DATA EXCHANGE IN MULTI-DISCIPLINARY OPTIMIZATION FRAMEWORKS

A thesis presented to

the faculty of

the Fritz J. and Dolores H. Russ College of Engineering and Technology

of

Ohio University

In partial fulfillment

of the requirements for the degree

Master of Science

Arun N. Nambiar

March 2004
This thesis entitled
DATA EXCHANGE IN MULTI-DISCIPLINARY OPTIMIZATION FRAMEWORKS

BY
ARUN N. NAMBIAR

has been approved for
the School of Industrial and Manufacturing Systems Engineering
and the Russ College of Engineering and Technology by

Robert P. Judd
Cooper Industries Professor

R. Dennis Irwin
Dean, Fritz J. and Dolores H. Russ
College of Engineering and Technology
Abstract


Data Exchange in Multi-Disciplinary Design Optimization frameworks (91pp.)

Director of Thesis: Robert P. Judd.

An efficient and effective data exchange mechanism is indispensable for multi-disciplinary optimization frameworks that provide a common working environment to increase interdisciplinary interactions and reduces the design cycle time. Some of the existing data exchange mechanisms have been studied and a new data exchange mechanism has been implemented. The scope of this work has been to develop two tools namely the Vector tool and the Name-Value tool to handle data in the form of one-dimensional arrays and name-value pairs respectively. A data caching algorithm has also been implemented so that the two tools interoperate with the help of adapters that translate data across the two formats.

Approved: Robert P. Judd

Cooper Industries Professor
Acknowledgments

First of all, I would like to express my sincere thanks to my thesis advisor, Dr. Robert P. Judd. This research would not have been possible without his guidance and support. I would also like to thank all the members of my committee for all their help and assistance. I would also like to thank my family for having stood by me throughout the completion of this thesis.
# Table of Contents

Abstract.................................................................................................................................................................................. 3

Acknowledgments ........................................................................................................................................................................ 4

Table of Contents ........................................................................................................................................................................ 5

List of Figures............................................................................................................................................................................ 9

1 Introduction........................................................................................................................................................................... 10

1.1 Background........................................................................................................................................................................ 10

1.2 Scope of this work ............................................................................................................................................................. 12

1.3 Organization........................................................................................................................................................................ 12

2 Existing Frameworks............................................................................................................................................................. 14

2.1 ModelCenter ........................................................................................................................................................................ 15

2.2 AML.................................................................................................................................................................................... 17

2.3 iSIGHT.................................................................................................................................................................................. 18

2.4 FIDO.................................................................................................................................................................................... 18

2.5 FACET................................................................................................................................................................................ 19

2.6 OPTIMUS............................................................................................................................................................................ 20

2.7 DOME................................................................................................................................................................................ 21

2.8 CML.................................................................................................................................................................................... 22

2.9 IMAGE ................................................................................................................................................................................ 22

2.10 FIPER............................................................................................................................................................................... 24
2.11 Drawbacks of Existing Data Exchange Mechanisms .......................... 25

3 Data Exchanger .............................................................................................. 29

3.1 Exchanger .................................................................................................. 32
3.2 Tool ......................................................................................................... 33
  3.2.1 Vector ............................................................................................... 35
  3.2.2 Name-Value ..................................................................................... 36
  3.2.3 Partitioner ......................................................................................... 37
  3.2.4 Table ................................................................................................ 38
3.3 Region ........................................................................................................ 39
3.4 Tool Operator ........................................................................................... 40
3.5 Region Operator ....................................................................................... 41
3.6 Adapter ..................................................................................................... 42
3.7 FIPER Parameter .................................................................................... 44
3.8 Conceptual Example ................................................................................ 44

4 Vector .............................................................................................................. 48

4.1 Vector Details ............................................................................................ 48
  4.1.1 Tool Specifications ........................................................................ 48
  4.1.2 Delimiters ......................................................................................... 49
  4.1.3 Groupers .......................................................................................... 49
4.2 Region Operators (ROPs) ........................................................................ 50
  4.2.1 Element ............................................................................................ 50
4.2.2 SubVector........................................................................................................ 51
4.2.3 All..................................................................................................................... 52
4.3 Tool Operators (TOPs) ....................................................................................... 52
4.4 Important Methods............................................................................................. 53

