Chapter 6: Experiment 2 – Systems Modelling Platform

This chapter introduces and elaborates the architectural abstractions that were utilized in the experimental system titled Systems Modelling Platform [105]. 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 support multi-modelling so as to create an integrated understanding of the system as a whole. Accordingly, the platform adopts an extensible architecture so that new approaches to modelling systems can be developed independently and then plugged into the platform on demand. The reasons for adopting an extensible architecture include catering to the complexity of the domain and provisioning for extensions.

In the context of Plug and Play, the objective of this experimental system is to “transition from an extensible system architecture into an integrated extensible system architecture”, wherein the plugged components interact with each other and maintain consistency relationships with regard to the information they manipulate. This is achieved by providing necessary meta-information manipulation and exchange services as part of the plugged components. Accordingly, the hypothesis for this experiment is: “If components that are dynamically attached and detached can maintain consistency relationships on the data they manipulate, then an integrated view of the data can be created”. The core functionality of the platform is the ability to support a multitude of modelling techniques and views which are implemented as corresponding modellers. The methodology adopted to validate the hypothesis is to treat each of these modellers as a pluggable component, with each of them providing a view of the model (based on the meta-model of the system), thereby maintaining the consistency relationships. Accordingly, the goal of this experimental system is to “enable Plug and Play of modellers while maintaining integrity of the models”.

This chapter is organized as follows: a) The section on the problem space provides an overview of the multi-modelling 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 systems modelling platform.
6.1 The Problem Space

Systems are multi-dimensional, complex and have multiple ideals [106-109]. One of the biggest challenges with systems, is the uncertainty on where do they begin and where do they end; what is inside and what is outside [106-109]. This is because what is perceived to be the system is an approximation of the real system [51][106-109]. It is possible to learn about the real system incrementally and improve the approximate system or system-in-focus; as the gap between the approximate and the real system is the source of the feedback and the basis for the incremental understanding. One iteration of understanding the real system could be identifying interesting properties, cognizing interesting insights based on these properties and creating models that capture this information [51][106-109]. In the world of systems, an iterative approach to incrementally obtain understanding involves successively spanning many dimensions of the system and adopting a holistic attitude with regard to it [51][106-109]. Traditionally, system thinkers adopt an array of modelling approaches (influence diagrams, system dynamics models, viable system models, living systems models and so on) to develop an understanding of the system.

The existing modelling tools are designed and developed by multiple vendors and support different modelling techniques, modelling languages and modelling notations. Most of the tools, support a limited set of views of system-of-interest, which are expressed as corresponding system models [106-109]. In many cases, it is not possible to add more models as that would break the consistency relationships that exist within the modelling tool. This situation results in ambiguity in understanding the system as a whole. System thinkers would like to create a few models that capture certain aspects of the system-of-interest and would like to see how these models expressed using different modelling notations. This has resulted in a situation where system thinkers are forced to operate in silos whenever they utilize any of the modelling tools. Additionally, there is no consistency in the way the models are defined or analysed or solved in the different modelling tools and they cannot be extended any further as they are bound by the techniques that they support.

6.2 The Solution Space

Systems modelling involves creating relations taking into account other relations. It is the way in which model elements are put together to create a representation of the system-of-interest and is designed top-down. It is semantically motivated and is understood through reductionism. A model comprises of a set of model elements and the logical/formal structure by which they are interconnected. It establishes the interdependence between the various model elements and is a description of how the model elements
compose structurally and operate dynamically. The fundamental characteristic of a model is that every time it is changed, say by addition or subtraction of a relationship or model element, then the properties of the whole model changes. Systems modelling involves understanding how model elements are organized formally, semantically and how they are represented and governed by modelling rules.

