Global Journal of Computer Science and Technology: C
Software & Data Engineering
Volume 15 Issue 6 Version 1.0 Year 2015
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Design of Transmission Pipeline Modeling Language

By Japheth R. Bunakiye & Prince O. Asagba

Niger Delta University, Nigeria

Abstract- General purpose software design and development involves the repetition of many processes, and the ability to automate these processes is often desired. To formalize a software process, such as modelling pipeline systems that transport fluids, an existing general purpose programming language (GPL) can be extended with its important aspects extracted as a model. However, the complexities and boundaries the programming language places on the ability to concisely and clearly describe the designing and modelling processes of the pipeline configurations can be difficult. The reality is that the library of a typical GPL Application Programmers Interface (API) constitutes class, method, and function names that become available only by object creation and method invocation, and as such cannot express domain concepts effectively. An alternative approach is to develop a language specifically for describing the processes. A language formalism that encourages domain specific development and as a tool for solving the complex problem of efficiently and effectively aiding the pipeline engineer in the design and implementation of pipeline configurations is presented in this paper. The language tool is used on the .Net platform for domain specific software development.

Keywords: pipeline engineering, modeling languages, design principles, domain-specific modeling (dsm), model transformation.

GJCST-C Classification : B.5.1 C.1.3

Strictly as per the compliance and regulations of:

Design of Transmission Pipeline Modeling Language

Japheth R. Bunakiye [α] & Prince O. Asagba [σ]

Abstract- General purpose software design and development involves the repetition of many processes, and the ability to automate these processes is often desired. To formalize a software process, such as modelling pipeline systems that transport fluids, an existing general purpose programming language (GPL) can be extended with its important aspects extracted as a model. However, the complexities and boundaries the programming language places on the ability to concisely and clearly describe the designing and modelling processes of the pipeline configurations can be difficult. The reality is that the library of a typical GPL Application Programmers Interface (API) constitutes class, method, and function names that become available only by object creation and method invocation, and as such cannot express domain concepts effectively. An alternative approach is to develop a language specifically for describing the processes. A language formalism that encourages domain specific development and as a tool for solving the complex problem of efficiently and effectively aiding the pipeline engineer in the design and implementation of pipeline configurations is presented in this paper. The language tool is used on the .Net platform for domain specific software development.

Keywords: pipeline engineering, modeling languages, design principles, domain-specific modeling (dsm), model transformation.

I. Introduction

Domain concepts are representations of fundamental features inherent in specific fields of human endeavour. From these concepts models often referred to as the domain model, which characterize things in the domain can be derived. The description of concepts in this work was a domain analysis exercise, targeted at the salient technical characteristics prevalent in the domain of oil and gas pipeline engineering [18]. What happens is that pipeline components such as pipe cross sections, joints, fittings, and other pressure containing ones are produced with AutoCAD; these products usually referred to as graphics models now represent the pipeline components model from which the concepts for the language construction were derived [14]. It followed a precise path from specification of modelling primitives to formal feature models that moved into the formation of a language metamodel.

One purpose of a model in this circumstance is to reflect the control-flow of the design process without incorporating nonessential properties. To this end, the behaviour of meaningful design scenarios can be depicted in a metamodel [2]. In order to effectively incorporate stakeholders design intents and to ease the modelling processes, the domain specific modelling (DSM) approach was adopted. The DSM approach sees the model as the core entity throughout development and is basically a platform for language development. A language is therefore designed to specify the model. The language description entails flexibility, so that the pipeline context model can be applied productively[1].

In addition to providing a design framework for correctly fixing the application of the pipeline context model, modelling allows the pipeline systems designers to explore many different designs before representation. It is observed that computer aided design (CAD) software such as AutoCAD are indispensable tools in the pipeline engineering work environment, but most pipeline engineers find it worrisome to learn, understand and use conventional computer aided design (CAD) software in their line of business [3]. Modeling with AutoCAD for example has been ccomplex processes that are too costly to actually implement and refine. Modeling in a domain specific modeling system allows the modeler to easily modify the process and determine if the changes are effective.