5 Name Value............................................................................................................. 54
5.1 Name Value Details .......................................................................................... 54
5.1.1 Tool Specifications..................................................................................... 54
5.1.2 Delimiters..................................................................................................... 55
5.1.3 Groupers ...................................................................................................... 56
5.2 Region Operators (ROPs) ............................................................................... 57
5.3 Tool Operators (TOPs) ................................................................................... 58
5.4 Important Methods........................................................................................... 58

6 Adapters................................................................................................................ 61
6.1 TextToItem Adapter....................................................................................... 63
6.2 TextToOneD Adapter ..................................................................................... 64
6.3 ScalarToText Adapter..................................................................................... 64

7 Data Caching Mechanism .................................................................................... 66

8 Conclusion ............................................................................................................. 69

Bibliography .......................................................................................................... 70

Appendix A - A Practical Example of Data Exchanger using Vector.................... 74
Appendix B - A Practical Example of Data Exchanger using Name-value ............. 81

Appendix C - A Practical Example of Data Exchanger using Vector and Name-value ........................................................................................................ 86
List of Figures

Figure 1. Graphical Representation of data exchanger and data sources ......................... 29
Figure 2. Data Exchanger Components ............................................................................ 31
Figure 3. Sequential Access of data .................................................................................. 34
Figure 4. Random Access of data ..................................................................................... 35
Figure 5. A typical vector of elements .............................................................................. 36
Figure 6. An example of a typical name-value pair .......................................................... 37
Figure 7. A typical text region .......................................................................................... 38
Figure 8. A typical table region ........................................................................................ 39
Figure 9. Tool Operator .................................................................................................... 41
Figure 10. Region Operator .............................................................................................. 42
Figure 11. Adapters ........................................................................................................... 43
Figure 12. Partitioner tool on the Exchanger .................................................................... 45
Figure 13. Vector Tool in action ....................................................................................... 46
Figure 14. Name-Value Tool in action ............................................................................. 47
Figure 15 A Simple Processing System ............................................................................ 86
1 Introduction

1.1 Background

In this era of global markets and worldwide competition, it is imperative that manufacturing companies continuously upgrade their products in order to meet the fast-changing needs of the customer. Under these circumstances it becomes necessary to reduce the design-to-market times of the new products. More and more companies are moving from the traditional “over-the-wall” design engineering to concurrent engineering. Concurrent Engineering has been defined in a number of ways. One of most common and oft-repeated definitions is “a systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support. This approach is intended to cause the developers, from the outset, to consider all elements of the product life requirements” [28].

In most of the engineered and manufactured systems, such as automotive vehicles and aircrafts, interactions among the various components of the system often make the system a synergistic whole that is greater than the sum of its parts. The objective of Multidisciplinary Design Optimization (MDO) is to provide a consistent, formalized method for obtaining an optimized design of a complex system by taking into account the analysis solutions of multiple disciplines each of which optimizes the design based on different, often competing, criteria. [1]
Numerous problem-solving environments commonly referred to as “frameworks” have been developed that aim at maximizing the benefits of concurrent engineering by facilitating interdisciplinary interactions and enabling multidisciplinary design optimization (MDO) within a short time frame. A framework is defined as a “hardware and software architecture that enables integration, execution and communication among diverse disciplinary processes” [21].

Salas and Townsend [23] have laid down the requirements for a multidisciplinary design optimization framework. Any such integrated design environment geared towards concurrent engineering should facilitate parallel and distributed computing and allow multiple processes to be seamlessly linked in an automated fashion thereby eliminating the hurdles present in transferring data between processes. The aim of such frameworks is to provide a common working environment that increases interdisciplinary interactions and reduces the design cycle time.

An efficient and effective data exchange mechanism is indispensable for such integrated frameworks to accomplish their objectives. Data needs to be exchanged between various tools within the environment. Different tools might require data in different formats and the output of one tool might serve as input to another tool. With the increased interaction between various disciplines and tools in concurrent engineering frameworks, it is imperative to maintain the consistency and integrity of the data.
1.2 Scope of this work

The various data exchange mechanisms being used in some of the existing frameworks have been looked into and an improved data exchange mechanism has been proposed in order to handle some of the shortcomings of the existing mechanisms. The data exchange mechanism proposed comprises two major categories of components, namely the runtime and the design-time components. The design-time components would provide the necessary user interface to change the data exchange and parsing parameters. The actual mechanism of data exchange would be handled by the runtime components. The scope of this work is restricted to the runtime components. In addition to developing the overall mechanism, the objective of this work has been to develop two major tools namely the Vector tool and the Name-Value tool for handling data. The Vector tool deals with one-dimensional data and the Name-Value tool manages data in the form of name-value pairs.

