Architecture and Design of Corrosion Prediction Software Multicorp

Arkopaul Sarkar
Institute of Corrosion and Multiphase Technologies
Ohio University
Athens, Ohio
sarkara1@ohio.edu

Dušan Šormaz
Institute of Corrosion and Multiphase Technologies
Ohio University
Athens, Ohio
sormaz@ohio.edu

Abstract: *Multicorp is a corrosion prediction application based on a simulation engine called CorrSim developed in FORTRAN. Multicorp application is able to take information on various chemical and environmental conditions from user through a user interface. An underlying model called Multicorp Model is responsible for managing, calculating, transferring data to CorrSim engine and also reporting corrosion rate and other information. Multicorp application also supports persistent storage and retrieval of various corrosion prediction models as a form of XML files. In this paper, the architecture of Multicorp application, including its data model, data storage and retrieval strategies as well as flow of data within models and engine and general use case demo.*

Index Terms—Software, Design Pattern, Case Study, XML, Corrosion

I. Introduction

Institute of Corrosion and Multiphase Technology (ICMT), part of Ohio University, has been conducting research on internal corrosion of oil wells and pipelines for more than two decades. An industrial consortium joined by worlds twelve leading oil and chemical companies not only supports the entire research and facilities but also work closely with researchers to investigate new ways to deal with corrosion in multiple areas in refinery, rig and transportation of crude oil.

Multicorp, developed by Institute of Corrosion and Multiphase technology researchers and developers, models different electrochemical mathematical equations which can predict underlying corrosion faithfully by taking information on physic-chemical environment. Multicorp software is able to predict corrosion rate for various environment through numerical simulation of the chemical reactions over time and helps user to analyze the root cause of the corrosion with an insightful analysis dashboard.

Although the core of the simulation engine, called CorrSim, is developed in FORTRAN, Multicorp model built in Visual Basic .NET package is solely responsible for managing modeling information which is basically different parameters of physico-chemical environment. Section II describes a brief background of Multicorp development. In section III, the architecture of the model and its instantiation strategy is described in detail. Multicorp implements its own file system to store modeling data persistently. In section IV a detailed discussion on storage and retrieval strategy of Multicorp is presented. In section V, overall flow of the data among models and user interface connection to CorrSim engine and build strategy of Multicorp application is discussed. A walkthrough of the user interface of Multicorp along with analysis of different performance metrics of Multicorp is presented in section VI. Before all of these, a short history of Multicorp is presented in the next section.

II. Background of Multicorp

Corrosion prediction in oil and gas pipelines is a critical aspect of modern oil and gas exploitation [1]. Models to predict corrosion can be classified into two groups:

Empirical models are based on experimental measurement of corrosion rates, and regression model (or multiregression models) to fit the experimental data. Those models tend to be rather simple (deWard [2]) and they capture limited ranges of independent variables (for which experiments were performed). Some of those models have been implemented and marketed as corrosion prediction standards (Norsok [3]) and software [4].

Mechanistic models, which attempt to capture the electro-chemical processes that govern corrosion formation, including mass transport, diffusion, gas and liquid flow, and electrolyte formations. Those models are of different levels of complexity. Freecorp [5] uses a simplified electrochemical model of steady state corrosion formation to predict corrosion rates. Solution to corrosion formation has been presented in the form of a system of partial differential equations describing both transitional and steady state behavior [5]. Such model is usually solved by some finite difference method in the form of dynamic simulation

Multicorp software is based on the work by Nesic and his collaborators at ICMT in modeling multiphase flow corrosion mechanisms. Solution to partial differential equations is obtained by in-house solver, which is now converted into CorrSim solver implemented in Fortran 90. Initial versions of Multicorp (MULTICRP V3 and MULTICORP V4) were implemented in VB6 as preprocessor and User-interface language. The work described in this paper relates to redesign of those earlier systems into efficient corrosion prediction tool using the modern software development technologies (object oriented modeling, XML, and MVC paradigm).

III. MULTICORP MODEL

Multicorp Model defines different types of physical and chemical environments through different classes from a hierarchical class model. The class model is created by taking advantage of inheritance property of any object oriented modeling where different sub-models inherit attributes and properties from their parent model. The entire Multicorp Model is depicted in the UML class diagram in figure 1. Brief descriptions of important models in the Multicorp are given below.

