Chapter 10: Experiment 6 – Instrumentation Platform

This chapter introduces and elaborates the architectural abstractions that were utilized in the experimental system titled Instrumentation Platform [124-125, 131]. While there exist a significant number of architectural elements and design patterns in this system’s architecture, the focus of this discussion is on those that are potentially useful as Plug and Play constructs. The purpose of this platform is to non-intrusively monitor and collect information pertaining to software systems in a network. To facilitate this, the platform provides the capability to string together a collection of soft-sensors that collect information pertaining to different software systems and different software technologies based on a user defined configuration. Accordingly, the platform adopts an extensible architecture in which soft-sensors are developed independently and then plugged into the platform as defined by the instrumentation needs. The reasons for adopting an extensible architecture include the need to extend the soft-sensors, interact with a wide variety of software systems, gather a wide variety of information, and the complexity of the software technologies to gather information from.

In the context of Plug and Play, the objective of this experimental system is to “transition towards high performant scalable system”, wherein, extensible soft-sensors can be plugged and unplugged based on the user defined configuration and can consistently deliver high performance and longevity (24 x 7 unattended performance). This is achieved by standardizing the plug and play interface, configuration driven consumption of extensible component services and lossless resource management. Accordingly, the hypothesis for this experiment is: “If the plug and play mechanism is simplified, high performance scalable systems are possible”. The core functionality of the platform are: a) support for a wide variety of soft-sensors, b) support for a wide variety of software systems, c) support a wide variety of software technologies and d) support a wide variety of soft-sensor configurations. The methodology adopted to validate the hypothesis is to treat each of the soft-sensor as a pluggable component which gathers data based on a user-defined configuration. Accordingly, the goal of this experimental system is to “enable Plug and Play of soft-sensors based on a user defined configuration”.

This chapter is organized as follows: a) The section on the problem space provides an overview of the instrumentation problem, b) The section on the solution space provides a possible solution that is Plug and Play compliant, c) The section on the requirements lists down the stakeholder concerns, d) The section on the quality characteristics lists down the functional and non-functional characteristics of the system, e) The section on decomposition brings forth the underlying component framework and supporting run-time, f) The next 4 sections provide an overview of the various components and their
composition, and g) The last section discusses about the Plug and Play architectural abstractions of this experiment. It uses the architecture description template presented in Annex A to express the architecture description of the instrumentation platform.

10.1 The Problem Space

Globally, the prevalence and proliferation of software systems has been increasing exponentially and this is evident in a variety of industrial sectors like retail, banking, transport, telecom, manufacturing and so on. Organizations who offer products in these sectors develop software systems that are increasingly becoming larger, complex and sophisticated. Every new release of existing software is growing in size, functionality and complexity [124-125, 131]. Emerging technologies, methodologies and approaches are increasingly being used by organizations to provide software solutions that are useful, usable, desirable, efficient, effective and different. However, the ability to deliver quality software is a big concern in the software industry. It is a known fact in the industry that many software projects result in schedule slippages, poor quality and cost overruns in spite of the tremendous effort impended by project managers and program managers. Even though programmers put in lot more effort than that is budgeted for to develop a software solution, the situation has not changed. One of the issues faced by organizations is the lack of real-time data with regard to the software process. Another issue is the lack of real-time software performance data. Often, the key performance indicators depend upon role playing people to provide the necessary information or on the events that are triggered when certain milestones are achieved. The granularity of the available data does not allow for much analysis either. There is not much audit trail available to perform a forensic analysis either. As a result, organizations find it difficult to ensure quality of software as well as deliver a software within the agreed upon timeframes.