1.3 Organization

Chapter 2 looks into some of the existing data exchange mechanisms. A brief overview of the various components of the proposed mechanism and how they function in tandem is given in Chapter 3. The two major tools used for data exchange currently implemented are the Vector Tool and the Name-Value tool. The Vector tool handles data in the form of one-dimensional array of elements separated by delimiter characters. A detailed explanation of this tool is given in Chapter 4. The Name-Value tool, which deals with data in the format of name-value pairs separated by delimiters, is explained in detail
in Chapter 5. In order to attach tools to regions that are semantically different from the preferred format of the tool, adapters are constructed at runtime. Adapters convert data between formats and enable different tools to attach themselves to semantically incompatible regions without any extra work from the user. Chapter 6 describes the types of adapters available and their roles. The central part of the data exchange mechanism is the data-caching algorithm. This algorithm is responsible for maintaining the consistency of data during the data exchange process. A detailed explanation of this algorithm is given in Chapter 7. The conclusions and recommendations for future research are given in Chapter 8. A step-by-step process of how to use the data exchanger and the Vector tool in particular is given in Appendix A. A similar step-by-step explanation of the Name-Value tool is given in Appendix B. Appendix C explains the use of the Name-Value, Vector and Table tools to obtain information from a file generated by a simulation tool which models a simple processing system.
2 Existing Frameworks

Over the past 15 years, a lot of effort has been expended towards developing MDO frameworks that help in accelerating the design process and reducing the time-to-market for manufacturing products. Salas and Townsend [23] have listed the requirements that are considered necessary in any framework for an MDO environment. Some of the requirements are listed below.

- Should provide an intuitive Graphical User Interface (GUI)
- Should be based on Object Oriented Design principles enabling distributed computing
- Should be extensible to integrate new processes
- Should handle large problem sizes
- Should enable Collaborative Design
- Should facilitate easy configuration of problems
- Should allow integration of legacy and proprietary codes
- Should execute multiple processes in parallel
- Should support user interaction during the design cycle
- Should provide database management features to maintain a centralized database
- Should provide the capability to visualize the intermediate and final results
- Should provide feedback on the system status during the design process
Essentially, the framework should provide an integrated design environment where multiple processes can be seamlessly linked in an automated fashion thereby eliminating the hurdles present in transferring data between processes. Data management is of vital significance while permitting concurrent and distributed design evaluation. In order to make the design process more efficient, it is imperative that such frameworks have a concrete data exchange mechanism that maintains the integrity and consistency of the data across various tools. A variety of MDO framework tools have been developed both at commercial establishments and at government funded research organizations. It is difficult to find a single tool that meets all the requirements as laid down by Salas and Townsend. Different tools concentrate on different aspects of the framework requirements. Some of the relevant tools have been described below with their salient features and drawbacks with more emphasis on their data management systems.

2.1 ModelCenter

ModelCenter [16, 2, 15] is a software tool from Phoenix Integration Inc. that provides an integrated environment in which design and manufacturing engineers can perform distributed modeling and analysis. Using ModelCenter, it is possible to integrate different simulation and design tools in order to facilitate a thorough analysis of the design and help solve a wide variety of problems. Designers can access the different modules of the tool from remote locations thereby enabling distributed computing. ModelCenter works best in combination with another tool called Analysis Server.
Analysis Server is a Java-based wrapper tool that allows the user to wrap design and analysis tools into reusable components that can be published on the network. The Analysis Server protocol is very similar to HTTP. These components, once published, can be accessed from any computer irrespective of the configuration. Applications which are based on input and output files, are wrapped using the FileWrappers. FileWrappers describe the methodology to parse input files and run the applications. Excel worksheets can be integrated into the environment using ExcelWrappers. Other complex simulation tools can be wrapped using Java Wrappers, which are essentially Java Beans.

