

Modeling Methodology for Application Development in Petroleum Industry

Cong Zhang, Viktor Prasanna,* Abdollah Orangi,† Will Da Sie,‡ Aditya Kwatra,§

Abstract

The development of applications for monitoring, control, simulation and diagnosis in the petroleum industry involves a multitude of complex software tools. These tools have their own formalisms, semantics and use different abstractions to represent the system under development. They use different data formats to represent data in the software tools. Each application requires coupling of two or more different such complex software tools. Providing efficient interaction between these complex software tools using different abstractions, formalisms, data formats, etc. becomes a mammoth task. Thus there is a need to provide a unified environment that allows capturing the desired application and provide a framework for interaction between the necessary software tools. This paper discusses the formal metamodels to describe the individual formalisms in the desired unified environment. These metamodels, created by the Generic Modeling Environment (GME), define the domain-specific modeling language for application development in the petroleum industry.

1 Introduction

For the past few decades, there has been a constant effort to maximize the oil production using various complex simulation tools. There has been a need for providing an efficient decision support system, which requires proper interaction between these complex simulation tools. Integrated Asset Modeling (IAM) is the technique used to model different assets (*physical*: wells, blocks, etc.; *non physical*: control strategies, optimizers, etc) to provide efficient management between the assets. The petroleum industry has recognized the opportunity for IAM to address multiple challenges.

IAM presents an intensive operational environment involving continuous series of decisions based on multiple criteria including safety, environmental policy, component reliability, efficient capital, operating expenditures, and revenue. Asset management decisions require interactions among multiple domain experts, each capable of running detailed technical analysis on highly specialized and often compute intensive applications. These technical analysis executed in parallel domains over extended periods can result in divergence of assumptions regarding boundary conditions between domains. A good example of this is pre-development facilities design while reservoir modeling and performance forecasting evaluations progress. Alternatively, many established proxy (or reduced form engineering) models are incorporated to meet demands of rapid decision making in an operational environment or when data is limited or unavailable. The delivery of enriched information that results from these technical analysis into real time operational domains is another challenge addressed by IAM [5]. A consequence of these previous conditions is the additional demand for rapid delivery of relevant data to these applications at the desired frequency and/or density, synchronized in time over multiple sources.

Large volumes of data from multiple sources result from progressively improving new capabilities for well measurement, seismic data acquisition, and continuous data collection. The systems are used to monitor and control an asset component in the petroleum industry to gather and analyse data in real time. IAM ensures proper coordination between data collection sources and data processing destinations.

The asset management toolkit needs to be configurable and custom fit for purpose to handle a great diversity of needs over a large portfolio of assets that range from big to small, low cost-low volume to major capital-high volume, onshore-offshore, brownfield (mature) to greenfield (new developments). It is also desired to implicitly couple technical compute applications to run as one distributed system while maintaining component ownership within the respective domains. A need for optimization of the asset that encompasses all system components and potentially involves multiple nested optimization loops and sequences of actions within a control strategy also forms an important challenge addressed by IAM.

*EE-Systems, USC, Los Angeles, California 90089. Email: {congzhan, prasanna}@usc.edu

†Department of Petroleum Engineering, USC, Los Angeles, California 90089. Email: orangi@usc.edu

‡ChevronTexaco, San Ramon, California 94583. Email: Will.DaSie@chevrontexaco.com

§EE-Systems, USC, Los Angeles, California 90089. Email: kwatra@usc.edu

The industry has made some progress towards more integrated approaches to managing and operating assets. Integrated operational workflow mapping to ensure efficient and aligned use of critical resources is often the first step [8]. Top-down modeling using simplified models or proxies to support flexible decision and uncertainty analysis systems has been applied [1, 4, 5, 13].

At a more fundamental modeling level, application developers have coupled reservoir, network, and process simulators to more accurately model physical interactions between system components with a view to system integrated optimization. The variety of possible combinations of simulation technologies requires developing flexible coupling schemes, and patterns of interaction between software components.

