. An agile Integrated Development Environment (IDE) with enhanced refactoring capabilities can ease the development process.

Inspector is an intuitive UI oriented IDE allowing for development of control interfaces and data processing. It features a state of the art visual interface composer fitted with an ample set of graphical components offering rich customization. It also integrates a scripting environment for soft real time data processing and UI scripting for complex interfaces.

Furthermore, Inspector supports many data sources. Alongside the short application development time, it means Inspector can be used in early stages of device engineering or it can be used on top of a full control system stack to create elaborate high level control UIs.

Inspector is now a mission critical tool at CERN providing agile features for creating and maintaining control system interfaces. It is intensively used by experts, machine operators and performs seamlessly from small test benches to complex instruments such as LHC or LINAC4.

## INTRODUCTION

In a constantly evolving research environment, changes in software may be indispensable for the control of new machines. Software can become outdated with the introduction of new technologies or advancements that facilitate usage and provide improved features.

Complex systems in accelerators require intricate software that requires proper maintenance and updates. Furthermore, software in accelerators are mostly developed in the scope of a Team or Department for a specific apparatus using local knowhow. Developing this kind of software may require users to participate in the software development in conjunction with the developers. Users may lack the availability or skills to profoundly help the development of software, potentially leading to poorly developed applications or applications that do not fulfil the needs of stakeholders.

Therefore, and with the increasing number of unique software applications, the combined time and cost of maintaining and developing software is becoming too high. UI software requires large amounts of graphical user interface (GUI) related code, increasing the development cost and restricting the skills necessary to create adequate user control interfaces.

Inspector proposes a separation between the UI and the software technology, essentially allowing the creation of zero code applications in order to diminish the cost of developing and maintaining such applications.

## ZERO CODE CONCEPT

Following the *What You See Is What You Get* (WYSIWYG) approach, Inspector introduces a visual IDE for the development of UI control applications. It applies the concept of abstracting the application from the underlying technology. It requires no code in the creation of applications. This approach means that Inspector allows the creation of UI control applications simply by using visual tools. As such, users can perform changes to Inspector-created applications or even create full applications without the involvement of developers and without coding skills.

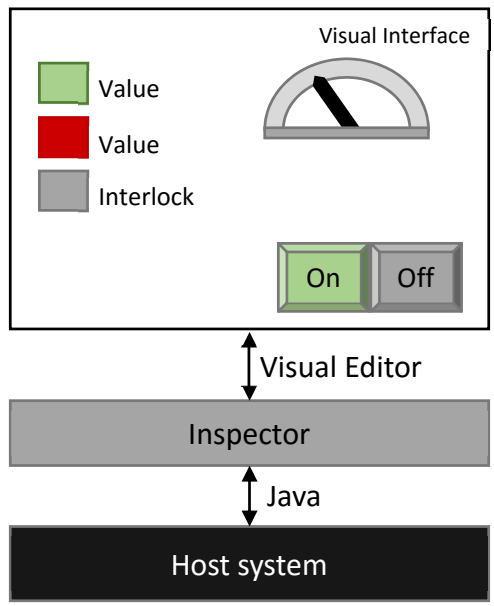


Figure 1: Integration between interface and system.

The creation of Inspector applications allows users to concentrate their efforts on the data presentation, rather than on implementation details. Inspector takes responsibility of making the applications portable, within the production environment. It also provides graphical tools for the creation of the user interfaces, along with accurate UI elements to present many types of data. As illustrated in Fig. 1, Inspector acts as an abstraction layer between the visual interfaces of control applications and the rest of the system, including any frameworks used.

Applications created with Inspector are independent of the base technology, meaning that any maintenance in regards to technology or frameworks is not handled by the user/developer but solely by Inspector. Any changes must

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA184



enforce backwards compatibility to avoid breaking changes for existing applications.