10.2 The Solution Space

Traditionally, instrumentation is an effective solution in improving data quality. By examining the data gathered by the various instruments, it is possible to determine the cause for failure and a causal analysis of the context in which the failure happened can lead to successful recovery [124-125, 131]. By analysing the gathered data over a time-scale, it is possible to identify recurring problems. By analysis and simulation of software development activities based on the gathered data, it is possible to identify and solve the associated delivery quality issues as well [124-125, 131]. By analysis and simulation of usage data, it is possible to identify user behaviour, software developer assumptions and the gap between these two. Soft sensors are used in industries like chemical plants, refineries, power plants, and so on to provide
inputs for real-time diagnosis, prediction, control, implementing governance policies, management control and so on. They provide a viable alternative to manual and hardware sensory networks and provide useful insights which can be used to execute reliable processes. In this instrumentation platform, soft sensors are considered as a viable technology for the measurement of any management variable or process parameter or product parameter that are encountered both during the software development life-cycle as well as during the software deployment and usage life-cycle. The objective is to enable collection of software process/product metrics at high sampling rates and lower financial burden [125].

Accordingly, the goals of the instrumentation platform are:

- Create an instrumentation platform in which multiple soft-sensors can be plugged in and can be instantiated in order to capture information pertaining to software processes and various software applications.
- Support aggregation of the real-time information collected from these soft-sensors, providing ways and means of analysing the streaming data and drawing meaningful inferences out of it.
- Support the development of an integrated view of the software process and the associated software based on the different sensory information.
- Support the different soft-sensors and also the internal consistency relationships between these soft-sensors that arise out of using these instrumentation techniques.
- Support the measurement and improvement of software process quality and software quality.

### 10.3 Requirements for the Instrumentation Platform

i) Monitor software processes, their execution, outcomes, and performance indicators in real-time (Activity monitoring, outcome monitoring, resource monitoring).

ii) Mine data from different IT systems in real-time (Collecting information from different platforms, computing systems, software systems, technologies).

iii) Draw meaningful inferences from the gathered data in real-time (Correlating information to process execution and identifying metrics, activities, role players, constraints).

iv) Analyse streaming data gathered from multiple sources in real-time (Collecting information pertaining to different role-players and collating them in a single location).

v) Prepare and transform the collected data in real-time (Collecting data pertaining to primary measures and applying functions to derive secondary measures).
vi) Provide the ability to subscribe to desired data in real-time to different subscribers (Managers might like some information, administrators would like some others and so on).

vii) Capture events in the End user’s computing device.

viii) Configure system resources for which events needs to be captured and events that needs to be listened.

ix) Configure event details and supplemental event details that needs to be recorded.

x) Expose event logging API so that application specific events can also be captured.

xi) Manipulate and interact with operating system and other applications to facilitate event capturing.

xii) Provide the ability to consolidate information collected by soft-sensors in real-time.

xiii) Improve efficiency of organizations and the accuracy of information maintained by the organizations thereby reducing business latency.

xiv) Establish interfaces to all IT systems in an organization with the ability to read, write and manipulate information.

xv) Accommodate change in IT systems, while capturing information from it, which arises due to change in business processes or underlying technology or new layouts.

xvi) Collect as much relevant data as necessary for obtaining the necessary insight for making fact based decisions.

xvii) Support a wide variety of interfaces for wide variety of technologies and IT system architectures that keep evolving all the time.

xviii) Handle micro-level information flows between software applications, supporting processes, and supporting computing resources.

xix) Provide the ability to extend existing soft-sensors and to add new soft-sensors to the Instrumentation platform.

xx) Standardize and scale up instrumentation for deployment across all customers engagements.

xxi) Non-intrusive event capture and anonymizes event data

xxii) Time-stamp events and classify events based on pre-defined categories

xxiii) Every event is associated to an appropriate event handler

xxiv) Attach collection of event handlers to every process in the end user device

xxv) Plug-in extensions for categories of events to facilitate scalability of listeners.
10.4 Quality Characteristics of Instrumentation Platform

Quality characteristics of a system are a set of essential and distinguishing attributes that have a pragmatic interpretation of the system’s inferiority or superiority. By controlling the quality characteristics, it is possible to ensure that the system delivers desired value to its stakeholders [100-102] [132-133]. For this discussion purposes, the author utilized ISO 25010 [103] as the reference quality model. The subsequent sections list down the quality characteristics of the Instrumentation platform.