The advocated shift in the design environment is domain specific modeling, which resolves many of the problems inherent in the protocol based GPL/ CAD systems design standard. In this approach, the metamodeling mechanics allows the stakeholder to determine the intents on an interface with very familiar notations, which means the design complexity is drastically reduced and control transferred from the complex CAD system to the domain expert. This allows the pipeline engineer to simply input familiar notations *(i.e. pipeline engineering concepts that are very familiar to them e.g. pipe diameter, fittings dimensioning, flow metrics etc.)* on an interface to get the kind of design, simulation artifact and other pipeline configurations without having to use any CAD or related system [5]. Domain specific modeling involves the logical use of models as core entities throughout development; it is

Author α : Dept. of Mathematics/Computer Science, Faculty of Science, Niger Delta University, Nigeria.
e-mail: rb.japheth@ndu.edu.ng
Author σ : Department of Computer Science, Faculty of Physical Sciences and Information Technology, University of Port Harcourt, Nigeria asagba. e-mail: prince@uniport.edu.ng

simply a domain specific modelling language whose type systems and semantics will formalize the structure, behaviour and requirements within the domain of oil and gas transmission pipeline engineering. The transformations from the AutoCAD objects to the language formalism are typically designed starting with abstract concepts and are iteratively refined into detailed descriptions. Therefore, the language needs to reflect this *transmission pipelines* development cycle, and can still provide valuable information about the process at every level of abstraction [9].

II. Related Work

A very recent language formalism implemented by Phillip et al. [12] is a methodology addressing issues surrounding a scheme for modeling, scalability and accessibility to modeling and verification processes for practitioners within the railway domain. Their work introduced a methodology for developing domain specific languages for modeling and verification that aims to aid in the uptake of formal methods within industry. It also concretely illustrates the success of this methodology for the railway domain. This present work has acknowledged the design methodology and the specification patterns of the domain specific language for the application in the Railway industry as presented by Philip et al [12]. In our approach we have made efforts to move away from the use of the Generic Modelling Environment (GME) suite for specifying modelling concepts. The challenge in the UML paradigms is the lack of a semantic definition within the context of the metamodel. This problem has negative impact on reusability of DSMLs, because a well-made DSML captures the concepts, relationships, integrity constraints, and semantics of the application domain and allows users to program declaratively through model construction. Incorporated in our metamodel is a semantic module to alleviate this challenge.

Milan et al. [11] discuss a method for designing modelling languages by presenting a platform independent model (PIM) for information systems (ISs). The concepts are described by Meta Object Facility (MOF) specification, one of the commonly used approaches for describing meta-models. One of the main reasons for this technique is to specify the concepts through the meta-model, as well as a domain analysis purposed at creating a domain specific language to support IS design. As such, it complements our technique, which is a top down approach. Similarly, Christian Hahn and Klaus Fischer [13] presented a UML based domain specific modelling language for multi-agent systems (DSML4MAS), in their approach the language semantics are restricted only to the definition of concepts and their relationships within the metamodel. UML is not an end user representation language, and so domain specificity couldn't possibly be better represented then our approach. The focus of Jonathan Sprinkle et al. [10] research uses endogenous refinements approach to analysing models on a shred metamodel with only evolutionary changes. Starting with a set of rules, model transformation was automated between the source and the target environments all in the same problem space. Conceptually, this work is closely related to ours, but we transformed a seemingly graphical domain model to a textual application model for user interaction.

III. The language Design Considerations

The consideration is modelling pipeline design including pipe sections joined with fittings and other supports features such as flanges, bolting, gaskets, valves, hangers and the pressure containing portions of pipeline components [7]. A pipeline design dedicated for transmission of oil and gas from wells to tanks for storage or to refineries for processing. The pipe sections joined with fittings etc. are here referred to as the pipeline model; they are graphics models, solid objects aggregated from primitives of AutoCAD that depict the typical pipeline fundamentals, materials and joints in situ that forms the instances of the language creation [18] .

a) *Capturing the aspects of design*

Domain specific modeling of solid objects such as oil and gas pipeline components comes in different forms. Although there are many different ways to modelling, very common steps that capture the aspects of designing a modelling language that exemplifies stakeholders design intents in the domain of oil and gas pipeline engineering are presented below. The identified ones are:

- *Effortlessness:* the design aspect has to capture metrics that can enable a non-programmer or a non-technical domain expert model a pipeline design without necessarily writing lines of codes
- *Tractability:* the language design should capture applicability tailored to stakeholders design intents and view points
- *Reflectiveness:* the language should be able to accurately reflect a pipeline design scenario in order to correctly represent useful artefacts i.e. the language should be able to evolve products that can reflect oil and gas pipeline design artefacts
- *Passability:* the language design has to capture the aspects of symbolizing the actual execution of a pipeline transmission process

These steps are clearly stated in the language design specifically to achieve significant functionality

These steps are clearly stated in the language design specifically to achieve significant functionality during implementation [6]. In conventional engineering design modeling, objects are explicitly described, for this reason, when one aspect of the model is changed; often several changes have to be made to satisfy design intent or the implicit rules of the design. All these changes have to be made because the software [19] does not keep track of the rules and the modeler must decide where and when they are broken. In AutoCAD, for example models are created in a conventional way. AutoCAD, however, comes with more than one programming environment for creating a set of instructions, including the rules and constraints of the design as well as parameters defining certain aspects of the design, which can be used to build a model [7]. These instructions can be used to build the model from scratch, each time using the same parameters, or experimenting with different ones. The parameters can be numeric values, relationships, and can even include graphic parameters already existing in the model (e.g., a building lot, angular pipes, etc.).