## INSPECTOR ARCHITECTURE AND IMPLEMENTATION

Inspector is a multi-tiered application developed in Java. Its architecture comprises a visual IDE client application and a multi-server backend. The architecture and implementation are portable and flexible to support the execution of the application in many different host systems in the CERN environment, which includes Virtual PCs, control machines and user workstations, with several operating systems. Due to unpredictable client application usage, the server architecture needs to be efficient, supporting many data and connections while being scalable for the dynamic loads presented by clients.

### *Visual Application Environment*

The Inspector visual editor is a Java application. It uses the Swing graphical toolkit [1] for the graphical core with numerous custom GUI elements built on top of it. The usage of the Java Virtual Machine (JVM) [2] ensures portability within the CERN host systems environment, where the application must execute on SLC6 [3], CentOS and Windows operating systems. Any system dependent code is delegated to servers, which execute in a controlled environment.

The application architecture comprises several interconnected modules, as shown in Fig. 2.

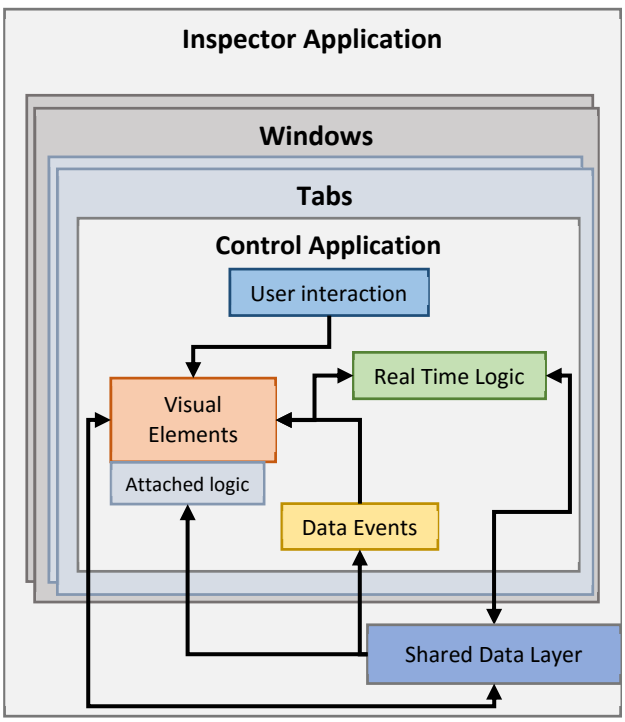


Figure 2: Inspector application architecture modules and data exchange routes.

To provide extensible customization capabilities, the Inspector application supports multiple windows, which internally can have multiple tabs. Each tab contains one control application that itself contains visual elements and logic.

Visual elements, named Monitors, handle user interaction and data display. These are custom GUI elements that interact loosely with other application systems, such as logic parts. In the case of data displays, they have simple supporting logic, named Attached Logic in Fig. 2. The attached logic perform simple data manipulations, such as determining if a value is inside a certain range or transforming values, e.g. converting a timestamp value into a date string.

Logic that is more complex can be included in separate modules executed within the application independently from the visual elements.

Data exchange inside the application is done via a Data Layer. Modules can access metadata and data in an abstract manner using the data layer. The data layer contains extensions that implement the data access and handle communication. It also exposes other features such as metadata discovery, simple data filtering along with data reading (get), writing (set) and subscriptions.

### *Load Balancing Backend Design*

The main data access for Inspector happens through the backend servers. The backend design connects the Inspector client application and the Data Access frameworks and devices. It comprises a Proxy server, Data servers, Synthetic Data servers and a Log server, as illustrated in Fig. 4.

The Proxy server is the backend entry point. It contains maps of all available data points (also named properties) and knows all active servers. The proxy server knowing all other servers performs load balancing. When a client requests a new property subscription, the proxy will delegate the subscription to the best available data server. The determination of the best data server depends on the number of active subscriptions on each data server, along with the CPU and memory load factor.

The Data servers are responsible to create and manage the properties and respective data access. They run on independent processes, and possibly in different hosts, to provide redundancy and scalability. The Data servers also act as a centralized data access, since they serve as a bridge between the client applications and the device access.