1. Functionality:
   a. Suitability: Soft-sensors capture atleast 80% of the software process data and software usage data
   b. Suitability: Atleast one soft-sensor can be used to capture event information
   c. Accurateness: The information captured by the soft-sensors accurately reflect the state of the software process or the software performance.
   d. Accurateness: Unattended, 24 x 7 hours event capture
   e. Interoperability: Has connectivity with all technologies that are deployed in an operating system.
   f. Compliance: Complies with all the constraints advocated by the operating system.
   g. Security: Restricts information mining to those that comes under the purview of the active user.

2. Reliability:
   a. Maturity: Soft-sensors are mature enough to help address 80% of the instrumentation needs.
   b. Fault Tolerance: Any issues that arise due to instrumentation is handled gracefully.
   c. Recoverability: On exceptions, uses rules and conditions to rollback or abort tasks (50%)

3. Usability:
   a. Understand ability: Standard and consistent interface for configuring the soft-sensors, for storing the collected information and for analysing the event data
   b. Learnability: Guides through the steps of using the platform (50% guidance)
   c. Operability: Uses familiar interfaces for interacting with the User (80% compatibility)

4. Efficiency:
   a. Time Behaviour: Always completes the sensing on or before time and within schedule (100%)
   b. Resource Behaviour: Consumes low computing resources (100%)
c. Resource Behaviour: Captures user interaction activities (80%)

5. Maintainability:
   a. Analyzability: Users can keep track of progress always (100%)
   b. Changeability: Users can change their soft-sensors on demand (100%)
   c. Stability: Absorbs 80% of perturbations introduced by users and their environments
   d. Testability: System performance and its use to sense data is reproducible (100%)

6. Portability:
   a. Adaptability: Adapts to change in the organization/working environment (100%)
   b. Installability: Organizations can scale up the level of sensing across its associates (90%)
   c. Conformance: Conforms to organization infrastructure policies (100% compliance)
   d. Replaceability: Users can replace any of their soft-sensors (100%)

10.5 Decomposition of Instrumentation Platform

Decomposition is a form of problem solving and is a hierarchical and incremental break-down of a problem into components that are easier to conceive, understand, specify and maintain in such a way that the software system can be reconstructed from those decomposed components by composition. The greatest value of decomposition is the insight it provides into the structure of the software system. By successive decomposition, a set of components and the logical/formal structure by which these components interact can be arrived at. Each component affects the behaviour of the software system as a whole and they coordinate and coexist together to manifest the properties of the software system. The act of decomposition starts with understanding the problem by a reductionist understanding, diagnosing it using patterns of components and their relationships and synthesizing an approximate representation.

In an industrial setup, in order to produce a product with high quality, process control is necessary. Instrumentation is the basis for process control [38-39]. Measurement is a primary tool for collecting information pertaining to execution of industrial processes. The collected information can be utilized to understand, manage, manipulate and improve the quality of the processes. In order to improve the reliability of the process, precise measurement is essential. Instrumentation aids in gathering the measures pertaining to an industrial process (specifically the different process variables) with sufficient accuracy by utilizing appropriate sensors and this information serves as the basis for controlling the way the processes are performed.
In the software industry, instrumentation has been utilized predominantly to trace, debug, profile, and log a software’s execution and is often limited by the execution coverage. Moreover, instrumentation has already been part of the program of a software system and causes dramatic impact in the system’s performance. The approach that is discussed in this chapter, proposes the use of soft sensors that resides outside the software system (it is not programmed as part of the software system) to gather meaningful data about process execution which is then utilized to monitor in real-time the process progress [124-125, 131]. As a value add, this data can be used to monitor, analyse and improve the productivity and efficiency of various role players in the process [124-125, 131].

![Diagram of Instrumentation Platform]

**FIGURE 10.1: DECOMPOSITION OF INSTRUMENTATION PLATFORM**

The proposed Instrumentation involves: a) Defining a set of key process indicators, b) Defining a set of basic and derived measures on these indicators, c) Defining a set of sensors that can detect and provide a measurable output, d) Defining a set of processes for instrumentation, e) Defining a set of transformations on the measures, f) Establishing man-machine interfaces, g) Networking a collection of sensors and the information collected by them, h) Monitoring process progress using these sensors, i) Ensuring consistency of quality of the collected data, and j) Deploying the instrumentation while
monitoring the gains over time. The objective is to apply instrumentation on software processes by identifying those man-machine interfaces which can be completely instrumented by using minimal organization resources and interventions.