According to Maier [51], systems modelling is the creation of abstractions or representations of the system to predict and analyse performance, costs, schedule, and risks and to provide guidelines for systems research, development, design, manufacture and management. Modelling starts by first understanding the purpose and objectives of the system-of-interest. The next step is to identify the structure of the system and the behaviour that it exhibits in relation to the environment that it operates in. The structure of the system is understood by means of identifying the system elements; the influences these elements cause on the systems purpose and objectives; and the influences that they cause on each other. The behaviour of the system is understood by identifying the processes that aid in achieving the systems behaviour, the system elements that are involved in achieving this behaviour and the capabilities of the system that are utilized. Often the structure and behaviour are augmented with the system characteristics in order to have complete understanding of the system. The last step is to express the identified constructs using an appropriate modelling language and diagramming notation. No system can be modelled exactly and hence it is necessary for constructing a variety of system models of varying complexity and then arrive at an integrated understanding of the system [51][106-109]. In such situation, it is essential to ensure that these models are consistent and in alignment to each other.

Accordingly, the goals of the systems modelling platform are:

- Create a modelling platform in which multiple modelling languages can be supported by means of appropriate modellers.
- Support the development of an integrated systems model based on the different models that systems thinkers develop.
- Support modelling a wide spectrum of models ranging from form to function, wherein one end would be physical models and the other end would be mathematical/virtual models.
- Support the different modelling techniques and also the internal consistency relationships between the models that arise out of using these modelling techniques.
- Support the formulation of abstractions of systems, system elements, their inter-relationships and the system characteristics and their appropriate representation.
For facilitating this, a platform that facilitates modelling is proposed. The objective of this platform is to support formulating and resolving the model design problem. Defining the modelling problem involves identifying the purpose of the system being modelled, identifying the desired properties of the system, identifying the modelling elements that comprise the system, identifying archetypes to be used and planning and preparing the design methodology to be adopted. Resolving the modelling problem involves identifying the modelling languages, identifying the levels of discourse, segregating the different levels of concerns, adapting a formal method for change management, creating spatial structures that represent these models, representing these structures as models within the defined constraints.

### 6.3 Requirements for the Systems Modelling Platform

i) Systems thinkers should be able to select a specific modelling technique that is based on a modelling language and express their models using this modelling technique.

ii) Systems thinkers should be able to take information from one model and use it in another.

iii) Systems thinkers should be able to use a collection of models to express their system.

iv) Systems thinkers should be able to work on multiple system models at the same time.

v) Systems thinkers should be able to utilize multiple modelling techniques to express a single system.

vi) Various systems modelling techniques like Viable System models, Living System models, Business System models, should be supported by the modelling platform.

vii) System thinkers should be able to expand the collection of modelling techniques available at their disposal in the modelling platform.

viii) System thinkers should be able to create different views of their modelled systems.

ix) System thinkers should be able to save model state and resume modelling at a later point in time.

x) System thinkers should be able to rework their models from any point of their model formulation.

xi) System thinkers should be able to reuse their models across different modelling instances.

xii) There should be a common elemental data structure/schema for the models so that interoperability amongst the models can be supported easily.

xiii) There should be a common collection of primitive graphical user interface (GUI) elements using which all the user interactions are expressed. This collection of GUI elements should be extensible so that new models can be supported.

xiv) There should be a set of core functionalities that is common across all modelling techniques.

xv) Information pertaining to a model can be collected in any format. This information should be converted and expressed in a unified format which is common across all models.
6.4 Quality Characteristics of Systems Modelling 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 Systems modelling platform.

1. Functionality:
   a. Suitability: Aids systems thinkers in performing atleast 80% of their modelling tasks
   b. Suitability: Atleast one modelling technique can be used by the systems thinker to model their system-of-interest
   c. Accurateness: Systems thinkers should be able to accurately represent the model of their system-of-interest.
   d. Interoperability: Systems thinkers can transfer exchange their model information amongst different models.
   e. Compliance: Complies with all the modelling constraints advocated by the modelling technique.
   f. Security: Systems thinkers can be assured that atleast 80% of their system-of-interest can be modelled in the platform.
   g. Security: Uses open format for storing information in storage media and while exchanging information between models.