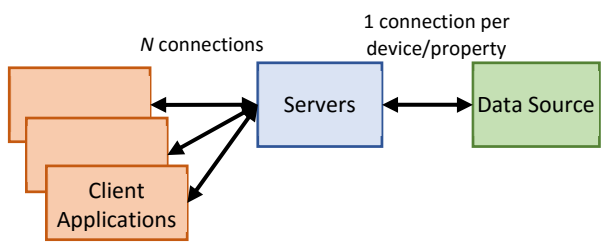


Figure 3: Centralized data access between many clients.

The centralized data access, illustrated in Fig. 3, guarantees that for each unique property access there is only one connection to the data source. Given this, the connection effort and data transmission overhead are moved from the data source into Inspector Data servers.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA184

This centralization reduces the load on the data sources themselves when many clients are connected.

The server backend contains also a distinct type of Data servers, named Synthetic Data servers. The Synthetic server executes in a similar manner to the Data servers but allows for data transformation. These servers have a custom equation interpreter that performs simple data transformation, changing the data before it reaches the clients. The interpreter can combine multiple data or apply mathematical functions; for example, it can run an FFT [4] on a data set. These operations, which may require high processing power, execute in the server reducing computational load on user host machines. This server also applies centralization, i.e. if many users are observing a certain data transformation then only a single execution is performed, instead of one per user.

Data logging is also possible in the Inspector Server backend. The backend contains a specialized Log server that logs data generated by Data and Synthetic servers. The data is logged in circular buffers stored in raw files. Furthermore, the Data and Synthetic servers also perform automated logging. Every data point generated by the system remains in a circular buffer up to 1000 single values. Clients can access the logged data transparently from Inspector visual elements, for example: for time based single values, the Chart Monitor automatically gets these last data points if available.

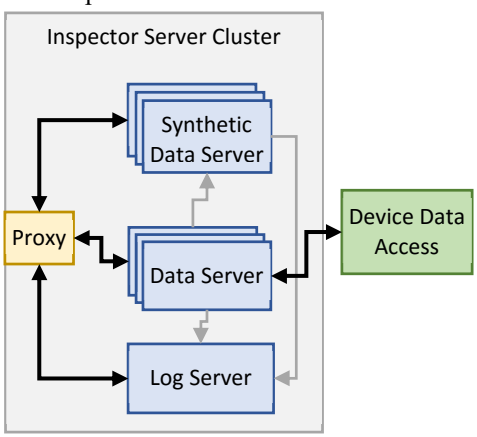


Figure 4: Inspector Server architecture.

Establishment of connections is done via the Proxy server, illustrated in Fig. 4. After the discovery and connection, communication between servers happens directly. Semi asynchronous RMI [5] is the communication method used to exchange data between all servers.

### *General Architecture Structure and Integration in the Control System*

Inspector client applications and servers integrate through the Data layer extension in the client application and the Proxy server in the backend structure, illustrated in Fig. 5. The communication, just like between different servers, uses RMI. The data exchange happens with direct RMI communication and an observer pattern [6] for property subscription. Invocations in RMI interfaces during data publishing are strictly asynchronous to avoid blocking if clients are slow to respond.

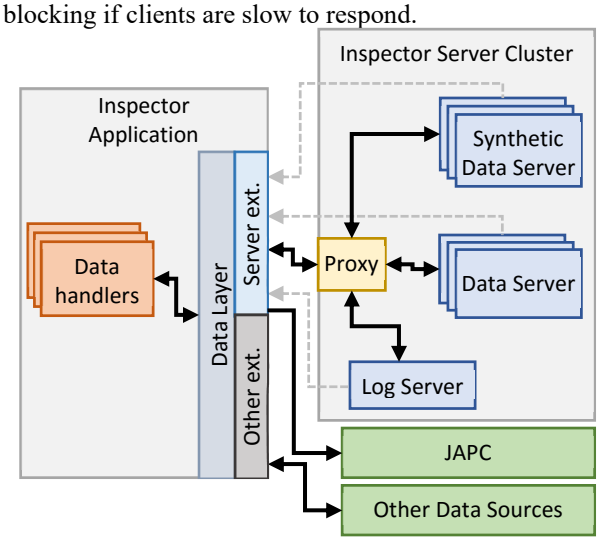


Figure 5: Integration of Inspector client application and servers.