A. AbstractModel

The root of the model is an abstract class called *AbstractModel* which owns all general attributes and properties of every model in Multicorp. Two most important attributes of *AbstractModel* are two collections which store parameters and parameter groups for individual models. All other attributes can be classified into three different categories.

Identifier attributes – Identifier attributes store model specific identification data. *ModelName* stores the textual name of the model, *ModelID* stores the runtime instance number of the model, *ModelType* stores the type of model and *ParentModel* holds the textual name of the parent model.

UI attributes – UI attributes help model to connect to the GUI element, manages different user interaction in the model. Few examples of these type of attributes are *DisplayName* (textual name of the model shown on the UI), *mInstruction* (text to store instruction for a particular model), *ModelState* (state of the model at a certain time of the corrosion modeling process) and few integer flags to identify different states of the model during user interaction and execution.

Listener attributes – Every model registers to different listeners in Multicorp to either respond to any user interaction on GUI or any change in other models. Through listeners models work in a coherent and synchronized manner in Multicorp. All listeners are stored in a collection called *ModelListeners*.

AbstractModel also has the set of most generic properties. Most of these properties are either targeted to managing different attributes of the model notably parameters and parameter groups. There are a set of properties which are designated to read and write into XML files meant for persistent storage of data. Another set of properties perform all types of calculation in the model based on the values of parameters. The calculation properties are mainly overridden in sub models and different chemical equations are implemented which perform calculation to produce results specific to a particular model.

B. CompositionModel

Composition model is inherited from ChemistryModel which is in turn inherited from *Abstractmodel*. Chemistry model is responsible for storing contents of different chemical components of aqueous, gas and hydrocarbons in the system. Numerous chemical calculation is performed in this model to calculate the percentage of different ions in the system, concentration of gas and other chemicals and saturation level of pH and other hydrocarbons.

C. FlowModel

Flow model is responsible for capturing different metrics for defining the condition of various gas and liquid flow in the pipeline. *FlowModel* itself is an abstract class inherited from *AbstractModel*. *FlowModel* is further classified by two abstract classes such as, *AbstractGasFlow* for gas flow and *AbstractLiquidFlow* for liquid flow. Different flow parameters such as, viscosity, velocity, surface tension, percentage of mixture and superficial velocity is used to calculate not only the flow rate of the oil, water, or mixture in the pipeline but also flow pattern, stress on the pipeline, slug deposition rate and water wetting durations.

D. CondensationModel

CondensationModel is a special type of corrosion model which captures environmental parameters which affects the corrosion of the top of the pipeline. This class is inherited from *AbstractCondensationModel* which in turn is inherited from *AbstractModel*. *CondensationModel* captures pipe quality, outside environment variable for either land, ocean or air and insulation data. This model can calculate water and hydrocarbon condensation rate at top of the pipeline.

E. PipeLineModel

PipeLineModel is inherited from *ProtoPipeline* which is an abstract class extended from *AbstractModel*. *PipelineModel* lets user to define pipe topography. Pipelines can be layed in 100m sections with chosen inclination and declination. Along with the topography, every pipeline section also stores individual properties such as, diameter, roughness, thickness of wall, conductivity and one or more insulation layer and their properties.

F. CorrosionModel

CorrosionModel is the most important model of Multicorp because this model manages the simulation. As *Corrsim* is packaged in a Fortran DLL file, this model also manages native function calls to *Corrsim* dll. Multicorp application supports three types of simulations, single point, parametric and line.

Corrosion at a single point is simulated over time through *PointModel*, which is inherited from Abstract Model. Parametric simulation simulates corrosion over time by varying different parameters. Batch Model is mainly responsible for managing this type of simulation. In the end, *LineModel* manages line run, which simulates corrosion along the length of pipeline. Both *BatchModel* and *LineModel* class are inherited from *MultiplexModel* class which is inherited from *AbstractModel*.

G. Parameter Model

Multicorp Models stores data in different types of parameters. Parameters are tightly linked to user interface elements. Every types of parameter are inherited from *AbstractParameter* class (see *Figure 2*).