Accordingly, the instrumentation platform can be decomposed into the top level components: a) Instrumentation framework that supports instrumentation and process control, b) Instrumentation framework run-time that supports the instrumentation framework and also enables networking of different sensors, c) Sensor managers which manage a collection of sensors and the measurements that they should identify, and d) Data managers which manages the information collected by these sensors. This decomposition is illustrated in Fig 10.1.

10.6 Instrumentation Framework

Instrumentation is the process by which the tasks that are performed on software systems are monitored and data corresponding to these tasks collected in order to analyse the process metrics and performance indicators [124-125, 131]. In the prevalent graphical user interface driven operating systems, events are the basis for communication and coordination across heterogeneous user interface elements and the underlying program logic. Whenever any action is performed on these systems, it triggers a plethora of events and additional event data which enables the underlying software system to take the next step in its program logic. The type and nature of events that are triggered depends on the tasks that are performed on these software systems. The instrumentation framework facilitates recording these software system events in a standard, centralized way which can then be examined, analysed and pattern matched to correlate with processes.

The instrumentation framework utilizes a collection of daemons pertaining to the different sensory systems (essentially sensor managers and associated soft sensors) to capture event information and associated event data from the underlying software systems [125]. The sensor manager and associated soft-sensors maintain a manager-worker composite system to work together and collect the event information. All the information pertaining to the different events across different categorizes are consolidated and passed on to the dispatcher which categorizes the events and passes them onto the respective handlers. The job of the handlers is to segregate the available event information, transform them as necessary and organize them into appropriate buckets so that it becomes easy to do further processing on the captured data. The Analyser operates in two modes: a) It aids real-time diagnostics by analysing streaming data in real-time, b) it aids forensic diagnostics by analysing off-line data. Depending
on the purpose of the diagnostics to be performed, either of this analysis mode is supported by the instrumentation framework.

The Data miner component collects data from different systems (platforms, logs, technologies, infrastructure, networks, devices and so on) pertaining to different processes in different process states and streamlines them. For this purpose, it utilizes the sensor manager and the sensor network. The data miner also operates across a network of computing systems to collect and collate data in real-time. It aids in monitoring software processes, their execution, outcomes, and performance indicators in real-time (Activity monitoring, outcome monitoring, resource monitoring and so on). The transformer component aids the instrumentation framework in drawing meaningful inferences from the gathered data in real-time (Correlating information to process execution and identifying metrics, activities, role players, constraints). It also prepares and transforms the collected data in real-time (Collecting data pertaining to primary measures and applying appropriate transformation functions in real-time). The decomposition of the instrumentation framework is illustrated in figure 10.2.

![Diagram of Instrumentation Framework](image)
10.7 Instrumentation Framework Run-Time

There are many soft-sensors that are used during instrumentation. All these sensors are nearly independent of each other and continuously evolve. While the soft-sensors provide the necessary sensing capabilities, the run-time provides the interfaces for plugging in these sensors and associated capabilities into the instrumentation framework. This sensor capability is utilized by the instrumentation framework by means of the instrumentation framework run-time which facilitates the dynamic late binding of sensors into the instrumentation framework. The run-time provides a standard mechanism for this purpose which comprises of: a) a locator service to uniquely identify a plugged in sensor, b) a discover service to uniquely identify the software systems that these sensors are applicable to, c) a creation service to facilitate the instantiation of the sensor, d) a bus service to obtain the streaming data from the sensors, e) a monitoring service to facilitate sensor invocation and operations, and f) an eventing service to facilitate additional event handling. In essence, the run-time controls the various soft-sensors in such a way that during execution it is difficult to separate them from native components.

![Diagram of Instrumentation Framework Runtime](image)

**FIGURE 10.3: SOFTWARE DECOMPOSITION OF INSTRUMENTATION FRAMEWORK RUNTIME**

A design pattern used in embedded systems to support sharing of information is the bus. The bus serves as the channel using which information can be transmitted in the form of a stream. This is a candidate design pattern in the instrumentation framework run-time [IFR] for supporting information...
sharing between the sensors. In computer architecture, control channel enables the processor to transmit control signals to the attached computing resources whenever specialized operations (for example: data transfer, device I/O) are required to be performed. This is a candidate design pattern that can be used by IFR when dealing with streaming data. Another design pattern that is used in USB devices for integrating parts is the host controller. The host controller facilitates interaction between the USB based devices and the host system. The host controller is a candidate pattern in the IFR for supporting interaction between the parts.