Communication from the client to devices (namely sets or writes) happens directly from the client applications with no server interaction. The Data layer, in the client application, performs sets directly through JAPC [7], in the case of the default data sources. Any other data layer extension must implement a similar methodology. This behaviour along with the combination of a Role Based Access Control (RBAC) [8] ensures that only authorized users perform property sets.

## INSPECTOR AS AN APPLICATION DEVELOPMENT TOOL

Inspector includes several features to simplify the development of control applications.

### *Rich Visual Editor*

Inspector's visual editor application is a pure WYSIWYG with a simple interface and interactions. The interface, in Fig. 6, although based on Swing, is entirely comprised of custom visual components that follow a custom UI design scheme.

The editor includes a visual browser for data discovery and a working set creator. Users can build working sets to group and manage data sources for each project.

Using properties is as simple as dragging them from the data source list and dropping them inside the editor main panel. Once a property is dropped inside the editor, a visual menu appears with all the available monitors in which it is possible to display the property value. Inspector internally obtains the property metadata, i.e. its value type and other information, such as read/write access, and only displays adequate monitors for each property.

Tailoring a project is as simple as organizing monitors inside layout elements, such as popups. Monitors can be moved via drag actions, resized or customized by changing their attributes. Simple data transformation can also be done at Monitor level. Attributes of monitors can be altered

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA184



based on the property values, using ranges for example, or the data itself can be transformed by using formatters.

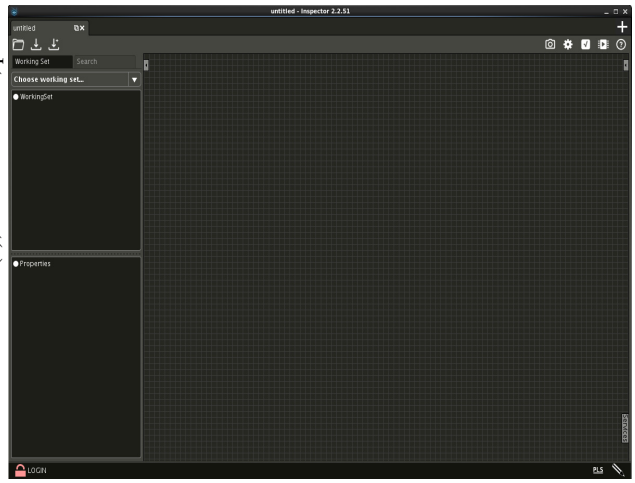

Figure 6: Inspector main application window.

The vast assortment of monitors, including polar plots, multi-state switches, buttons, dials, etc. can be organized via automated layout options. Visual elements can be organized in a grid system and layout constraints can be applied to multiple elements with single actions.

### *Soft Real Time Data Processing*

Monitors allow for simple data transformation, but many control applications require logic that is more complex. Inspector introduces two mechanisms to perform intricate data transformation and closed loop control.

A visual equation editor is included in the application. This equation editor facilitates the creation of mathematical equations that are executed in the Synthetic servers. Equations may receive inputs from properties and can perform mathematical operations or apply functions to values. The output of an equation is always one data value, single or complex. The computation of a synthetic equation happens based on a timer or an event.

For more intricate logic and data processing, Inspector includes a specialized module: Blueprints.

Blueprint is Inspector's powerful data processing and scripting system. It combines visual scripting, for users with limited programming knowledge, and high-level integrated language scripting, including Python support.

Blueprints execute in the client Inspector application as managed sub-applications. They are isolated from the core Inspector system, but can interface with visual elements and properties.

### *Agile Development*

Applications developed in Inspector can be easily changed after the development. In fact, due to the simplicity of Inspector visual development, applications are mostly developed iteratively, with the users making changes as they use the applications. Inspector also includes support for version control. It integrates with SVN [9] without use of external tools.