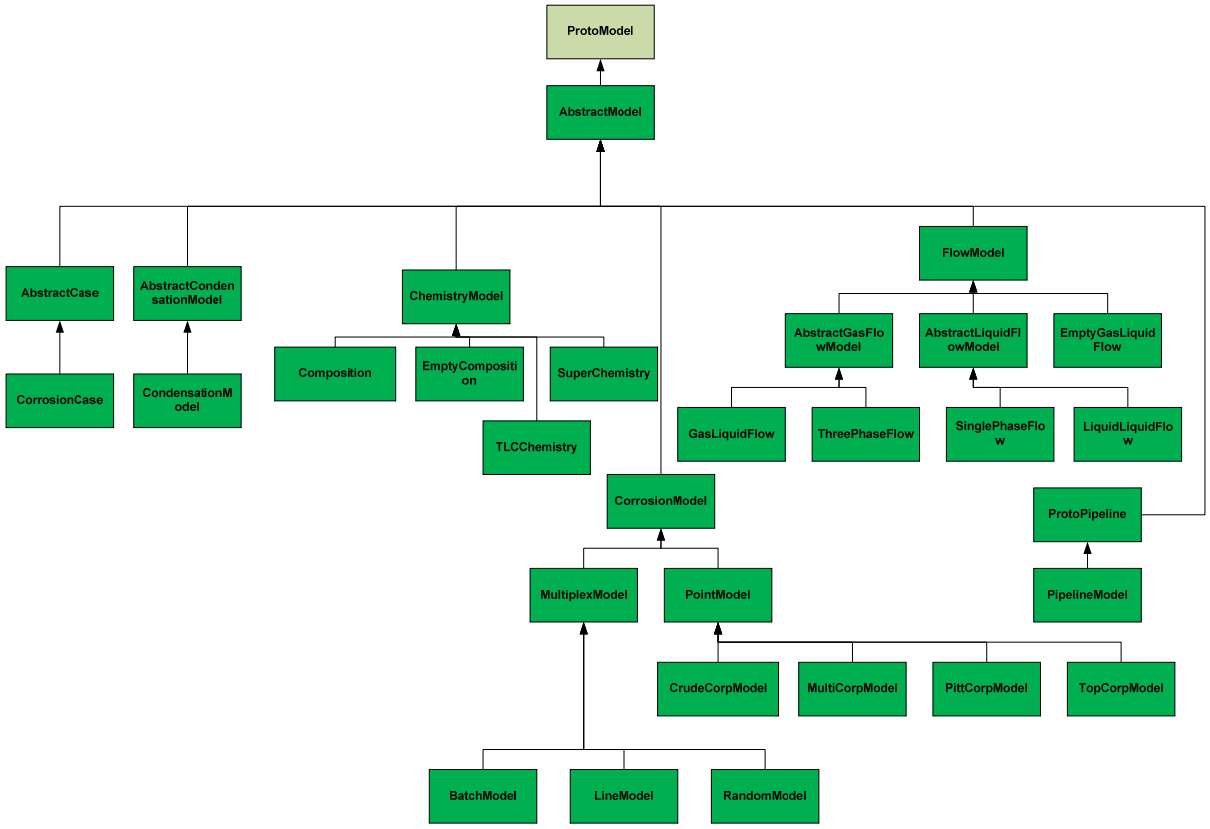

Figure 1: Class hierarchy diagram of Multicorp Model

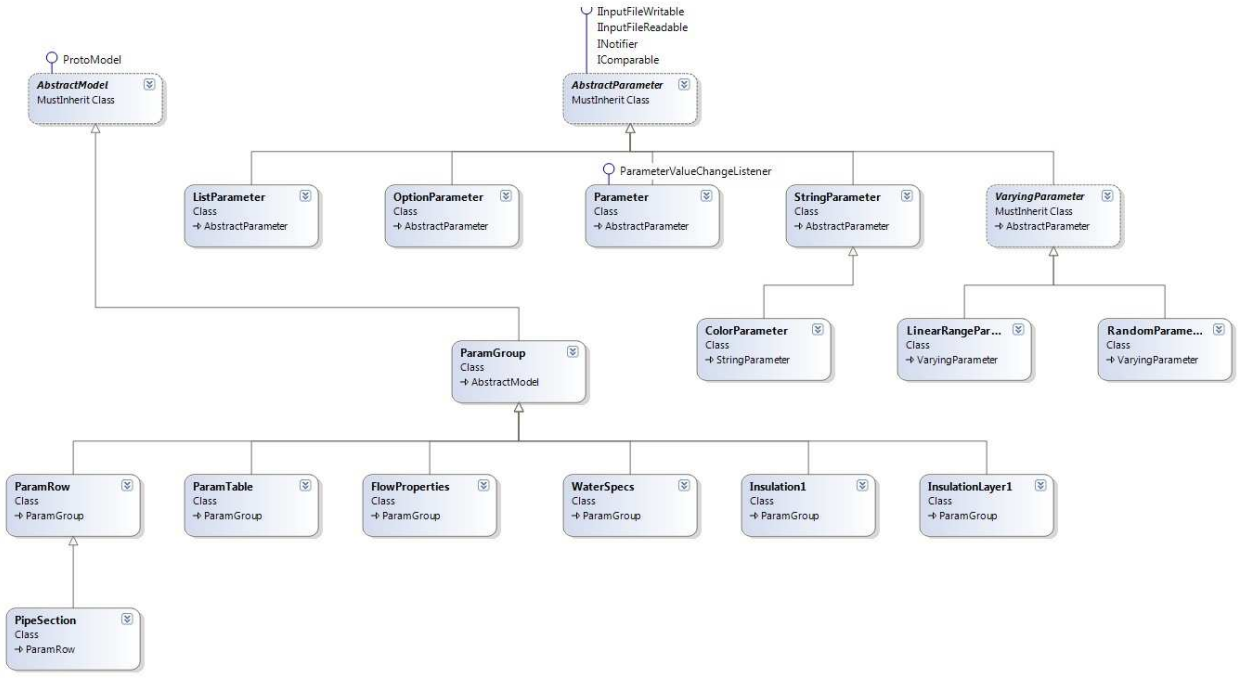

Figure 2: Multicorp Parameter model

Parameter class is the most used type of parameter which stores values of different types of chemical and physical species and samples. It is worth to mention that Multicorp supports multiple languages and unit conversion. Every parameter thus also has list of units it supports along with conversion factors. This enables user to change units of individual parameter separately on the fly. *Parameter* also stores two types of limits such as, hard limit and soft limit. For the reason that corrosion chemistry is always valid in a certain range of these parameters, user can be warned easily with the help of these limits, in case any value is entered which violates these limits.

Every model stores several parameters from same family under a parameter group. Parameter group class *ParamGroup* is also inherited from *AbstractModel* giving Multicorp model tremendous flexibility to store parameters either directly under a model or under a parameter group. Two special types of parameter groups are *ParamTable* and *ParamRow*, both inherited from *ParamGroup*. These two special Parameter groups are responsible for storing table data. Same principle is applied to store *PipeLine* topology data in *PipeLineModel* where each pipe section data is stored in individual *ParamRow*.

H. Multicorp Model association and dependency

Multicorp system instantiate different models to build a complete corrosion profile, based on the options chosen. For example, a corrosion model investigating bottom of the line corrosion for only water flow and simulating corrosion at single point will consist of instances of *CompositionModel*, *SinglePhaseFlowModel* and *PointModel*. *CorrosionCase*, inherited from *AbstractModel*, is also a model but it works as composite. Following classical composition pattern, where multiple similar objects are stored in a composite, *CorrosionCase* stores multiple instances of various model classes. As every model including *CorrosionCase* is inherited from same *AbstractClass*, any generic operation requested to CorrosionCase may also requested to all other objects in *CorrosionCase*. This is mainly useful for least common denominator type of operations such as, data load, calculation and save, where this composition model provides tremendous ease in development and object management. However, it is to be noted that in traditional composition model, Composite associates with its components individually by 'has a' relationship. In *CorrosionCase*, all models are stored in a collection of type *AbstractModel*. These collection stores instances of specific subtypes at runtime but *CorrosionCase* can perform any least common denominator type operation on any model stored in the collection without knowing which subtype it is calling the operation on.

I. Multicorp factories and prototypes

Multicorp implements abstract factories to return concrete instances of models casted into abstract parent class of the particular model. One factory is created for each of the main models such as; Composition, Flow, Condensation, Pipeline, Simulation, and *CorrosionCase* (see Figure 3). Every factory class is singleton and has a static function createModel, inherited from *AbstractFactory* which returns the instance of the model. Thus, user needs to know only the type of *AbstractFactory* to access *createModel* method of any other factory. In this way, complex conditional statements are abstracted in the factory class and concrete model classes are never exposed.