The programming environment makes it possible to define variables [18]. It also allows conditional branching to different sets of instructions in the program and can repeat the instructions until a condition in the program or model is met. This capability of defining solid behaviours through variables fosters model interaction in such a way that transfer of information is only possible within the set conditions in the CAD system [5].

One basic consideration and challenge is the issue of interaction between models, interactions in the way of concepts devoid of possible parametric constraints within a CAD system [20]. Interactions that can produce other complete models with noticeable properties relative to a given set of concerns in relevant domains that captures accurately and concisely all of its interpretation and design intent for specific problems and solutions. This has not been achieved with current CAD systems, and coupled with the third generation programming APIs inherent in them, they still lack sufficient linguistic power to handle domain and platform complexities and hasn't moved speedily with domain technologies [19]. Model interactions that creates new objects that encapsulates and relates the details pertinent to the viewpoint of domain experts is still lacking in current CAD/GPL modeling systems.

This constrains the expressiveness of the modeling systems, and the primary concern with this limitation is that it is a limitation imposed by the systems internal construction and technologies. Additionally, how the designs will be created depends on the underlying APIs and how the design will execute once compiled. In domain specific modeling, the modeler may want to experiment with familiar domain notations to obtain feedback. Therefore, a new language design is needed that focuses on and represents the concepts of domain models rather than relying on CAD systems and programming languages [9]. The believe is that such software development efforts will enable stakeholders to cope with platform complexities, it will also be cost effective, save time, and raise productivity levels [8].

b) *The Methodology*

The approach is hinged on examining the requirements of a modeling language for the oil and gas transmission pipeline domain. The requirements criteria are based on getting the pipeline models from AutoCAD and making them to represent things in the pipeline engineering domain. The aim is to take away the design and programming complexities associated with any CAD/GPL systems. The expectation of adopting this methodology is a pipeline systems modeling language (PSML) [2], which fundamentally, should support pipeline engineering concepts rather than relying on function calls and method invocations inherent in programming languages. There are quite a number of implications to this design methodology: the language is user friendly, showcases concrete syntax of domain notations that makes it more attractive to domain experts without programming expertise. Another implication is that the context free grammar is recursively defined to capture only oil and gas pipeline physical components configurations and constraints [6].

The syntax and semantic definitions of the language were clearly defined to exemplify our approach. The semantics are precisely defined and specified as denotational units to capture concurrency, and communication abstractions of the features of the pipeline product family. PSML incorporates a language construct called a *translator*, which is a process oriented specification that computes the resource request tendencies from the application model, which allows the stakeholder to evolve designs according to the defined viewpoints. In the core of the grammar is the vocabulary of components and associated attributes and values, which are transferred into an instruction sequence corresponding to any particular feature model as the modelling element. The translator does the transfer through a translation scheme based on syntax directed translation. The attributes such as angle, units, length, and size from the vocabulary of components keeps track of the resulting design object once a request triggered by stakeholders design intent is made into the system [17]. To achieve this possibility, the non-terminals such as fitting type (flange-ft.) and type name (elbow-T) etc. are marked with the attributes-angle, units, length, and size, and value points(x, y, z),and must be available when referenced within the instruction sequence of the context free grammar (CFG). The translation scheme which serves as the translation engine now enables the processing of these modelling elements into new artefacts [16]. In the operational

sequence (i.e. integrating the semantic elements) of the translation scheme, the grammar symbols associated with attributes in the CFG are rendered semantic actions inserted within right sides of the productions [2], so from each non-terminal, a value function that has a formal primitive parameter for each inherited attribute is made. The values are then returned to complete translation with the correct tokens specified.

c) The Language Rudiments

The predominant factor of an engineering design process is a task on the interactive aggregation of graphics primitives, graphics assemblies and subassemblies of CAD systems, which can be used interchangeably to produce solid model. With this language it is simply a modeling action, the PSML syntax for a modeling action is:

$$modeling\ identifier\ \{\dots\dots\dots\dots\dots\}$$

The resultant language rubrics are simply defining notations for the concrete syntax. The possible representations of the model are denotational semantic algebra as follows:

$$concrete\ syntax\ semantic\ algebra$$
$$\{$$
$$modeling\ define\ \{\}$$
$$modeling\ command\ \{\ \}$$
$$modeling\ design\ \{\ \}$$
$$modeling\ test\ \{\ \}$$
$$\}$$

Though presented here is not enough details about this high level descriptions, it provides information about what steps need to be completed and the order in which they should be performed in order to trigger a modeling action.