2. Reliability:
   a. Maturity: Modelling techniques should be mature enough to help address 80% of the modelling needs of systems thinkers
   b. Fault Tolerance: Any issues that arise due to interpretation of a modelling technique should be handled gracefully.
   c. Recoverability: On exceptions, uses rules and conditions to rollback or abort tasks (50%)
   d. Recoverability: Store and Retrieve models and model workspaces from a storage media

3. Usability:
   a. Understand ability: System thinkers can comprehend and use the model (100%)
   b. Learnability: Guides through the steps of using the modelling platform (50% guidance)
   c. Operability: Uses familiar interfaces for interacting with the User (80% compatibility)
4. Efficiency:
   a. Time Behaviour: System thinkers should be able to model their system-of-interest within a predefined time frame. The platform should not be processing forever.
   b. Resource Behaviour: Platform should use minimal resources to satisfy the modelling needs of the systems thinker.

5. Maintainability:
   a. Analysability: System thinkers can keep track of progress always (100%)
   b. Changeability: System thinkers can change their models on demand (100%)
   c. Changeability: System thinkers can model the same system in multiple models.
   d. Stability: Absorbs 80% of perturbations introduced by the working environment.
   e. Testability: System performance and its use to model a system is reproducible (100%)

6. Portability:
   a. Adaptability: Adapts to change in the working environment (100%)
   b. Install ability: It should be easy to expand the collection of modelling techniques that are available in the Platform (90%)
   c. Conformance: Conforms to Modelling constraints (100% compliance)
   d. Replace ability: System thinkers can replace any of their Models (100%)

6.5 Decomposition of the Systems Modelling Platform

Systems modelling [51][106-109] aids in capturing the static nature of the system, including the structure of the system and its usage. It also aids in capturing the dynamic nature of the system, by means of how the components of the system interact with each other and the environment. In essence, systems modelling is the act of defining the structure and behaviour of the system in order to facilitate an abstract expression of the system. The resultant model is expressed by using the constructs of a modelling language which defines the syntax and semantics of the model elements. A model arises from the necessity of expressing the system by utilizing a specific set of model elements for the purposes of understanding, diagnosis, intervention and improvement. It involves understanding how the resources necessary for creating the model is organized formally, semantically and how it is represented and how these representations can be acted upon to produce effective transformations.
Systems modelling involves: a) Identifying the schema that needs to be adopted, b) Identifying the system elements that needs to be captured, c) Identifying the model elements that correspond to these system elements, d) Identifying the relationship between these model elements, e) Identifying the properties of the model elements in accordance with the schema, f) Identifying the properties of the relationship between model elements in accordance with the schema, g) Expressing the model elements and their relationships in an appropriate form, h) Tagging the form with the properties of these elements and their relationships, i) Ensuring consistency of the representation, and j) Analysing the model for consistency with the System-of-interest. The objective is to setup the necessary infrastructure that can aid in capturing different models of the same system-of-interest.

Accordingly, the systems modelling platform can be decomposed into the top level components: a) Systems modelling framework that supports the ability to integrate different models, b) Systems modelling framework run-time that supports the framework and also enables model integration, c) Modeller container which manage a collection of modellers, and d) Model repository which hosts the different models that are manipulated by the platform. The decomposition of the platform is illustrated in Fig 6.1.
6.6 Modelling Framework

The act of systems’ modelling starts with understanding the system to be modelled by a reductionist understanding, diagnosing it using archetypes, patterns, reference models, and templates and finally synthesizing/constructing an appropriate symbolic/mathematical/conceptual/physical representation. It is an act of identifying a configuration that satisfies the stated constraints and formulating it as mutual relationship of configuration items. It is also the process of developing an analogical system of relations expressed as a model, catering to a predefined form and convention, comprised of entities and relationships between them. Modelling involves identifying the characteristics of the system and expressing it as models so that the system can be understood [51][106-109]. It also allows visualization of information about the system represented by the model. The act of Modelling can be bottom up/inside out, by which details of the system are build utilizing knowledge about components (parts) and interconnections and how they compose together to realize the characteristics of the system, and/or it can be top-down/outside in, by which details of the components (parts) and interconnections are extracted from the knowledge of the whole.