## INSPECTOR CREATED APPLICATIONS

Applications created in Inspector run in the Inspector client application. The Inspector application has two modes: *Edit* and *View*. In *Edit*, the default mode, Inspector makes available editing tools and data browser. In *View* mode, Inspector starts in an optimized mode without editing functionalities. Developed applications exhibit the same behaviour in both modes.

Inspector applications are used throughout CERN by experts and machine operators. They control operational equipment in accelerators such as LHC [10], LINAC4 [11] and SPS [12]. An example of such application is presented in Fig. 7, showing a control application for the LHC Radio Frequency (RF) cavities.

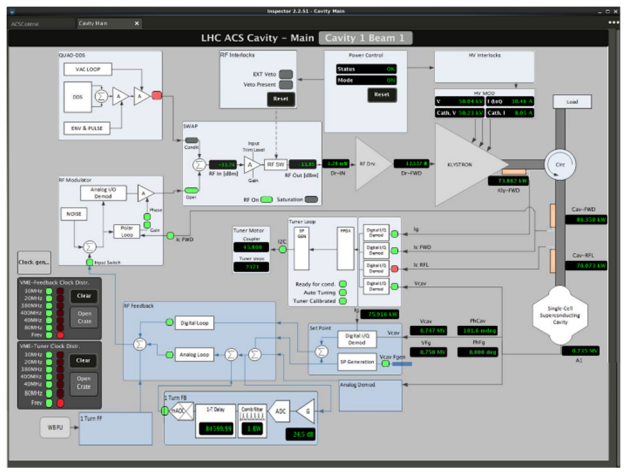


Figure 7: LHC RF Cavity control application developed in Inspector.

Other uses of Inspector include diagnostic of devices, via rapid application prototypes, and control of laboratory test devices.

## CONCLUSION

Inspector is a valuable application used throughout CERN. Many users from several teams and engineering areas depend on Inspector or Inspector developed applications.

Deployed in CERN's complex control system, Inspector has been a steadily growing application that many users are still discovering. Its main stakeholder is the RF and OP Groups in the Beams Department at CERN, where Inspector is one of the main control applications.

The technologies used in Inspector have proved adequate for the project. Several issues arise occasionally, due to the vastly diverse host environment conditions and growing user base. These issues lead to the introduction of alternative implementation methods, such as the asynchronous RMI used for server-client communications or an in-application stability tracking system.

Overall Inspector has been a reliable system with a steady user base and an important role at CERN.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA184

## REFERENCES


[1] Getting Started with Swing, https://docs.oracle.com/javase/tutorial/uiswing/start/index.html

[2] The Java Virtual Machine Specification, http://docs.oracle.com/javase/specs/jvms/se7/html/index.html

[3] Scientific Linux CERN 6 (SLC6), Linux @ CERN, http://linux.web.cern.ch/linux/scientific6/

[4] D. N. Rockmore, "The FFT: an algorithm the whole family can use", Computing in Science & Engineering, vol 2, Jan/Feb 2000.

[5] Java Remote Method Invocation, Oracle, http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Observer", in Design Patterns: Elements of Reusable Object-Oriented Software, 1994, pp. 293-303.

[7] V. Baggiolini *et al.*, "JAPC - the Java API for Parameter Control", in *Proc. ICALEPCS'05*, Geneva, Switzerland, October 2005.

[8] S. Gysin, A.D. Petrov, P. Charrue *et al.*, "Role-Based Access Control for The Accelerator Control System At CERN", *in Proc. ICALEPCS'07*, Knoxville, Tennessee, USA, 2007, paper TPPA04, pp.90-92.

[9] Apache Subversion, https://subversion.apache.org/

[10] The Large Hadron Collider, https://home.cern/topics/large-hadron-collider

[11] Linear accelerator 4, https://home.cern/about/accelerators/linear-accelerator-4

[12] The Super Proton Synchrotron, https://home.cern/about/accelerators/super-proton-synchrotron