IV. Modeling Primitives

The modeling primitives are the resources to creating a pipeline model that creates the platform for tackling the complexity of CAD systems being unable to express domain concepts effectively. The ability to express domain concepts effectively allows the domain expert to recreate a variety of interdependencies that occur within a modelling process. The language logic allows modeling actions to require and provide resources, which typifies the need for the production of a transmission pipeline model. Using the *option* constructs, valuations can be initiated to provide more optional and variable entities for a particular modeling action. The optional entities are functions defined recursively over abstract syntax arguments that do denote unique scenarios as follows:

$$modeling\ define$$
$$\{$$
$$option\ \{function\ \}$$
$$provides\ \{statements\ \&\&attributes\ \}$$
$$option\ \{end\ \}$$
$$\}$$

Some conditions must be met for the modeling definitions to be precise, the statements provided ensures that the definition standards are correctly put in place. Now the *option* action for the valuation functions cannot be possible unless the statements and the pipeline components attributes are available for processing. Using these primitives, a stakeholder can initiate interdependencies that could exist within a pipeline design by specifying aspects of its functional quality. Though the syntax is the pipeline domain organizational structure with the semantics indicating the configuration constraints such as attributes, relationship, interdependencies, and changes in system states due to compositions and domain-specific pipeline domain operational rules; specific qualities of attribute resource are essential in keeping track of domain specific relevant information [17]. The information is tagged with the pipeline component attribute values, so that in the end the vocabulary can be transferred as attributes into the instruction sequence in the language construct. The set of semantic rules and attributes (*A*) associated with each grammar symbol; value types such as string, real, and arity, and terminals are all assigned functional dependencies. The attributes are provided to describe the state of a resource and thus it would be clearer to state attributes as follows:

$$Attributes\ \{component.length.angle.thickness.size == 'complete\ design'$$

Attributes not only describe the state and specific qualities but also provide a means to describe changes to resources. They also provide some control over the translations from what the domain specific language does and what is carried out in real life. Through the attributes the vocabulary bridges the semantic gap between writing lines of code and design intents of stakeholders; this is made possible by raising abstraction levels of the problem domain and mapping these abstraction levels to appropriate concepts in the application domain. The statements are essentially necessary steps towards describing the state of the attribute of a resource in the application environment with the correct state after execution.

V. The Design Paradigm

There are some mechanisms put in place for describing the operational mode and control of a design. These mechanisms, which reflect the constructs for the operations designate the system flow in designing a pipeline model.

a) The CFG Instruction Sequence

The instruction sequence is the context free grammar in BNF notation. It is the repository of

specifications that guides the fundamental flow of instructions in the systems internal mechanism. The CFG design specifications are as follows:

$CFG\ instruction\ sequence$
{
$\boldsymbol{modeling}\ type\ name\{pipe-type\ \&\&\ fitting-type\ \&\&\ joint-type\ \&\&\ instrument-type\ \&\&\ support-type$
$\boldsymbol{modeling}\ \ attributes\ \{length\ \&\&\ size\ \&\&\ angle\ \&\&\ thickness\}$
{
$\boldsymbol{modeling}\ units\{double\}$
}

To start and complete a pipeline design, the CFG instruction sequence has to be accomplished in the internal mechanism to attain valid demonstration.

b) Repetitive tasks

This design step takes into consideration some conditions that occur quite frequently within the design process. A necessary condition is the repeating of certain vital steps whenever a particular design scenario returns. Following the functionality in the instruction sequence, the syntax for the iteration is specified to determine what repetitions need to be evaluated and affected:

$Repetitive\ tasks$
{
$\boldsymbol{modeling}\ type\ name$
{
$pt\{model == pt.val\}$
$ft\{model == pt.val\}$
$jt\{model == pt.val\}$
$st\{model == pt.val\}$
$it\{model == pt.val\}$
}
$\boldsymbol{modeling}\ attributes$
{
$pt\{add.type == id.token\}$
}
$\boldsymbol{modeling}\ units\ \{add.type == id.double\}$
}
$\boldsymbol{modeling}\ execute\ \{\ \}$