ModelCenter maintains the data integrity during the concurrent design process using a sophisticated Constraint Management System. It handles the data synchronization using a complex chaining algorithm. The underlying open architecture permits the user to control the flow of data through scripts and automatically synchronizes the data all the way up to the original input file. ModelCenter caters to both the end-user who would just want to wrap the tools and use them as they are and the programmers who would like to add some special functionality into the simulation tool. ModelCenter is written in C++ and is native to Windows NT. Although Analysis Server is Java-based and ModelCenter works in tandem with the Analysis Server by using the tools wrapped by the Analysis Server, platform dependency issues are a matter of concern since ModelCenter can be run only on Windows machines.
2.2 AML

Adaptive Modeling Language (AML) [4, 3, 25] is an object oriented modeling paradigm used in an advanced engineering framework to integrate the entire product development cycle and is developed by TechnoSoft Inc. It uses the syntactic characteristics of LISP in order to implement the framework. There are two main types of relations in the AML framework viz. class-subclass relation, which allows the traditional object oriented features like data encapsulation, multiple inheritance and polymorphism, and part-subpart relation, which depicts the tree structure of parts and its components. The AML framework comprises various modules for the different knowledge domains. The framework also allows for an open integration of additional modules that may be coded in different languages like C, C++, FORTRAN or LISP.

AML uses a unified part model paradigm in which all the information related to a particular part is stored in a single hierarchical object model of the part. This eases the bookkeeping of the data considerably. AML maintains the consistency of data across various simulation models through constraint mechanism. The properties involved in the design analysis are calculated only when the need arises. This is called the “demand-driven” behavior or “lazy” evaluation. In order to sustain data integrity, when a particular parameter is changed during the design analysis, all the other associated parameters are “smashed”. This is referred to as the “dependency tracking” behavior, which enables the engineer to manage the feedforward and feedback of design information. The parameter that is not smashed is called an “unbound” parameter. The value of the smashed parameter is recalculated only when a reference to it is made at a later stage.
The AML framework does not have a built-in optimization algorithm. It can, however, link with other optimization tools either through AML interfaces or by running the stand-alone programs and analyzing the output files. One such interface for the Design Optimization Tool (DOT), an optimization program in FORTRAN, with AML has been implemented and is called the Automatic Differentiation Algorithm (ADA).

### 2.3 iSIGHT

iSIGHT [12,23] is a complete design exploration environment that enables linking of various commercial tools and leveraging existing tools from Engineous Software Inc. iSIGHT also facilitates Microsoft Excel spreadsheets into the design environment. The iSIGHT environment is divided into several modules. The Tcl language is used as the interpreter to integrate the various processes. The problem can be constructed both through the GUI and the Multidisciplinary Optimization Language (MDOL). The various input and output parameters can be specified in the initial GUI environment. Existing legacy applications can be wrapped and the necessary input files are parsed to obtain the relevant information. This framework provides limited support for parallel processing and distributed computing. Since there is no specific data management system, the consistency of the data is maintained by actually writing and parsing the input and output files.

### 2.4 FIDO

Framework for Interdisciplinary Optimization (FIDO) [27, 26] provides a distributed computing environment for executing multi-disciplinary computations on a
heterogeneous network of computers. The concept being employed in FIDO is parallel and distributed computing. Although FIDO does not follow object-oriented design principles, emphasis has been placed on modularity. Codes for each discipline are executed on separate processors under the control of an executive on another processor and data are exchanged through the centralized data manager. The different modules communicate with each other through the Communications Library, which forms the backbone of the system. The data manager handles the storage and movement of data within the design environment. The data to be handled during the design process are defined during the set-up phase. A file containing all the pertinent information regarding the data to be monitored is created during the set-up phase. The conversion of data between formats across the various modules is done by a separate code called the interdisciplinary interface code (Interface), which also handles data transmission with the data manager. In order to facilitate viewing of the data during the design process, a separate segment called “Spy” has been incorporated into the environment. FIDO provides very little support by means of GUI for problem formulation construction.

2.5 FACET

Framework for Analysis of Coupled Engineering Techniques in Simulation (FACET) [10, 11] is an integrated design tool that allows the designer to carry out multidisciplinary analysis in a simulation setting. The underpinning idea of FACET is its modularity. Currently there are three main modules viz. the simulation module, CASCADE, the solution strategies module and the analysis convergence module. Besides
these three main modules, there are other modules for optimization, system planning and result verification. Each module of FACET functions as an independent entity. The modules communicate through input and output files. Each module uses a set of input files and generates a set of output files, which in turn may act as input files for some other modules. The GUI front-end has been developed using the Motif toolkit, which utilizes the ANSI C programming language. A series of GUI windows guides the user through the initial set-up procedure wherein the user can specify the variables and their optimization constraints. The FACET framework has been coded using Motif and hence is platform-dependent.