The tight integration of the software and its physical environment has profound impact on the software technology to be applied [6]. With *conventional* software development techniques, changing the software in response to the evolution of a field is expensive in terms of both time and effort. In most cases, it is much like developing a new software, which consists of a number of phases: (1) defining what the system should do (system specification), (2) choosing a particular solution to meet the system specifications (design), (3) actually putting together the pieces of the system (implementation and integration), and (4) evaluating the new system to see if it actually meets the specifications (testing).

Instead of developing a new software from scratch for each application, our unified generic environment will help provide the required interaction between the existing softwares.

These applications are based on existing simulation software. Development of the application consists of three major steps: (1) Capturing a simulation scenario. (2) Constructing a software-based structural model to map the component tasks in the scenario to a software structure. This mapping is necessary because multiple simulators may exist for the same component task. (3) Determining inter-software coordination mechanisms, such as interface wrapping and low-level communication within the application.

This paper focuses on the first two steps in the application development. Our application development is based on Model-Integrated Computing (MIC) [12]. MIC employs domain-specific models to represent applications being designed. The models specify the desired application functionality and available simulation tools. The modeling language capturing the application functionality is based on finite state machine.

The paper is organized as follows. Section 2 describes the concepts of MIC, a tool that implements MIC, and related projects. The reasons for using MIC are described in section 3. The modeling language composed for the petroleum domain is presented in section 4. We conclude in

Section 5.

2 Background

Modeling has been widely used for software development. Many analysis and design techniques use models to describe the necessary class and inheritance relationships in the software. However, models created using these techniques are loosely coupled to the actual system development cycle. The concept of MIC [12] can be used to form a tightly coupled environment. The software is modeled along with the environment and integration constraints. In MIC, the key element to facilitate the design process is the *model*. Details about MIC-based development process can be found in [10].

The *Generic Modeling Environment* (GME) is a configurable graphical tool suite supporting MIC [7]. GME allows the designer to create domain-specific models. A metamodel (modeling paradigm) is a formal description of model construction semantics. Once the metamodel is specified by the user, it can be used to configure GME itself to present a modeling environment specific to the problem domain. Figure 1 shows the vision of our proposed framework for integrated asset management. The framework, which is based on MIC, will be used to facilitate activities in the integrated asset management, such as data integration, simulation application development, etc.

We have applied MIC to one of our projects, MILAN, the Model-based Integrated simuLAtioN framework. MILAN is a model based integrated simulation environment for embedded system design and optimization through the integration of various simulators and tools into a unified environment [9]. Using the MILAN environment, the designer formally models the target application, underlying hardware, and constraints (latency, throughput, energy dissipation, etc.) through a graphical interface provided by MILAN. The models are stored in a model database. Every target system is specified as a model. Model interpreters are the software components that translate the information captured in the models based on the input format required by the integrated tools and simulators. Tools currently integrated into the MILAN framework include functional simulators, such as Matlab, SystemC and ActiveHDL, and a high-level performance estimator, HiPerE.

MIC is currently being applied to CiSoft, which is a center being jointly developed by University of Southern California and ChevronTexaco Corporation for research and training on interactive smart oilfield technologies [3]. Several research areas and their interactions are being explored at CiSoft. These include integrated asset management, well productivity improvement, robotics and artificial intelligence, sensors, databases, among others. Integrated Asset Management deals with effective and efficient interaction

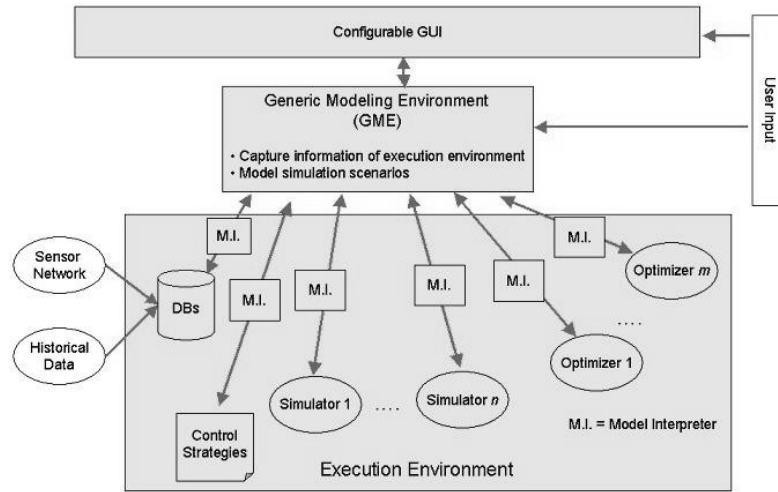


Figure 1. MIC-based framework for Integrated Asset Management.

among various assets improve the field management and thus production yields.