Although there are many conditions in the modelling process that are based on human judgment, when determining the path to take in PSML, the primitives of the first modelling action actually allow the process to be more dynamic by providing multiple options.

c) The Decision repository

This is the store house of the specifications of the semantic domain and its operations as depicted in the semantic algebra. The pipeline is the root concept, meaning that it is the target result of all the underlying interdependencies of the components interactions. The decision procedure for determining which path to take clearly describes the structure of the oil and gas pipeline domain and how its elements are used by the functions, which makes it easier to analyse the semantic definition concept by concept. The specifications are as follows:

$Pipeline\ root\ concept$

$\boldsymbol{modeling}\ first\ selection\ \{build\ pipeline == build\}$
$\boldsymbol{modeling}\ second\ selection\ \{size == dimension\}$
$\boldsymbol{modeling}\ third\ selection\ \{builder\ *\}$
}

In line with the earlier showcased modelling primitive's description, the processing of these primitives to artefacts is depended on the pipeline design configurations.

d) Traceability

This operational construct specifies a set of parallel actions within a pipeline build process:

$Traceability\ \{$
$\boldsymbol{modeling}\ build\ pipe$
{
$pipe == (); (double)$
$fitting == (); (double)$
$joint == (); (double)$
$support == (); (double)$
$instrument == (); (double)$
$angle == (); (double)$
$size == (); (double)$
$size == (); (diameter-inner\ (float))$
$size == (); (diameter-outer\ (float))$
$thickness == (); (int)$

$units == (); (double)$
$points\ x, y, z == (); (int)$
$pipe.join.method.thispipe.p(0.1)$
}
}

Parallelism is employed here generally to allow for the performance of the embedded actions that pertain to model execution. At this point, the language interpreter decides the path because the dynamic nature of the pipeline build processes does not adhere to the strict nature of programming languages.

VI. Progressive Language Features

The semantic module in this instance is an abstraction that describes the semantics, the syntax, the necessary parsing dynamics and the resultant abstract

syntax tree. What this means is that the language metamodel reflects the problem domain abstractions; incorporating domain concepts and associated rules in a detailed denotational semantic algebra presented in figure 1 to provide for better translation interpretation [22].

$$New \rightarrow name: String \rightarrow pipeSlope\ record$$
$$New \rightarrow name\ (name) = (name, fittings, parameter)$$
$$Move-to-position: Pipe-rec \rightarrow Pipe-rec$$
$$Move-to-direction\ (parameter) = (parameter \downarrow named\ quantity),$$
$$\ldots\ldots\ldots\ldots$$
$$Compute-pipeFlow: Pipe-rec \rightarrow fittings$$
$$Compute-pipeFlow: Pipe-rec \rightarrow joints$$
$$Compute-pipeFlow: Pipe-rec \rightarrow supports$$
$$Compute-pipeFlow: Pipe-rec \rightarrow instruments$$
$$\ldots\ldots\ldots\ldots$$
$$Compute-position\ (pipe-rec) = (cases\ (Pipe-rec \downarrow named\ quantity)$$
$$Compute-direction\ (parameter) = (cases\ (parameter \downarrow named\ quantity)$$

Figure 1 : Semantic Algebra of Resources

The editor defines the concrete syntax and creates interactive notations the end user will utilize to build his model. The target code layer is the rule processing module and code generator that enforces the rules defined in the metamodel. Bringing together all these modules into a unified modelling infrastructure covers the scope of the new system. Three collaborative sub-systems that can make the artefact orientation very feasible are suggested [23, 26]. The first phase is the domain model, which captures the metrics of the pipeline engineering field. The second layer is the user interface or application model that enables stakeholder interaction with the system and then a solution model that integrates the parsing mechanism for production of desired designs. As far as experts could see through to a design scenario, the system will be able to capture it and evolve a design that meets their needs. The user could make some input through guided notations from the interface, and the system can then match these inputs with a parsing grammar to produce desired designs. Internal communication among these phases is enforced and can be made possible by utilizing the .NET CLR Object Serialization system function tool set [64]. This denotationl definition of pipeline resources capturing the repository for the concepts of the language vocabulary, the domain abstractions and semantics, can allow users to perceive themselves as working directly with domain concepts [17].

VII. Discussion of Results

The tool is designed to translate a domain model which represents the relationships and classes of the core features of the application domain into a text template; resulting to the user interface environment [21]. The procedure for mapping to the text template is relatively through an object binder that specifies the event states.

a) The Domain Model

The domain model comprises the pipeline atomic and composite features [21]. The language encompasses in its domain model sound underlying pipeline engineering principles pertaining to the language keywords (see fig. 2), and how they are linked to produce a total life cycle approach to pipeline systems design and operation.