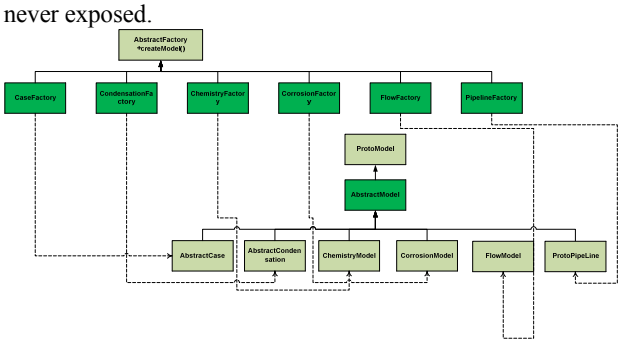

Figure 3: Abstract factories to instantiate Multicorp Model

Moreover, abstract factories don't create instance of any model directly. Instantiation process of any model is complex and heavy of memory as it includes I/O operations and XML query (explained in section IV). Therefore an ingenious object oriented design pattern, called prototype, is implemented to avoid repetitive execution of performance-heavy operations. Prototype is a classical creational pattern often used along with Abstract Factory. Prototype design pattern leverages on reusability of objects and saves memory and execution time [6]. *AbstractModel* class being the base class of all other model implements a pure virtual clone which is implemented by sub-models (See Figure 4). *ProtoManager* is a singleton class, which implements a collection to store instances of every model, which are called prototypes. When any model is requested by the corresponding factory, *ProtoManager* performs a deep clone on the prototype instance saved in the collection and returns it. Deep clone copies every attribute from the prototype to the clone but doesn't store any reference between them. Thus changes on the cloned instance doesn't impact prototype instances.

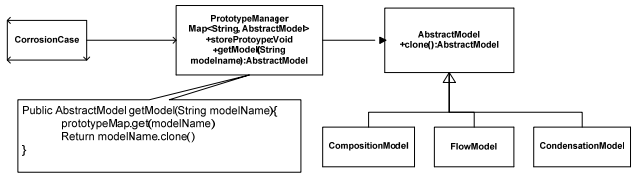

Figure 4: Prototype pattern in Multicorp

IV. Data Management

Multicorp system models corrosion by making composite of different models. This is explained in Section III.H. As a desktop application Multicorp saves model data in an XML formatted file. One big reason for choosing XML as a formatting style is that XML is a well-formed document as well as extremely flexible to support any user defined schema [7]. In addition to DTD is a common practice to ensure the validity of the document, Multicorp needs set of parameters for each model with default values, units and limits besides just the

structural integrity of the document. Therefore Multicorp uses a default XML document which stores all default values, descriptions, UI tags, units and limits for every parameter inside a well-planned XML node hierarchy which defines the composite structure of models. When Multicorp is asked to save the modelling data, it creates another XML document of type .mcinput, which follows the same node hierarchy as in default XML document and but stores only current values for individual parameters. As many non-changeable attributes of the parameters are always found from default XML document, .mcinput file doesn't store that information, resulting in much smaller file size.

Along with .mcinput Multicorp also saves simulation data and some temporary files along its process. They are listed below.

.mcorp - This is an archive file which contains other Multicorp files , which are

- .mcinput - Stores model data
- .input - FORTRAN generated pre simulation data
- .output – FORTRAN generated simulation results
- .mccase – Multicorp generated case file to execute in CorrSim project.

.tmpinput – A binary file of type .dat which contains object serialized data. This is used to save state of CorrosionCase temporarily.

The entire strategy of handling different files is explained as a flow chart in *Figure 5*.