3 Model Integrated Computing

3.1 Why MIC

Models provide a more effective way to develop large, reliable, real-time software systems because they help manage complexity through abstractions and formal descriptions of the physical systems. Model-based software synthesis is a part of the larger discipline of knowledge-based software engineering. It integrates artificial intelligence and software engineering by supporting specification methods, software synthesis, and analysis with application-specific knowledge formalized into models.

MIC is a system software development approach that promotes the use of domain-specific models to represent relevant aspects of a system. The use of domain-specific models have many benefits. For example, they help users specify systems using domain concepts. Domain specific modeling also allows users to specify systems at a higher level of abstraction. One of the most important characteristics of MIC is that it allows us to dynamically resynthesize a running system without changing anything but the relevant components. MIC works well when software requirements and specifications are constantly changing.

4 Modeling Methodology

Modeling has been used to capture the system design, synthesize executable systems and perform analysis or drive simulation. Our models have been designed to capture the various simulation scenarios possible in the petroleum industry. A simulation scenario can be defined as the interaction in terms of event flows and relative ordering of such flows. To model such a scenario, we need to capture both the building components and the interaction between them. The building components include simulators, optimization tools, databases, etc. Since our target applications are based on existing software, we are not concerned with modeling the internal structures or implementation of the building software components. Instead we only capture their interfaces each of which can be characterized with a set of input signals and a set of output signals.

We have used a hierarchical approach to model the various components, which provide the required abstraction for the interfaces of the basic building components. The hierarchical model, provides the user with different levels of abstraction, levels of integration and levels of visualization. The topmost level provides a high level of abstraction and the models get more detailed as one goes to the lower levels. At the lowest level of integration the components are tightly coupled and provide explicit data exchange, whereas at the highest level the integration is limited to higher level components. Figure 2(a) shows the top level components in an application. The components can be classified into *physical* (well, block, process, pipe, network, separator, etc.) and

non-physical(control strategies, drilling schedule, reliability models, constraints, assumptions, etc.) entities.

Few of these entities are as described in the following sub-sections:

4.1 Physical component models

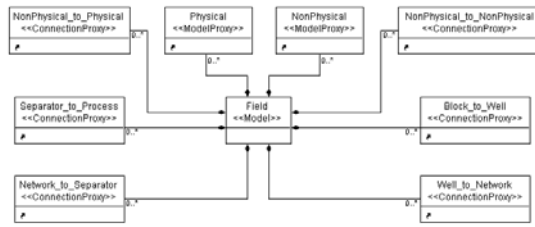
- **Block** With information in the geological model and reservoir simulator, a reservoir can be divided into several reservoir volume elements called the blocks. In our metamodel, each block is defined with the following parameters: *Original Oil In Place (OOIP)*, *primary decline curve*, *secondary decline curve*, *voidage target* or *recovery target*, among others.
- **Well** Wells are the source of precise information that we can obtain about the block in which the well exists. Our well model consists of various parameters and models to describe the physical structure and relationship with other operational components. Important parameters of a well include *on stream date*, *stimulation*, and *well capacity*. With on stream date, we can derive how long a well has been active. Through stimulation and well capacity parameters, a specific well design scheme can be applied. The well information model provides the information about where the well is physically located, which type of well (e.g., producer, water injection well, or gas injection well) it is, etc. Non-physical models, such as reliability model, assumption model, and real time data model, capture operational characteristics of a well.
- **Pipe network** A pipe network collects all fluid from all the wells at different locations and transfers it to the separators. Our pipe network model consists of three types of models: physical, data-related, and operational. *Joint*, *pipe*, and *choke* models describe how a pipe network is physically designed and constructed. Data-related models include *network real-time data*, *material balance*, *output stream* and *input stream*. Pressure and temperature distribution are two key parameters in the data-related models.
- **Separator** A separator is used to separate oil from gas and water. Generally, there are three stages for separation. Our separator model is used to represent the operations and components of a separator.
- **Process** After separation, the following processing actions are performed: product treatment, compressing gas to high pressure and sending it to market, chemical process, and fluid measurement.