In the electrical industry, Plugs and sockets are design patterns that allow electrically operated devices to be connected to the power supply. While the plug is a movable connector attached to the device, the socket is fixed connector that is attached to the power supply. The plug and sockets are candidate design patterns that can be used in the IFR to attach soft-sensors to the instrumentation framework. The instrumentation socket provides the mechanism to configure the soft-sensors to gather the necessary events and event data from the underlying software system. In the Windows operating system, applications are event-driven. The various applications do not call a function to obtain the events (keyboard, mouse, display etc), but rather wait for events to be posted onto their respective message queues. This eventing capability is a candidate design pattern for the IFR in order to handle the various events that occur in the parts that are plugged into the instrumentation framework.

10.8 Soft Sensors

Soft sensors are utilized to collect and store data on a wide variety of process parameters from a plethora of software systems [125]. In order to improve the sampling frequency, performance and efficiency of these sensors, they are designed such that each soft-sensor handles only a small sub-set of the events generated by the software system according to a pre-defined categorization (GUI events, Network events, Browser events, Device events and so on). Each of these sensors adopt a specific approach to capture data, some utilizes the event driven approach, some uses the data driven approach, some others utilize the model driven approach and so on. The consideration of an approach is based on the ability to identify the data to be captured and the underlying technology paradigm that needs to be tapped for gathering this information. However, it is essential that all these sensors interact and interface with the instrumentation platform using one protocol (which in this experiment has been the event driven approach) so that it is easier to optimize the performance and efficiency of the platform as a whole. The first step in the design of the soft-sensors is the identification and analysis of the data that is emanating from a software system that is of relevance with regard to process performance. This would involve
identifying the influential variables and events carrying information about dynamics of the sensed system. The next step is to find exceptions, missing data and outliers and see ways and means of handling these conditions. The next step is to design corrective actions for incomplete data. Once the requisite data is identified and their data schema put in place, the next step is to find out the ways and means to filter all the noise (unnecessary data or the exceptions or the outliers). The last step is to encode the ways and means while taking care that the state of the tapped software system is not affected in any way by these sensors.

### 10.9 Soft Sensor Run-Time

As discussed in section 10.2.7, there are many soft-sensors that are used as part of the instrumentation platform. Each of these sensors are categorized and grouped under relevant category so that the collected information can be organized and structured. All these sensors are interdependent on each other as the information that they capture is about the process performed by the role player. While the soft-sensors provides the desired capability for collecting information from a wide variety of software systems and technology infrastructure, the soft-sensor run-time provides the necessary run-time library support, operating system interface support and the ability to plug the sensor into the sensor socket of the instrumentation framework. Once the soft-sensor is plugged to the instrumentation framework, the sensor run-time serves as an abstraction layer that extends the capabilities of the instrumentation framework by considering these sensors as constituent elements of the framework. When the sensor is no longer necessary, the run-time provides the facility to unplug the sensor from the instrumentation framework.

![Diagram of Soft Sensor Run-Time](image)

**Figure 10.4: Software Decomposition of Soft Sensor Run-Time**
The run-time utilizes the interface services offered by the Interface channel to interact with the underlying software system and technology infrastructure. It utilizes the plug and play services offered by the interface manager to dynamically attach and detach from the sensor socket. The interface manager treats the host system as the central controller and the target system as the system to be monitored. The run-time utilizes the data services provided by the data channel to obtain data from the various software systems that it interacts with and transmits it to the host system. The run-time utilizes the configuration information received from the control channel to dynamically reconfigure the events that are monitored by the soft sensor as well as the data that needs to be collected. The run-time utilizes the services provided by the event manager to monitor and handle the events that are triggered by the host system as well as the target system. This decomposition of the run-time is presented in figure 10.4.