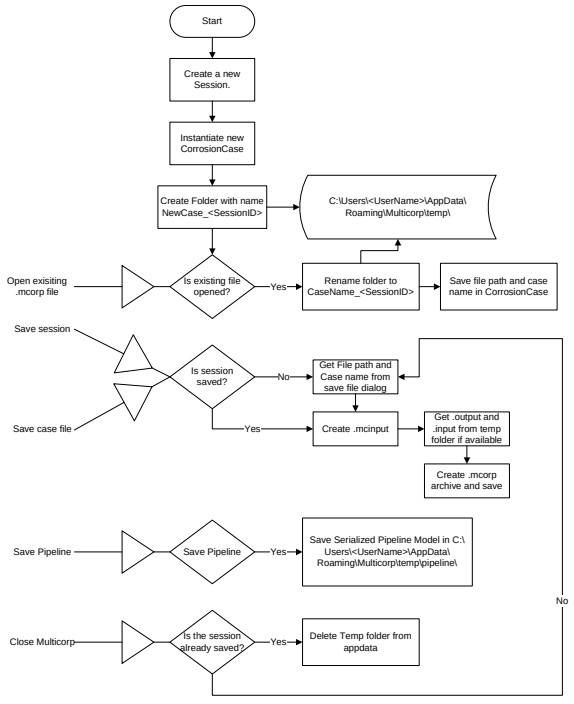

Figure 5: Multicorp file management strategy

Multicorp system is going through rapid evolution. In near future Multicorp system is going to be available through service APIs in near future. Moreover, work is also underway for transforming Multicorp into multi-tier web based product. Multicorp data architecture supports multiple types if data sources and storage system besides XML. To facilitate this, *AbstractModel* implements variable of type i. *DataHandler* is an interface which contains pure virtual methods such as, saveModel and loadModel. This interface is extended by concrete data handlers such as, XMLHandler or *DatabaseHandler* as displayed in *Figure 6*. Concrete handlers implement complex data management operations specific to the data targets. However, models don't need to know those specifics of data management and can simply call *saveModel* or *loadModel* method on the handler variable to load and save data.

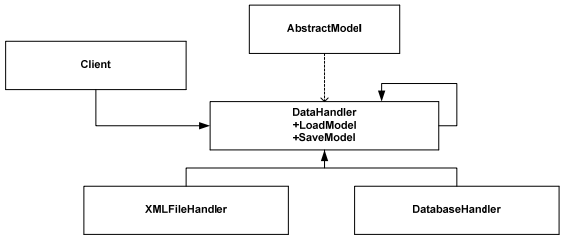

Figure 6: Abstracting data handling in Multicorp

V. System Architecture

In this section different design aspects of the overall system are discussed.

A. User Interface

Multicorp desktop application is built in conventional model-view-controller architecture. Multicorp model, discussed in section III, manages the data. Different view element depends on various controller classes, either custom or member of .net framework. The smooth interaction between GUI elements and data models is established by implementation of observer pattern, in which event raised by any GUI element is broadcasted to all observers registered to the event sender. For example if user changes a value of a parameter in GUI, every model listening to the event, will trigger the corresponding action (See Figure 7).

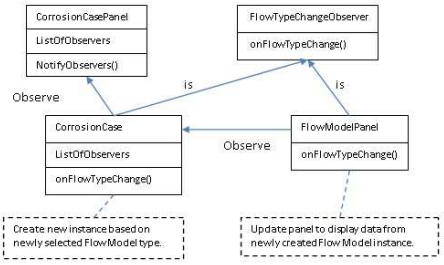

Figure 7: Observer pattern in Multicorp

This particular pattern lets the execution control transition from view to model seamlessly over a loose coupling between listener and sender [6]. The user interface of Multicorp is built closely following Microsoft Office's ribbon style [8].The interface is separated in four resizable areas, such as, ribbon area (top), process area (left), data area (middle) and trace area

(bottom) (See Figure 8). Ribbon area contains buttons, which triggers generic actions on Multicorp models. Every time a new tab is selected buttons are changed depending on the tab content. Process area shows the steps to create a corrosion model and also displays the status of every model. Data area displays parameters in groups, charts, tables and various other output elements. In the end trace area displays vital messages including warning and errors at the time of corrosion modeling.

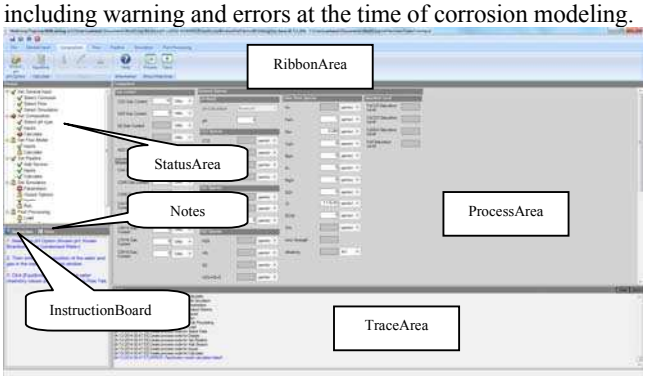

Figure 8: Multicorp user interface

In Figure 9 the loading sequence of Multicorp user interface components is explained in a sequence diagram. In the sequence diagram, it is shown that MulticorpWindowApp is the starting point of execution, which in turn instantiates and run MulticorpForm in a separate UI thread. *MulticorpForm* extends .NET library class for UI container called *Form*. *MulticorpForm* instantiates and build different panels, namely, RibbonArea, MiddleArea and StatusArea inside it sequentially. MiddleArea is the container for different other panels; such as, ProcessArea, TracePanel, and InstructionBoard.