2.6 OPTIMUS

OPTIMUS [14, 23] is another software tool that helps in integrating multiple simulation tools together. This manages the exchange of data between the various simulation tools and maintains the integrity of data. This is achieved through an underlying intelligent technology. Once the design variables are marked in the selected input files and the structure of input files is identified, OPTIMUS handles the optimization process. It relieves the analyst of considerable manual work in creating new input files for each design and extracting the values of interests from the output files. The input and output files are identified using the GUI module. A command file is created which includes the design inputs, design outputs and input and output file parsing commands. The environment also provides facility to integrate legacy applications into the framework environment.
2.7 DOME

Distributed Object-based Modeling Environment (DOME) [23] from MIT CADLab is a prototype implementation of a product development environment where design and manufacturing process is weighed against customer, company and market objectives. Central to the framework is the concept of Object-based Modeling and Evaluation (OME). The design problems are broken down into more manageable smaller modules and these modules interact and communicate with each other through standard distributed communication protocols. The OME provides a mechanism to create modules and link them together within the environment. The framework consists of a modeling layer in which the user defines the problem in terms of modules and their interactions. Based on these, an implementation layer is created which locates the remote or distributed modules and establishes communication between the local and remote modules. The interaction between the modules occurs through the distributed interface, which is wrapped around the distributed modules. This framework is implemented in C++ and Common Object Request Broker Architecture (CORBA), which is OMG’s open, vendor-independent specification for an architecture and infrastructure that computer applications use to work together over networks, is used to implement the communication protocols. The Graphical User Interface is implemented in X Window system using Motif.
2.8 CML

Compositional Modeling Language (CML) [18] is an object-oriented language that facilitates construction of behavior models of physical systems. These models are used in the early design stages and are grounded to a common ontology. The CML modeling environment enables information sharing between models used in the early stages of design. Collaborative Device Modeling Environment (CDME) is a web-based interface through which engineering specialists interact with the models defined in CML. Since CML is not designed to be numerically efficient, it is expected that CML/CMDE would be used in conjunction with an overall agent-based concurrent engineering framework. A Java Agent Template (JAT) has been developed to create agents that wrap the various engineering tools into the framework. The agents communicate with the CML models through Open Knowledge Base Connectivity (OKBC) protocol over the Internet and use the Knowledge Query and Manipulation Language (KQML) for exchange of information between agents.

2.9 IMAGE

Intelligent Multidisciplinary Aircraft Generation Environment (IMAGE) [7, 8, 9], from Georgia Tech University, is based on two basic activities viz. modeling design processes and providing technologies for implementing a suitable design environment. It provides an object-oriented data management utility for use during the design process. IMAGE works on the premise that an “agent” is one of the key technologies required for implementing an integrated computing environment.
An agent is defined in [7] as “a resource that has been modeled and wrapped for inclusion in a distributed design environment. Agent design requires a designer-centered, bi-directional wrap that is independent of proprietary boundaries and capable of supporting increasing fidelity models”. In this framework, there are three main agents namely, the resource, the model and the wrap. The wrap handles the bi-directional information exchange within the design environment. This is achieved using the Tk/Tcl (which stands for Tool Command Language which is an application-independent interpreted and typeless scripting language highly useful for integrating applications) utility package developed at U.C.-Berkeley [17].

The wrap comprises six components: the Communications Interface, the Protocol Filter, the Model Interpreter, the Resource Interpreter, the Graphical User Interface, and the Low Level Compliance layer. The Communications Interface is used for the bi-directional exchange of data within the design environment. The Protocol Filter provides the method for data encapsulation and a transfer format. One such transfer format is KQML, which provides tools to route information through the design environment. The Protocol Filter also handles the translation of data between these formats. The Model Interpreter and the Resource Interpreter are responsible for processing the information. The Graphical User Interface provides the necessary interface to enable user interaction. The Low Level Compliance layer acts as a “facilitator” to other components and also provides a location for data storage.
2.10 FIPER