**10.10 Discussions**

The experimental system addresses its primary goal of enabling Plug and Play of soft-sensors based on a user defined configuration by:

a) Encapsulating sensor functionality by encoding them into the soft-sensor component.
b) Interfacing with a wide variety of software technologies using corresponding soft-sensors.
c) Encoding the plug and play interface mechanism as sockets and compatible plugs.
d) Encapsulating the plug capability as part of the soft-sensor run-time component.
e) Encapsulating the socket capability as part of the component framework run-time component.
f) Listening for events and dispatching them to the component framework using the plug & socket interface.
g) Work concurrently to collect information pertaining to the underlying software systems and dispatch them to the component framework using the plug & socket interface.
h) Collecting event data based on the user defined configuration and dispatching them to the component framework using the plug & socket interface.

The critical elements events, services, user-defined configuration and event data that are required for proper functioning of the instrumentation platform have been made accessible through the plug and socket interface. Depending on the complexity of the underlying software system, events to be listened and the event data to be gathered, the sensors are designed at varying levels of complexity while conforming to the same interface. Individually, these sensors are designed for high performance, low resource consumption and low processor and memory footprint. Their objective is to listen for events...
and gather data as defined in the user-defined configuration. The sensors work concurrently, collecting data about the various events occurring on different software systems as and when they happen. They are independent of each other and collectively realize high performance, throughput and efficiency. Hence, it can be observed that “plug and play architectures are high performant, scalable and simple”. Accordingly, the hypothesis “If the plug and play mechanism is simplified, high performance scalable systems are possible” was verified. Further, the underlying principles and constraints that are discussed in the subsequent sections aid in reducing the complexity of the plugged components while delivering high performance.

10.10.1 Principle of simplicity

According to Maier [51], in order to reduce complexity, it is essential to simplify the subsystems by reducing them to the bare essentials. In the case of the instrumentation platform, the soft-sensors are simplified to the extent that they do nothing other than listen to a pre-defined set of events, collect data from the underlying software system when these events are triggered based on a user-defined configuration and pass this information to the component framework.

10.10.2 Principle of compatible support elements

According to Maier [51], the architecture of the supporting element should fit into the system that it supports. In the case of the instrumentation platform, the plug interface of the soft-sensors and the socket interface of the instrumentation framework are designed such that only compatible sensors can utilize this interface.

10.10.3 Principle of minimal communication

According to Maier [51], while choosing the sub-systems, it is necessary to define them in such a way that they are independent to each other and there is minimal communication between them. Accordingly, in the instrumentation platform, the soft-sensors listens to events occurring in the underlying software system, gathers event information from these systems and passes this information to the instrumentation framework. There is no interaction with any other component nor with any other sensor.

10.10.4 Principle of independent partitioning and low complexity

According to Maier [51], while partitioning a system, it is necessary to choose the elements so that they are as independent as possible; that is elements of low external complexity and high internal complexity. In the case of the instrumentation platform, the soft-sensors are designed to be independent
of each other and serve as extensible components. They are designed to listen for certain events occurring on an underlying software system and report these events. During sensors design, care is taken so that there is no overlap with regard to the events that are listened by the different soft-sensors. The way the sensors capture the event data is obfuscated from the component framework and its internal complexity well hidden.

10.10.5 **Principle of least privilege**

According to Whitman [123], every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job. In the case of the instrumentation platform, the different sensors listens to the events occurring on the software systems for which they have the privilege. In case sufficient privileges are not available then the soft-sensors do not listen to these events.

10.10.6 **Principle of economy**

According to Whitman [123], every subsystem’s design should be as simple and as small as possible. In the case of the instrumentation platform, the different sensors are designed for listening to a limited set of events and hence they are very small in size and highly optimized.

10.10.7 **Principle of localization**

According to Maier [51], it is desirable to utilize a configuration in which local activity is performed in high speed and global activity is performed in slow speed. In the case of the instrumentation platform, the different soft-sensors use system memory to capture the event information and dispatch them to the component framework while the component framework processes the event queue based on its ability to transfer the event data into persistent storage.