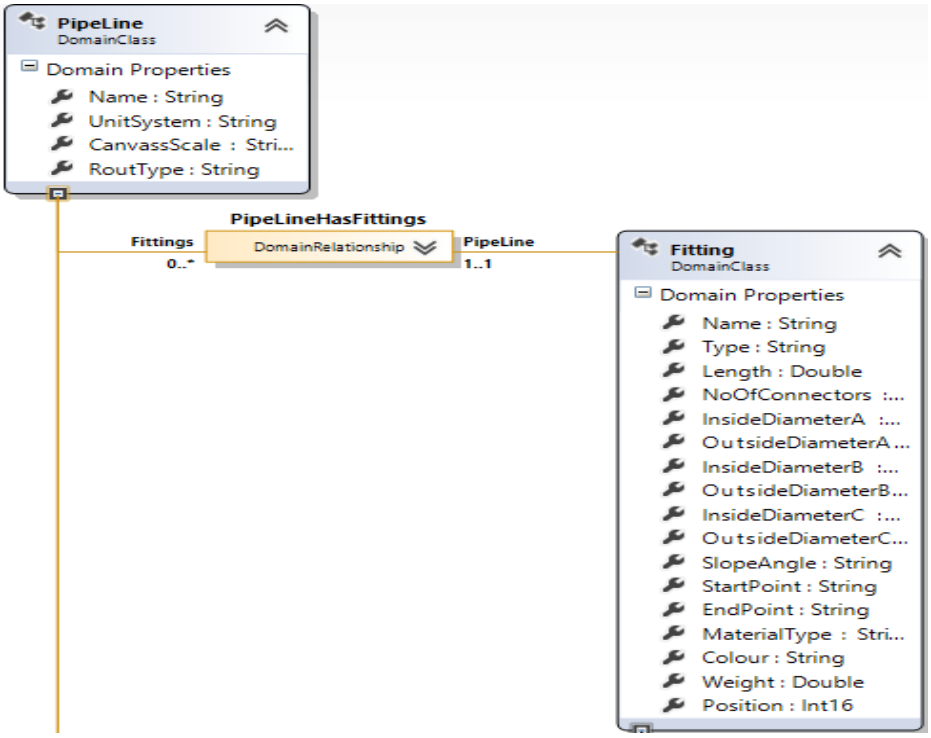

Figure 2 : Pipeline Engineering Principles

Figure 3 is showing how the language resources are related to produce a total pipeline system design operation. These relationships captured as all the semantic behaviours of the essential components and attributes, are the user centred composition rules of the semantic model comprising the events handler.