A four-year collaborative effort has been initiated under the sponsorship of National Institute for Standards and Technology (NIST) Advanced Technology Program (ATP) to develop a Federated Intelligent Product Environment (FIPER) [22]. This is a joint venture among General Electric, Parker Hannifin, BFGoodrich Aerospace, Engineous Software, Ohio University, Stanford University and Ohio Aerospace Institute to develop design support technologies. FIPER is a flexible design environment that facilitates concurrent access of design data over a distributed network and integrates knowledge-based engineering (KBE) tools into the Intelligent Master Model, which is an extension of the CAD Master Model paradigm. FIPER is a web-based distributed software architecture based on Java, which aims to “drastically reduce design cycle time, and time-to-market by intelligently automating elements of the design process in a linked, associative environment, thereby providing true concurrency between design and manufacturing. This will enable distributed design of robust and optimized products within an advanced integrated web-based environment” [5]. KBE facilitates the capture of the knowledge that acts as a basis for the design process. The KBE in the FIPER environment will be used in the MDO analysis for the decomposition of the optimization problem into smaller and more manageable parts. The FIPER Service Manager handles the requests from different clients and ensures data concurrency and consistency.
2.11 Drawbacks of Existing Data Exchange Mechanisms

From the above analysis of some of the design framework tools available in the market, it can be seen that it is difficult to identify a single tool that meets all the objectives of the multidisciplinary design optimization framework. Moreover, most of the tools do not utilize the web-based technologies that enable remote access to the environment from varied platforms, although an allusion to this has been made by Salas and Rogers [21]. An attempt has been made to study these existing design framework tools from the perspective of the mechanism of data exchange employed in each of these framework tools.

In FIDO, the data management system is taken care of by the FIDO Communications Library (COMMLIB). The COMMLIB acts as the backbone of the FIDO Framework and the various modules interact with one another through the COMMLIB. The Data Manager provides centralized access to the data in standardized formats through the COMMLIB using the message-passing functions in the library. The drawback of this system is that it does not provide a user friendly interface to specify the required design parameters. AML is based on the LISP programming language, which has the disadvantage that is fairly uncommon, requiring most users to familiarize themselves with the language before becoming a proficient user of AML. The user interface provided is also fairly ordinary because even simple declaration of variables requires programming knowledge. ModelCenter and iSIGHT provides excellent wrappers for Excel-based tools and command line tools. iSIGHT has a well-designed user interface that simplifies the task of dealing with complicated source files.
In DOME, a standard communication protocol manages the interaction between various modules of the framework thereby permitting inter-process communication. A wrapper interface is constructed around the local modules such that communications between the local modules are hidden from the remote modules and the interface handles the service calls from the remote modules. The changes in one module are communicated to the related modules through service calls to the concerned modules. This keeps all the different modules consistent to the local design changes. The GUI in DOME is implemented using the X Windows system and this permits the user to examine and make alterations to the configuration of the design problem. DOME also uses CORBA for interaction with CAD applications and database management systems. The CML environment is well suited for information exchange across a heterogeneous network. A web-based interface called CMDE is also available for users to interact with models in CML.

FACET is comprised of many modules interacting with one another. These modules communicate with each other through data files. Each module writes out the data generated by it into a data file, which is read in by the post-processing module. Sometimes the data is written out as a code, which can subsequently be compiled by other modules. The GUI permits extensive intervention by the user during the design process.

The information model used in the IMAGE implementation incorporates the use of Process Elements. These elements are instantiated as Agents. A vital component of an Agent is a Wrap. A Wrap is responsible for communicating information among resources
while conforming to the data exchange standards. The Wrap utilizes a Communications Interface to enable bi-directional information exchange within the design environment. The Communications Interface also handles events from the Graphical User Interface. The GUI is based on X-Windows and requires the necessary X-libraries and an X-display. KQML provides necessary information to route the information within the design environment. The knowledge is represented in Knowledge Interchange Format (KIF), which is based on LISP-like constructs. A Protocol Filter handles the translation of data across formats. The wrap is implemented using Tk/tcl utility package. Tk/tcl is an interpretive windowing system. The agents are platform independent albeit only within the UNIX operating systems.

OPTIMUS has a very intuitive GUI where the design data in the input and output files are specified. The GUI is written in C++ and Motif. The actions taken through the GUI results in a command file. The command file contains the specifications for the input and output file parsing. The user can edit this command file in order to include additional commands.