B. Dynamic Modeling

One of the unique features of Multicorp is to create additional models on the fly addition to the existing models. Every model of Multicorp, when instantiated to be part of a corrosion case, they are stored in a hash table as key value pair, where key is the name of the model. As explained in section III.H, when user request an action performed on a specific model, corresponding model is retrieved from the hash table and subsequent operation is performed on the model. This is possible because every generic function is owned by *AbstractModel*. The most important operation to be performed on any model is calculation of corrosion and various data. Models are capable of calculating itself too. This calculation is done based on the input parameters. Any new model created on the fly needs to know its parameters. This can be achieved by defining parameters and storing them in the parameter hash table of the new model. Next the model need to know implement a calculate function. The calculate method is not a part of *AbstractModel* but an interface called *ICalculate*. When a new model is created, a concrete implementation of *ICalculate* is supplied to the model. As the new model is extended from *AbstractModel* and *AbstractModel* has a dependency on the *ICalculate*, when user requests calculate operation on the model, the calculate method from the concrete calculation class is called. Moreover, every model is capable of storing multiple instances of subclass of *ICalculate*. At any point of time, only one instance is set active, however, it gives tremendous flexibility to users because alternative calculation logic can be implemented and compared without rewriting same block of code.

C. Multicorp build strategy

Multicorp is distributed in four different versions. They are Standard(CC-JIP), Water-Wetting(WW-JIP), Topcorp(TLC-JIP) and Topcorp-Water-Wetting(TLC-WW-JIP). Standard version provides basic features with other version including extra features on top of it. To facilitate this segregation, Multicorp is built in different functionally segregated modules (see Figure 10). Every module is compiled into separate DLL file. This type of modularized development helps Multicorp to package only required DLLs for a particular version and distribute as installable.

Corrsim is a separate FORTRAN development and has its own modular structure.

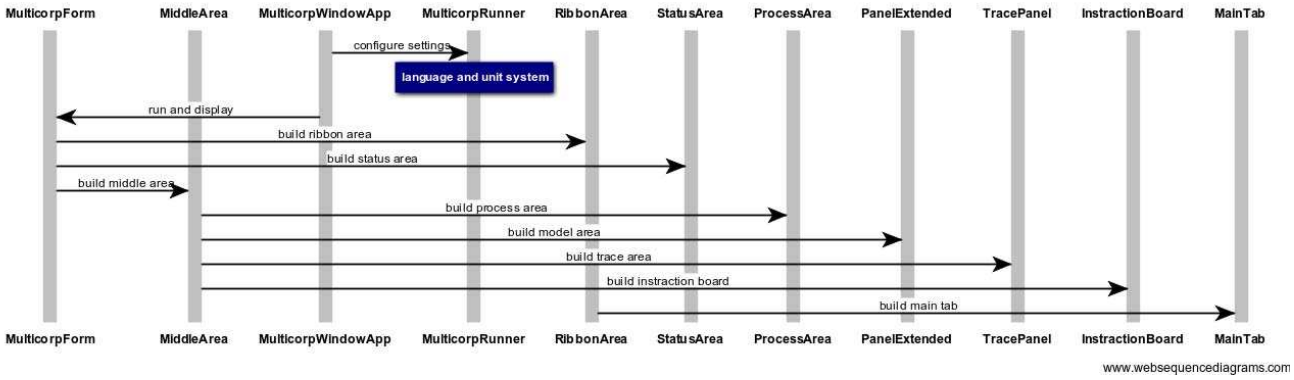

Figure 9: Sequence diagram of Multicorp GUI building process

After that compilation dependency, configured in Multicorp solution, links *Corrsim* DLL to Multicorp DLLs as reference. In that way, Multicorp function can make direct call to *Corrsim* functions.