Figure 6.2: Decomposition of Modelling Framework
The modelling framework utilizes a collection of modellers pertaining to different modelling languages, and corresponding modelling notations to capture an abstract representation of the system-of-interest in the form of a model. The different modellers that are plugged in provide the necessary capabilities in terms of syntax validation, diagramming and model manipulation to aid the modelling framework in expressing the different models. In order to develop the model representation, it is necessary to understand the system-of-interest and encode it using the chosen modelling notation. Additional information like the model element characteristics, causal relationships, and characteristics of the causal relationships needs to be expressed using this notation in order to have complete information. In this endeavour, the language that is used for representing the model and the process/technique used for developing the model is of essence.

Typically, a modelling technique would comprise of: a) a development life-cycle, b) a set of resources, c) a set of information structures and information flows, d) modelling conventions, e) model validation criteria and e) a schema for the model. As shown in figure 6.2, the modelling framework uses the services provided by the modeller to create a representation of the system of interest. It uses the services provided by the System Interrogator to create a concise understanding of the system based on stakeholder concerns. It uses the services provided by the model transformer to transform the model (either partial or full) that is encoded using a specific modeller into a form that is suitable for consumption by another modeller. It uses the detailing services provided by the model elaborator to add additional details to the model based on the user needs. It uses the services provided by the model analyser to validate and verify whether the encoded model is complete in all aspects. It uses the services provided by the integrator to develop the unified model of the system.

6.7 Modelling Framework Run-Time

There are many modelling techniques, modelling tools, and modelling languages that can be used by system's thinkers. A modelling platform that supports all these techniques and tools becomes too complex to design as a single monolithic system. Modularity and extensibility are design principles that facilitate extension of the platform so that new modellers can be added to it as and when demanded. However, the class of extensions that can be supported at run-time are anticipated extensions which are predominantly variability of the parts along certain dimensions. In the case of the modelling platform, the modellers are candidate parts. A good design practice in this situation, is to embed the necessary extensibility services in the run-time system. The objective of the run-time system is to provide necessary
interfaces that support interaction with the modellers, sharing of information with the modellers, integration of the modellers with the modelling platform and management of the modellers’ life-cycle.

The bus design pattern can be used in the modelling framework run-time [MFR] for supporting information sharing between the modellers. The host-controller design pattern can be used to facilitate interaction between the modellers (say model exchange). The interoperability-bus design pattern can be used in MFR for supporting information exchange between the modellers. The plug and socket design pattern can be used in the modelling platform to dynamically plug and play modellers at run-time. The event-driven approaches adopted by graphical user interface based operation systems can be used in MFR to handle the various events that occur in the modellers.

As shown in figure 6.3, the modelling framework is a composition of: a) Modeller Socket which serves as the extension point for a modeller, b) Host Controller which serves as the node that provides resources, capabilities and services to the modellers, c) Information Bus which provides resources, capabilities and services for manipulating and transforming data, and d) Event Handler which serves as the placeholder for invoking services from various modellers and processing their results in response to the various events triggered by the modelling platform.

### 6.8 Modellers

A model is a simplified representation of the system and is intended to promote understanding [51][106-109]. There is always a trade-off as to the level of detail that is included in the model. The extent of a models’ usefulness depends on the extent to which it promotes understanding. If the model is overtly
simplified with less detail then the model does not promote understanding, on the contrary, too much
detail makes the model complicated. In order to create a holistic view of the system, multiple models are
integrated, with each model defining a set of properties corresponding to the respective concerns
[51][106-109]. The different models allow system thinkers to look at the system at different levels of
detail. They can be used to structure, identify, analyse and synthesize systems wherein each model
commutes with the systems and relates to it.