4.2 Non-physical component models

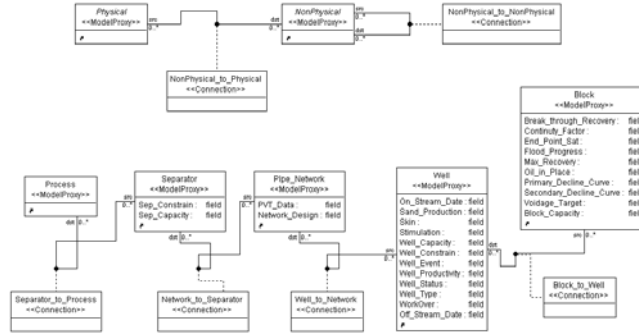
- **Assumption model** To simulate an asset, there are some assumptions involved. When we get new information from other sources, some assumptions may no longer be valid. Also, a model updating can create new results that directly affects other assets.
- **Drilling schedule** Drilling schedule model is a complement of the *on stream date* parameter in describing when a well will be active. Specifically, we define a drilling schedule with the following parameters: *measure depth*, *order priority*, *rig allocation*, *start date*, and *days to drill*.
- **Real time data** Real-time data model consists of three aspects: real-time production data, real-time access by an asset, and real-time action.
- **Control strategy** The control strategy model provides the heuristic operating rules and the optimization control for the system.
- **Reliability** Reliability models are used capture the reliability and availability of each asset and its building components.

4.3 Modeling component interaction

Once the components have been modeled, the next step is to model the interactions between different components. Modeling the interactions between the building components is not straightforward. Dataflow has been widely used in modeling many application behaviors, especially for signal processing applications [9]. Dataflow graphs can successfully represent unconditionally executed sequences of computation, and they can also represent iteration successfully in situations where the number of executions is known and independent of the data. However, two limitations of dataflow graphs make them unsuitable for modeling the interactions in a petroleum applications. Dataflow graphs can not represent (1) conditional execution or (2) data-dependent iteration. To overcome the limitations, many techniques are proposed to increase the expressive power of dataflow graphs. For example, a hybrid control flow/dataflow model can model a sequential mode of computation while allowing some freedom for re-ordering computation. The graphical programming language in [11] provides a similar capability. However, there are also disadvantages of the hybrid model [2]. In this paper, we propose to use finite state machine (FSM) to model the interactions. FSM is one of the commonly used techniques to specify the behavior of a system. In an FSM diagram, states are represented by labeled nodes and transitions by directed



(a) Modeling Paradigm to display hierarchical approach



(b) Relationship between components modeled with connections

Figure 2. Hierarchical Approach for designing modeling paradigms

edges. The labels on the edges indicate the events that trigger the transition, the actions generated by output, and potentially the actions performed on local variables. Our study on simulation scenarios in petroleum applications shows that FSM can model most of the scenarios. Furthermore, FSMs allow much more compact representation of system behavior than the control flow/dataflow hybrid models.

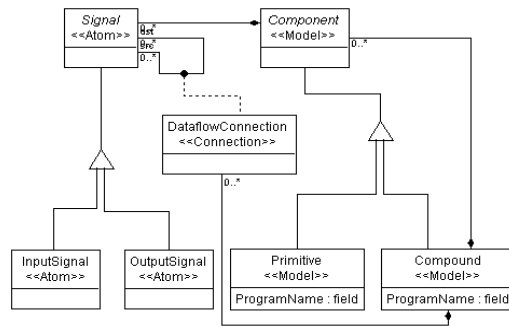
Our application models can formally describe scenarios represented by sequence diagrams as collections of *components*, which represent functional units in the scenario, and *connectors*, which represent the means of information exchange among the components. Figure 3 shows the metamodel using UML class diagram notation. *Component* is an abstract base class to capture common characteristics of the classes: *Primitive*, and *Compound*. Primitives and compounds can be used to model various software components.