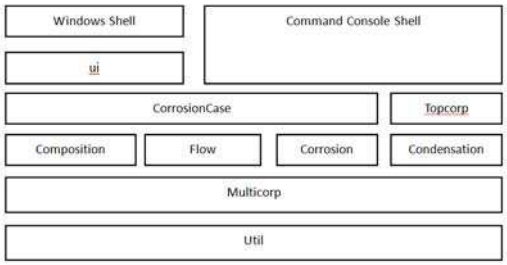

Figure 10: Package dependencies in Multicorp

Figure 11: Multicorp screen for configuring various models

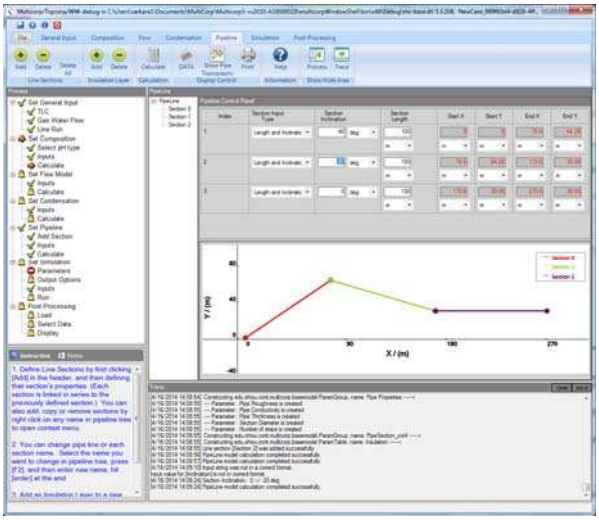

Figure 12 Multicorp screen for defining the pipeline topology

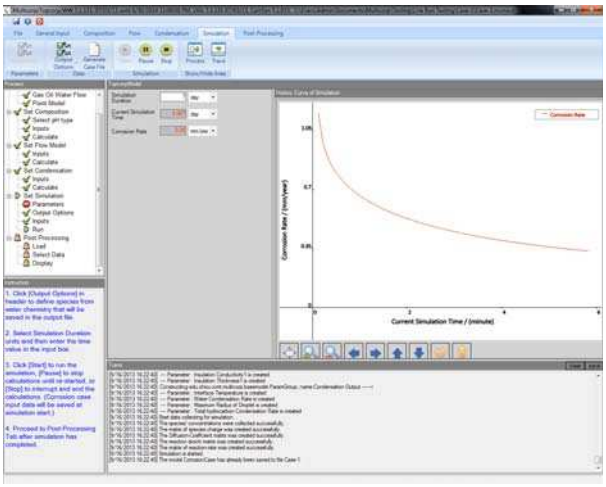

Figure 13 Multicorp screen showing dynamic simulation

VI. Demo and Performance Analysis

Multicorp has been tested with two groups of corrosion engineers: faculty and graduate students at ICMT (15-20 engineers), and ICMT industrial partners (25-30 engineers) in order to finalize it GUI to correspond to the ways that corrosion engineers would use to predict corrosion rates. Resulting user interfaces are shown in few figures. Figure 11 shows the general input screen where user configures various models that need to be calculated, by selecting select corrosion, flow and simulation type. All other tabs are dynamically generated once three selections are made. It is to be noted that the process tree shows the all steps required before simulation may be performed. Currently available tab and steps possible in that tab are indicated with a red arrow and all other tabs which are not available are marked as locked. The screen in Figure 12 show how user defines line topology in pipeline tab which only appears when line run is selected as simulation type. An interactive pipeline modeler helps user to define pipeline over an intended topology. The screen in Figure 13 shows the simulation tab while user is running a simulation and can monitor its progress in a dynamic plot.

Multicorp is also tested for its computationally efficiency and it has shown order of 10x speed up over previous version. In addition it is continuously been validated against experimental results obtained in ICMT labs and from its industrial partners.

References

[1] S. Nešić, "Key issues related to modelling of internal corrosion of oil and gas pipelines – A review," Corros. Sci., vol. 49, no. 12, pp. 4308–4338, Dec. 2007.

[2] C. de Waard, U. Lotz, and D. E. Milliams, "Predictive Model for CO 2 Corrosion Engineering in Wet Natural Gas Pipelines," Corrosion, vol. 47, no. 12, pp. 976–985, Dec. 1991.

[3] NORSOK, "M-506 CO2 corrosion rate calculation model." NORSOK, p. Rev. 2, 2005.

[4] Wood Group Intetech, "Electronic Corrosion Engineer." 2013.

[5] S. Nešić, H. Li, J. Huang, and D. Sormaz, "An open source mechanistic model for CO2/H2S corrosion of carbon steel," in NACE International Corrosion Conference & Expo, 2009.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design patterns: elements of reusable object-oriented software. Pearson Education, 1994.

[7] S. Holzner, Inside XML. Indianapolis, Indiana: New Riders Publishing, 2001.

[8] D. You, "Re-engineering of the Legacy Software Systems by using Object-Oriented Technologies," Ohio University, 2013.