Each model is understood, worked upon and then composed keeping in mind the constraints of the
system and the conditions in which the system exists. They can be either independent or dependant and
dynamic. Each model is a different perspective in representing the system and if semantically motivated
explains how the system is understood, analysed and synthesized [51][106-109]. In many situations, these
individual models are often treated as independent models which is not really the case. Many of these
models overlap and they are also projections of the system-of-interest in accordance to a specific
viewpoint and contain a subset of the constructs, relationships, and attributes pertaining to this viewpoint
and provide insight into certain aspects of the system. It is necessary to aggregate/compose all these
models together to form a unified model to create a better understanding of the system.

**Figure 6.4: Software Decomposition of Modeler**

Modellers are utilized to express a system by means of the model elements corresponding to a
modelling language. The extent to which a modeller is effective and suitable for modelling the system-of-
interest depends on the clarity of the models that are developed and the ease with which the model can
be understood. There are three main characteristics of a system that a modeller captures as part of the
model: a) structure of form of the system, b) semantics or meaning of the system elements, and c)
pragmatics or usability of the system. The structure of the system is predominantly captured in terms of entities (model elements) and their interrelationships so that the system is represented unambiguously. The semantics of a system is predominantly captured as an abstract representation and it is a composition of the semantic entities corresponding to individual system elements. It defines the way the model is interpreted. This interpretation and assignment of meaning is according to predefined rules/methods. The operational semantics method defines the meaning of the model based on the sequence of interpretation configurations; the denotational semantics method maps a model directly to its meaning using a valuation function and the axiomatic semantics method expresses the properties of the model using axioms and inference rules and the meaning of the model depend upon the kind of properties that can be proved about the model. The pragmatics of the system is predominantly captured as process models using which the way the system is used to perform certain functions in real-world is captured.

Accordingly, a modeller essentially supports three different activities: a) Structure modelling using which the different entities, their interrelationships and their associated characteristics are expressed, b) Semantics modelling which the semantic interpretation of the modelling elements are expressed, and c) Process modelling using which the way the different entities and their capabilities are utilized to achieve certain objective are expressed. Accordingly, the corresponding model elements are manipulated on the underlying model. The modeller ensures that this model doesn’t violate the semantics of the modelling language. The modeller, as shown in Figure 6.4, is a decomposition of: a) Structure modeller for expressing the form of the system, b) Process modeller for expressing the processes that the system entails, c) Semantic modeller for expressing the way the model elements are interpreted, and d) Model interpreter for associating meaning to the modelling elements based on the underlying modelling language.

6.9 Modeller Run-Time

As discussed in section 6.8, there are many modellers that can be used for multi-modelling. Each of these, capture certain characteristics of interest using several techniques and also represent the model using different notations. The resulting models are interdependent as well as consistent and aligned to each other, since they represent different aspects of the same system. The modellers provide the desired capability for modelling the system using different languages. The modeller run-time provides the necessary support for run-time library, event handling, model storage and to plug and play modellers. Once a modeller is plugged into the framework, the run-time serves as an abstraction layer that extends the capabilities of the framework by considering these modellers as constituent elements of the
framework. The run-time utilizes the interface services offered by the connection receptacle to interact with the socket. This enables it to dynamically attach and detach from the modelling platform. The run-time utilizes the capabilities provided by the service desk to support different functionalities necessary for modelling. It utilizes the data services provided by the information bus to exchange model information with the modelling framework. It utilizes the eventing services provided by the event manager to monitor and handle the events that are triggered by the modeller as well as the modelling framework. This decomposition of the run-time is presented in figure 6.5.

![Diagram of software decomposition of modeller run-time](image)

**Figure 6.5: Software Decomposition of Modeller Run-Time**

### 6.10 Discussions

The experimental system addresses its primary goal of enabling plug and play of modellers while maintaining integrity of models by:

a) Encapsulating modelling functionality by encoding them into the modeller component.

b) Encoding the plug and play interface mechanism as socket and compatible plugs.

c) Encapsulating the plug capability as part of the modeller run-time component.

d) Encapsulating the socket capability as part of the component framework run-time component.

e) Dispatching the events to the appropriate handler by using the plug & socket interface.

f) Providing the services to the appropriate consumer by using the plug & socket interface.

g) Managing the model information by using the exchange services of the plug & socket interface.
h) Ensuring model integrity by considering each modeller as contributing to different views of the integrated model.