Primitives are the leaf nodes in the hierarchy. To model the interface of a software component, each primitive is associated with several attributes specifying program information of the component. The attributes include program name, program parameters, etc. Primitives can contain *InputSignal* and *OutputSignal*, but not component. On the other hand, compounds are the composite objects and can contain components, i.e. primitives and compounds, creating a hierarchy of any depth. *Signal* is an abstract class to capture the input and output interfaces of components. They

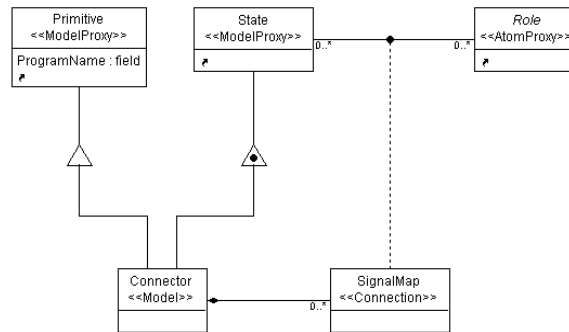
can be connected to form the data flow among the components.

As shown in the metamodel (see Figure 3(b)), the behavior of a *connector* can be modeled as a finite state machine. Its implementation can also be specified with the *ProgramName* attribute. A connector consists of a set of *roles* specifying the interface of this connector. The semantic relationship between a connector’s behavior specification and the role specifications is that the connector behavior specifies the coordination of the roles. Note the use of *proxies* for Primitive, State, and Role that refer to existing classes defined in different metamodel sheets. This is the preferred way of doing metamodel composition in GME. The original metamodels are unchanged and a new metamodel sheet is created where concepts from the original metamodels are referred to by class proxies. New concepts and connections can be introduced to the new metamodel sheet. The use of proxies is important in metamodeling for a large system. Separation of concerns can be achieved by having different people working on different metamodel sheets and using proxies to combine their work.

Application models constitute the highest level of abstraction. They capture application requirements with specific global functions. The behavioural interactions of those functions can be specified using a set of scenarios that define the desired interaction relationships between compo-



(a) Components



(b) Connectors

Figure 3. Metamodel for modeling applications

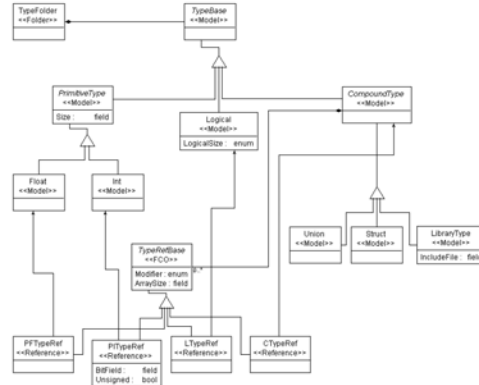
nents. Connectors are used to coordinate the participating components in an application. They specify the application specific glue that composes the component behaviors to achieve global functions. The components and connectors have signals and roles respectively. The binding between the signals and the roles is specified by model users.

4.4 Data type models

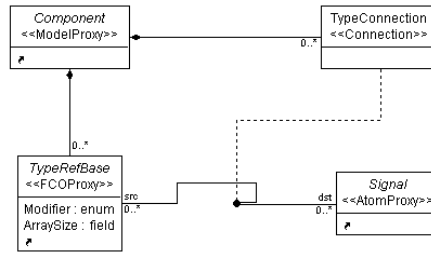
Data plays a key role in today's petroleum industry. Exploration and production geoscientists rely heavily on various data, such as maps, logs, seismic sections, well test reports, etc. Many issues related to data exist in petroleum industry. In a simulation application, data used by different software components may take different data format. Thus

feeding data from the output of one software to the input of another is not feasible. Furthermore, even the same data stored in a database could be interpreted differently by various software. In this paper, we are concerned with modeling the data flowing among various software components. The data type models described in this section represent a lower level of abstraction than the application models. They are attached to the application models, more accurately describing the data to/from each component in an application.