An improved data exchange mechanism has been proposed which would be implemented in the FIPER framework. This data exchange mechanism would be implemented in Java. Since Java is inherently compatible across multiple platforms, the data exchange mechanism and the framework is consequently accessible across different platforms. The capability to handle multiple data source types like Text Files, Microsoft Excel spreadsheets, and JDBC is also built into the mechanism. The data consistency and integrity is maintained using a data-caching algorithm. Data are converted across various
formats using adapters. The integral components of the Data Exchanger include Region, Tool, Exchangers and Adapters. The User Interface for the data exchange mechanism is much friendlier and easier to grasp than some of the existing data exchange mechanisms. Although the option to write the code for the extraction of data from the source files exists, it is not imperative for the user to know any programming in order to effectively use the Data Exchanger. Web-based technologies have also been incorporated into the new framework so that the tool can be accessed from remote locations.
3 Data Exchanger

In any MDO framework, a data management system is necessary to maintain the consistency of data across the different tools. The Data Exchanger handles the data management system in the FIPER framework. The Data Exchanger has been designed to work within the FIPER environment and also as a stand-alone component. It is used to extract data from or insert data into multiple data sources apart from FIPER parameters. The data sources planned for version 1.0 of FIPER include Files and Excel worksheets. Figure 1 displays a graphical view of the exchange vis-à-vis the data sources.

Figure 1. Graphical Representation of data exchanger and data sources
Multiple data sources can be used by a single Data Exchange component. The data is represented in different formats viz. Vector, Item, Table or Text. A vector is a single dimensional array of elements. An item represents the value portion of a name-value pair. A table denotes a two-dimensional representation of data with rows and columns. The text format is data in the form of a continuous sequence of characters.

The Data Exchanger consists of a run-time component and a design-time component. The run-time component deals with actual update of the files and requires instantiation of the various java classes. The third-party applications and FIPER parameters are also updated during the run-time. No GUI is involved in this component since the design-time component comprises a graphical user interface (GUI) portion that allows the user to make modifications to the parameters involved in the data exchange commands. The GUI portion is implemented using Java Swing components.

Some of the goals for the design of the Data Exchanger enumerated in [6] include extensibility by providing facility to add multiple data sources/sinks and tools for processing data without any changes to the fundamental Data Exchanger code and flexibility to use any tool with a compatible region. Efficiency is also one of the foremost concerns in a Data Exchange mechanism. There are two modes for the Data Exchanger viz. Sequential wherein a constant amount of memory is used by the tool irrespective of the size of the data and Random where the memory used is proportional to the size of the input. It is also desired that there are no hard-coded limits on the number of nested tools and regions built into the architecture. It should be dependent on the user’s available
system memory. Any failure during the data exchange mechanism should be reported as an error.

The components of the Data Exchanger module are explicated beginning in Section 3.1. Figure 2 shows the interaction of the various components of the data exchange mechanism in context of the FIPER environment.

![Figure 2. Data Exchanger Components](image)
3.1 Exchanger

The Exchanger identifies the source and destination of data. It is attached to the data source file and connects to various tools that perform the actual data exchange. The data flow in the exchanger is specified in two different modes viz. GET and PUT. If the desired implementation is such that the data needs to be only extracted into the FIPER environment, the exchanger can be instantiated in the GET mode. The GET mode precludes the possibility of the data being written out. If it is desired that the data be modified and the modified data be saved into an output file, the exchanger can be instantiated in the PUT mode. The PUT mode necessitates that the destination of the output be specified. The destination could be either the data template itself or another data source.

File Exchangers are used to parse text files and handle transfer of data between the files and FIPER environment. Other exchangers like Excel Exchanger, CAD Exchanger and JDBC Exchanger help to exchange data from Excel worksheets, CAD systems and JDBC databases respectively.

In order to use the Excel Exchangers, it is essential to have third-party software that enables seamless integration of Microsoft Excel with Java. One such product is the J-Integra Suite from Intrinsyc Software [13]. This enables software developers to link to COM products (products that are developed based on the Common Object Model architecture that allows components developed by different vendors to be combined into a variety of applications) and leverage existing software code. The Java - Excel Bridge enables Java developers to access Microsoft Excel and create and modify information in
There are other products available in the market that provides this functionality but this is the one that was looked into and used in this mechanism.