10.11 **The Plug and Play Architectural Abstractions**

This section introduces and elaborates the plug and play architectural abstractions of the instrumentation platform. While many of these abstractions were discussed in earlier experiments, their capabilities have improved at the end of this experiment, and are expressed as such. These abstractions are considered, based on the premise that “Software Plug and Play is a collection of standards, encoded in the run-time and embedded in the component schema, in order to support functional and structural extensibility of the instrumentation platform”. These abstractions are:
10.11.1 Plug

A plug comprises a collection of software resources that enables the plugged component to be executed on an underlying computing system. It serves as the gateway for the events and data that the plugged component manipulate as part of its computation and allows two way communication between the component framework and the plugged components. The plug encodes the interface mechanism for connecting to a compatible socket thereby enabling the plugged component to function as an external component of a component framework.

10.11.2 Socket

A socket is a handle that allows the plugged components to utilize the event and data channel exposed by the component framework. It dispatches the events and associated data that it receives from the plugged component to the component framework. As an interface, the socket performs a type checking for compatible plugs to validate the connection and disconnects those that fail the validation.

10.11.3 Component Framework

A component framework defines specific interaction and composition standards that should be adopted by the constituent components and provides the set of elements required for supporting the execution of these components. It implements a component schema which describes the characteristics of the constituent components in terms of the services, service operations, events, and other metadata that it should support. It configures, collects data and processes events from the plugged components.

10.11.4 Plugged Component

A plugged component extends and adds functionality to the component framework thereby enabling the evolution of the underlying system. It utilizes the run-time services provided by the plug to attach itself to the component framework. It processes various events that it receives as well as dispatches them along with the event data to the component framework. Its active configuration is determined by the component framework.

10.11.5 Event

An event is an encapsulated set of actions that are listened by the plugged components. Such events are triggered when a set of pre-defined execution conditions are met. Both the events and the
corresponding event data are passed onto the component framework by the plugged components for further processing.

10.11.6 Configuration

A configuration is an encapsulated set of preferences supported by a component on a set of resources that it provides. While the user defines the preferred configuration, the component framework percolates the new configuration information to all its constituent elements (including the plugged components).

10.11.7 The Plug and Play Component Model

The Plug and play component model is put together by considering the plug and play architectural abstractions that were discussed earlier. Accordingly, the constituent elements of The Plug and play component model are: a) The component framework which provides the composition semantics, the logical structure for facilitating components composition, underlying structure in terms of interfaces, resources and protocols for the composition of components playing specific roles, b) The component framework run-time which serves as a socket and provides the necessary functionality for supporting composition of components, communication between components, manipulation of components, management of components performance, invocation of functions and services supported by the components and transformation of data from one form to another, c) The Data store which hosts the information that is gathered, processed and transformed by the plugged components, and d) The components manager which manages the plugged components, the data that they collect and their respective run-times. Figure 10.5 provides a decomposition of this model.

While the plugged component hosts the services and functionalities necessary for the composed component, its run-time provides functionality and resources needed by the component, and also the ability to serve as a plug. It provides basic services necessary for the various components to work together, generation of events which are then passed onto the component framework for further action and so on. The interface services provided by the plug enables the socket to communicate and interact with the plugged component using the accepted interfaces. While the component framework hosts the functionalities and processes that are utilized by the end-users, its run-time provides functionality and resources needed by it, along with the ability to serve as a socket. It provides basic services necessary for various plugged components to work together, for raising and handling events, for invoking various
functions that are supported by the plugged component and for managing data flow between the plug and socket.

![Diagram of Plug and Play Component Model](Image)

**Figure 10.5: Decomposition of Plug and Play Component Model**

**10.12 Summary**

This chapter introduces and elaborates the architectural abstractions that were utilized in the experimental system titled instrumentation platform [124-125, 131]. The approach to transition towards high performant scalable system was illustrated by utilizing soft-sensors to capture events and event data from the underlying software infrastructure based on user-defined configuration. Accordingly, the hypothesis “If the plug and play mechanism is simplified, high performance scalable systems are possible” is verified. Understanding the key architectural principles of this experiment helped identify the underlying plug and play architectural abstractions. This was culled out and expressed in the form of an abstract Plug and Play component model.