Data type models are used to capture the inputs and outputs of each processing component in an application model. Each signal of a component is associated with a data type. This can be a simple built-in type such as a double, float, integer or character or an array of any one of these, or a user-defined aggregate type. Aggregate types are explicitly



(a) Metamodel for modeling data type



(b) Composing data typing with application modeling language

Figure 4. Modeling data types

modeled by combining single (and arrays of) built-in types. This signal typing is easily specified within each component instance. The aggregate signal types include elements to allow for breaking signal into constituent parts or combining simple types into complex types. Merging several simple streams into a larger structure helps reduce higher level interconnection visual complexity if a logical grouping can be developed. The data type metamodel is shown by Figure 4(a).

The relations of the data types are also modeled. For example, if a given type needs to be converted to another type with a conversion function, a model capturing the conversion function has to be used between the two types.

The application models and the data type models are composed together according to the metamodel in Figure 4(b). The *TypeConnection* connection between component signals and the *TypeRefBase* class is introduced. *TypeRefBase* represents a reference to data type models. *Type-*

Connection assigns the referred type to the given port.

5 Conclusion

This paper has shown how a unified environment is created for developing a class of petroleum applications, which consist of various software components. The modeling paradigms reflect our current understanding of petroleum engineering domain. By adding data type models to application models, two-level abstraction has been achieved. To improve the modeling paradigms with more levels and more capabilities, we will have more interactions with domain experts. To facilitate the development of complex applications in petroleum industry, other modeling paradigms than what we have defined will be explored.

Acknowledgment

This research was (partly) funded by CiSoft, (Center for Interactive Smart Oilfield Technologies), a Center of Research Excellence & Academic Training and a joint venture between the University of Southern California & Chevron-Texaco.

References

- [1] S. Begg, R. Bratvold, and J. Campbell. The value of flexibility in managing uncertainty in oil and gas investments. In *SPE 77586, SPE Annual Technical Conference and Exhibition*, San Antonio, Texas, 29 September-2 October 2002.
- [2] J. T. Buck. *Scheduling Dynamic Dataflow Graphs With Bounded Memory Using The Token Flow Model*. PhD thesis, University of California at Berkeley, 1993.
- [3] CiSoft: Center for Interactive Smart Oilfield Technologies. <http://cisoft.usc.edu>.
- [4] A. Cullick, D. Heath, K. Narayanan, J. April, and J. Kelly. Optimizing multiple-field scheduling and production strategy with reduced risk. In *SPE 84239, SPE Annual Technical Conference and Exhibition*, Denver, Colorado, October 2003.
- [5] F. Floris and M. Peersmann. E&p decision support system for asset management - a case study. In *SPE 65146, SPE European Petroleum Conference*, Paris, France, October 2000.
- [6] J. Frederick P. Brooks. No silver bullet: essence and accidents of software engineering. *Computer*, 20(4):10-19, April 1987.
- [7] GME: Generic Modeling Environment. <http://www.isis.vanderbilt.edu/Projects/gme>.
- [8] P. Janele, T. Galvin, and M. Kisucky. Integrated asset management: Work process and data flow models. In *SPE 39712, SPE Asia Pacific Conference on Integrated Modelling for Asset Management*, Kuala Lumpur, Malaysia, March 1998.
- [9] MILAN: Model-based Integrated Simulation. <http://milan.usc.edu>.
- [10] G. Nordstrom, G. Karsai, M. Moore, T. Bapty, and J. Sztipanovits. Model integrated computing-based software design and evolution. In *Conference on Life Cycle Software Engineering Technology for Modern Avionics, Missiles, and Smart Weapon Systems*, Huntsville, Alabama, August 2000.
- [11] P. D. Stotts. The PFG language: Visual programming for concurrent computing. In *Proc. Int. Conf. on Parallel Programming*, volume 2, pages 72-79, 1988.
- [12] J. Sztipanovits and G. Karsai. Model-integrated computing. *Computer*, 30(4):110-112, April 1997.
- [13] G. Williams, M. Mansfield, D. MacDonald, and M. Bush. Development and applications of sustaining integrated asset modeling tool. In *SPE 89974, SPE Annual Technical Conference and Exhibition*, Houston, Texas, September 2004.