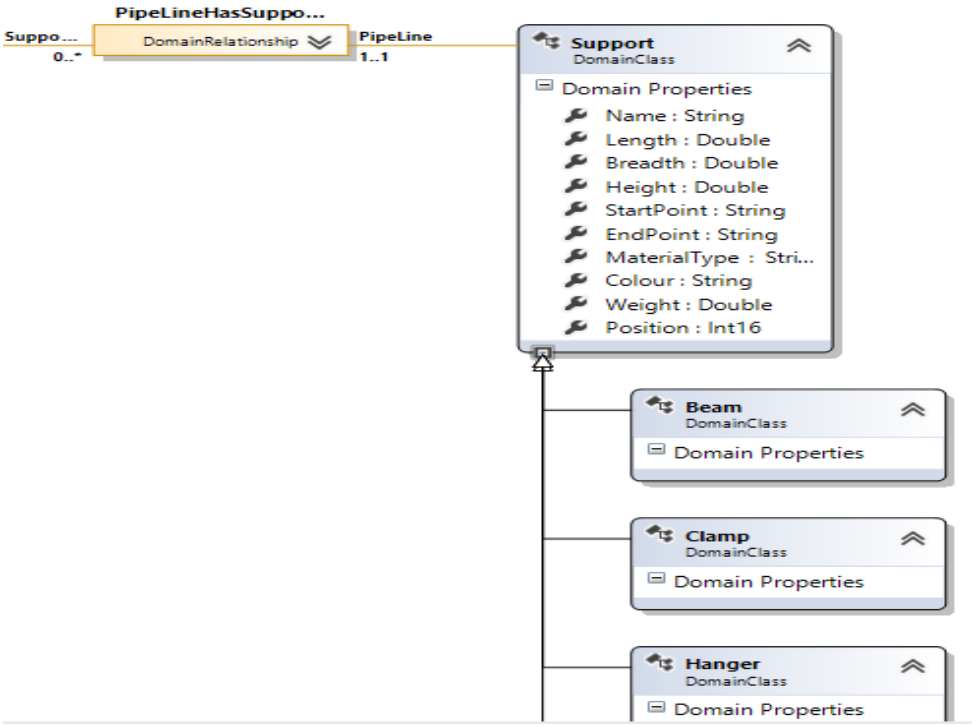

Figure 3 : Domain Model Relationships

b) Data Binding

In order to evaluate these semantic behaviours as the overall performance of the modelling system, a text template transformation is automatically performed via data binding. The data binding process is automated to be an object binder from the .net platform that specifies the event states. The events become more vivid as text inputs from the UI, what happens is that the components container binds the data source from the internal representations to the PSML model. Shown in figure 4 is the code snippet for the data binding action that results in the UI.

```
partial class PipeLineControl
    {
        public IContainer Components { get { return components; } }

        /// <summary>Binds the WinForms data source to the DSL model.
        /// </summary>
        /// <param name="modelRoot">The root element of the model.</param>
        public void DataBind(ModelElement modelRoot)
        {
            WinFormsDataBindingHelper.PreInitializeDataSources(this);
            this.pipeLineBindingSource.DataSource = modelRoot;
            WinFormsDataBindingHelper.InitializeDataSources(this);
        }
    }
}
```

Figure 4 : Object Binder

The user interface or application model in figure 5 is the layer that enables stakeholders' interaction with the system. As far as experts could see through to a design scenario, the system will be able to capture it and evolve a design that meets their needs. The user could make some input through guided notations from the interface, and the system can then match these inputs with a parsing grammar following internal communication among the application model, the domain model and the translator.

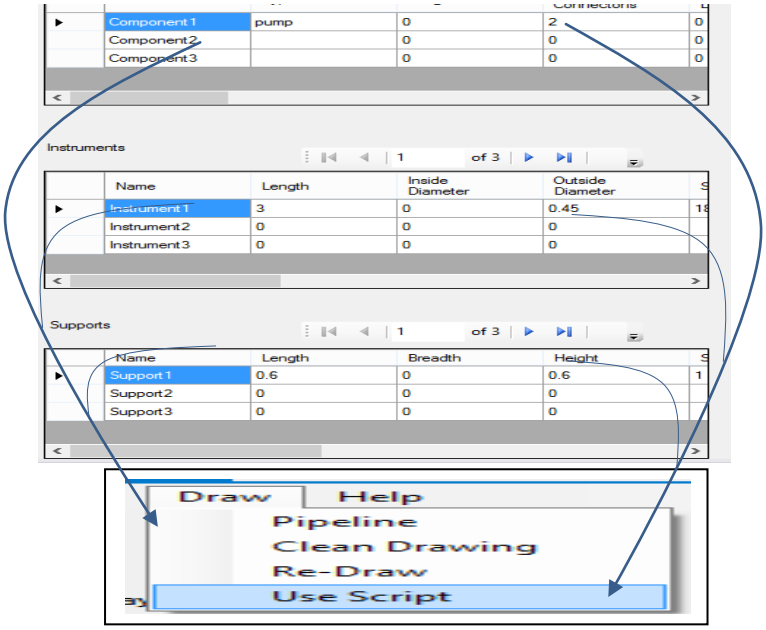

Figure 5 : Modelling Action with User Interface

VIII. Conclusion and Future Work

A modelling idea based on domain specific modelling is presented with the intention of highlighting the essential components of pipelines designed to transport oil and gas from source to destination. We utilized this domain specific modelling philosophy as a framework for designing a domain specific language for

modelling transmission pipeline designs. The language has the expressive capability to model pipeline designs at abstract and concrete levels of specification. This language has a number of features such as application model with familiar notations that allows flexible development and specification. However, the significance of constructing this new language is lack of tool support and modeling for the purpose of tackling complexities associated with computer aided design systems and general purpose programming language platforms for modelling engineering designs such s transmission pipeline systems. To provide support for the language, we tested the implementation of the application model through a text template transformation of the domain model of the language metamodel. The testing of the language tool was based on the .Net platform for domain specific software development. In the future the focus will be on the strategies for implementation of the integration of the editor and the grammar, which will lead to the actual writing of virtual pipelines.