The three critical elements, events, services and models that are required for proper functioning of the systems modelling platform have been made accessible through the plug and socket interface. Additionally, by conforming to a meta-model, these models are consistent to each other by design and collectively they contribute to the integrated model of the system. As a result, it can be observed that “plug and play architectures did not impact data integrity”. Accordingly, the hypothesis “If components that are dynamically attached and detached at run-time can maintain consistency relationships on the data they manipulate, then an integrated view of the data can be created” was verified. Further, the underlying principles and constraints that are discussed in the subsequent sections enable the transition from an extensible system architecture into an integrated extensible system architecture.

6.10.1 Principle of relationships among elements

According to Maier [51], in order to maintain consistency on the data that are manipulated by sub-systems, it is necessary to ensure that there exists consistency relationships among the elements so that they can collectively provide additional value. In case of the modelling platform, the modellers conform to corresponding meta-models, while serving as extensible components. The desired consistency is maintained at the meta-model level which is not violated by the modellers. As a result, the encoded models are consistent to each other. In essence, this means that the component framework provides the context for the modellers.

6.10.2 Principle of accommodating change

According to Maier [51], in large systems, evolution is a process of ingress and egress. Accordingly, evolution in the case of the modelling platform is made possible by the ability to plug a modeller; to plug an upgraded or new modeller; and to unplug a modeller that has outlived its expectations.

6.10.3 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 modelling platform, the plug interface of the modeller and the socket interface of the component framework are designed such that only compatible modellers can utilize this interface.
6.10.4 Principle of being understandable

According to SEBOK [137], it is necessary to create abstractions wherein unnecessary details are obfuscated while focusing on essential characteristics. In the case of the modelling platform, the way the different models are integrated together is obfuscated, with the users focusing on expressing the models of their interest and the integrated model is formed based on the underlying unified meta-model.

6.10.5 Principle of similarity

According to Read [110], productivity can be maximized through favourable experience by providing a similar user interface. In the case of the modelling platform, all the modellers share common modelling capabilities. Variability in such cases, is achieved by capturing model information that is relevant to the meta-model that is supported by the modeller.

6.11 The Plug and Play Architectural Abstractions

This section introduces and elaborates the plug and play architectural abstractions of the systems modelling platform. 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 systems modelling platform”. These abstractions are:

6.11.1 Plug

A plug comprises of 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, data and services that the plugged component manipulate as part of its computation. 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.

6.11.2 Socket

A socket is a handle that allows the plugged components to utilize the data, events and services that are exposed by the component framework. It facilitates two way communication of data and one way communication of events and services between the plug and 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.
6.11.3 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.

6.11.4 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.

6.11.5 Event

An event is an encapsulated set of software actions that are triggered during component execution, along with a set of data that capture the computational state under which the event was triggered. Such events are triggered when a set of pre-defined execution conditions are met. They are triggered inside the plugged component and dispatched to the component framework for processing.

6.11.6 Service

A service is an encapsulated set of software functionalities offered by a plugged component, along with a set of policies of usage that declare the information necessary for the successful execution of the service, that can be invoked by other components for different purposes.

6.11.7 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 Information repository which hosts the information that is gathered, processed and transformed by the plugged components, and d) The components manager which manages the plugged components and their respective run-times. Figure 6.6 provides a decomposition of this model.

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 consuming various services that are supported by the plugged component and for managing data flow between the plug and socket.

6.12 Summary

This chapter introduces and elaborates the architectural abstractions that were utilized in the experimental system titled Systems Modelling Platform [94-95]. The approach to transition from an extensible system architecture into an integrated extensible system architecture was illustrated by utilizing a set of pluggable components that are attached and detached to the platform on demand. These pluggable components conformed to an underlying meta-model thereby ensuring consistency
relationships on the information they manipulate. As a result, the hypothesis “If components that are dynamically attached and detached at run-time can maintain consistency relationships on the data they manipulate, then an integrated view of the data can be created” 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.