References Références Referencias

1. Richard F. Paige, Jonathan S. Ostroff , Phillip J. Brooke. Principles for Modeling Language Design *Department of Computer Science, York University, 4700 Keele St., Toronto, Ontario* Supported by the National Sciences and Engineering Research Council of Canada. Preprint submitted to Elsevier Preprint 28 February 2000.
2. Darren C. Atkinson, Daniel C. Weeks, and John Noll. The Design of Evolutionary Process Modeling Languages Department of Computer Engineering Santa Clara University Santa Clara, CA 95053-0566 USA, *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC-2004), November 30–December 3, 2004, Busan, Korea.*
3. Andrade S.F.A. (2011) Asymptotic Model of the 3D Flow in a Progressing-Cavity Pump SPE Journal Volume 16, Number 2, June 2011 p 451-462
4. F. Klar, A. K¨onigs, and A. Schurr, "Model Transformation in the Large," in Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, ser. N.York: ACM Press, 2007, pp. 285–294.
5. Alessandro NADDEO, CAD Active Models: An Innovative Method in Assembly Environment, Journal of Industrial Design and Engineering Graphics Volume 5 Issue No. 1 – 2010
6. Franklyn Turbak and David Gifford, (2008), Design Concepts in Programming Languages The MIT Press Cambridge, Massachusetts London, England
7. Germanischer Lloyd (2008) Energy Solutions Pipeline Management Solutions Industrial Services GmbH Oil and Gas Steinhöft 9 20459 Hamburg, Germany glis@gl-group.com www.gl-group.com/glis Hans Vangheluwe (2010), (Domain-Specific) Modelling Language Engineering, Lisboa, Portugal
8. I. Porres, "Rule-based update transformations and their application to model refactorings," Software and Systems Modeling, vol. 4, no. 5, pp. 368–385, 2005.
9. Angel Roman & Bruce Trask (2011), Applying Model Driven Technologies in the Creation of Domain Specific Modelling Languages, *Proceedings of 14th International Conference on Model Driven Engineering Languages and Systems Wellington New Zealand*
10. Jonathan Sprinkle, Jeff Gray, and Marjan Mernik Fundamental Limitations in Domain-Specific Modeling Language Evolution University Of Arizona, Ece, Technical Report #Tr-090831 1
11. Milan Čeliković, Ivan Luković, Slavica Aleksić, and Vladimir Ivančević, A MOF based Meta-Model and a Concrete DSL Syntax of IIS*Case PIM Concept ComSIS Vol. 9, No. 3, Special Issue, 1076 September 2012
12. Philip. J, M. Roggenbach, Encapsulating formal methods within domain specific languages: A solution for verifying railway scheme plans, Mathematics in Computer Science 8 (1) (2014) 11-38.
13. Christian Hahn and Klaus Fischer, Domain Specific Modeling Language for Multiagent Systems German Research Institute for Artificial Intelligence (DFKI); Springer-Verlag Berlin Heidelberg 2009
14. Autodesk Inc. (2013) *AutoCAD Release 2013 Programmers Reference Manual*
15. Anders Eriksson and Andler Jeff Offutt (2012), Model Transformation Impact on Test Artifacts: An Empirical Study, *Proceedings of ACM Conference,* MoDeVVa', Innsbruck, Austria
16. S. M. Sutton, Jr., D. Heimbinger, and L. J. Osterweil. APPL/ A: A language for software-process programming. *ACM Trans. Softw. Eng. Methodol.*, 4(3):221–286, July 1995.
17. S. M. Sutton, Jr. and L. J. Osterweil. The design of a next generation process language. In *Proc. 6th Euro. Softw. Eng. Conf. and 5th ACM Symp. on Found. Softw. Eng.*, pages 142– 158, Zurich, Switzerland, Sept. 1997.
18. *B.G. Technical* LTD - *B.G. Technical* Oil & Gas industry Port Harcourt, *Nigeria; www.bgtechnical .com/* Annual Reports 2009 to 2013
19. Kaskil, D.J. W. Buxton, D.R. Ferguson, Ten CAD challenges, IEE Computer Graphics and Applications 25 (2) (2005) 81-92.
20. Neil C. Katz, Skidmore, Owings & Merrill, LLP *(2007)*, Parametric Modeling in AutoCAD, AEC bytes Viewpoint Issue #32

21. Kyo C. Kang, Sholom G. Cohen, James A. H. William E (1990), Feature-Oriented Domain Analysis (FODA) Feasibility Study, *Technology* Technical Report CMU/SEI-90-TR-21 ESD-90-TR-222
22. David A. Schmidt, (1997), Denotational Semantics: A methodology for language development *Department of Computing and Information Sciences, 234 Nichols Hall, Kansas State University, Manhattan, KS 66506 schmidt@cis.ksu.edu*