3.2 Tool

The Tool is used to interpret and manipulate the data in the region (Regions are defined in Section 3.3). It provides operators to sub-divide the data into regions. They also provide operators to retrieve properties associated with the region. Tools can attach to only specific types of regions depending on the semantics associated with the region. The data flow in tools can be controlled using the mode of the tool. There are two modes permitted for the data flow in the tool: Sequential and Random. In the sequential mode, the data flow is unidirectional. Hence, the tool can make only one pass through the data in the region. The data is updated as the tool processes the data and no backwards movement is permitted in the region associated with the tool in this mode. This mode is useful for file-type data sources especially with large files as the memory usage is kept to the minimum. Figure 3 shows the sequential access of data.
Random mode permits bi-directional access of the data in the region. The tool buffers the entire data in the region and the data is not updated until the tool is closed. The memory usage in this case is dependent on the amount of data associated with the region. Figure 4 shows the random mode.
The different types of tools include Vector, Name-Value, Table, Partitioner, Text, Fixed, Name-List, BNF and XML. Some of these tools would be incorporated into the Data Exchanger architecture in Version 2.0.

3.2.1 Vector

The Vector tool handles data that can be converted into a one-dimensional array of elements. Delimiters are characters that separate the elements associated with the tool. A typical region to which the vector tool can be attached to is the OneD Region (Regions are defined in Section 3.3), which represents data in the format of a vector of elements in the row or column form as well as in the text form separated by punctuations. Figure 5 shows a typical vector region.
3.2.2 Name-Value

The Name-Value tool processes data in the form of name-value pairs where delimiters separate the name and the value portions and successive name-value pairs are also separated by delimiters. The typical region to which the name-value tool can be attached is the Item Region (defined in Section 3.3), which represents data in the format of contiguous name-value pairs. This tool uses an item-accessing scheme. An example of a typical name-value region is shown in Figure 6. This could be a part of a data file which specifies the material of the component under consideration.

![Figure 5. A typical vector of elements](image-url)
3.2.3 Partitioner

This tool processes data in the form of regular text. The typical region to which the partitioner tool can be attached to is the Text Region (defined in Section 3.3). This tool helps to divide the region into sub-regions based on line numbers or using pattern-matching techniques. Figure 7 shows a typical region on which a partitioner would be attached.
3.2.4 Table

A table tool provides a two-dimensional accessing scheme for the data in the associated region. Delimiters separate the elements in each row and column. Table tools can be directly attached to a TwoD region (defined in Section 3.3). The Table can be further sub-divided based on rows, columns and sub tables. The rows and columns can be represented in terms of numbers or string headers. In order to use the Table tool, it is imperative to have the regular expression libraries. Regexp [19] is a pure Java Regular Expression package available for use from Apache Software Foundation. Originally developed by Jonathan Locke, the package provides the necessary libraries for implementing regular expressions in Java. Figure 8 shows a typical table region.
3.3 Region

The Region represents the portion of the data associated with a particular tool it is attached to. The contents of a region can be read into a FIPER parameter (typed containers for data explained in Section 3.7). Similarly the contents of a region can be replaced by the contents of the FIPER parameter. The contents of a region can be copied into another semantically similar region. Regions do not store data; instead they have pointers to the tool that created it. A region can be recursively sub-divided into smaller regions using other tools. The different types of regions include Text Region, Scalar Region, Item Region, OneD Region and TwoD region among others.

A text region is used to represent files and other multi-line formats. The region comprises an array of lines. It is necessary to convert the data in this region into a structured format in order to attach the structured tools like Name Value or Vector. A

<table>
<thead>
<tr>
<th>PartNo</th>
<th>Length</th>
<th>Width</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>10</td>
<td>15</td>
<td>35</td>
</tr>
<tr>
<td>1002</td>
<td>15</td>
<td>25</td>
<td>45</td>
</tr>
<tr>
<td>1003</td>
<td>20</td>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>
scalar region is a region type that comprises an arbitrary sequence of characters. This region supports the functionality of directly writing into or reading from Java variables.

An item region represents a collection of name-value pairs with delimiters separating the name and value portions and also successive name-value pairs. This region is typically associated with the Name-Value tool. A OneD region represents a one-dimensional collection of vector elements where the elements are separated by delimiters, which may be strings or new-line characters. This region is typically associated with the Vector tool.

A two-dimensional array of contiguous regions is represented as a TwoD region. Delimiters in the form of strings or new-line characters separate the elements in each cell of the two-dimensional array. The Table tool typically attaches itself to this type of a region.

### 3.4 Tool Operator

The operators that retrieve characteristics of the region associated with the tool are called tool operators (TOP). They may be also used to set the attributes of the region. Figure 9 shows a typical TOP. A vector tool is attached to the OneD region and the TOP numElements returns the number of elements in the region.
## Region Operator

Every tool has region operators (ROP). These are operators that help subdivide the region into sub-regions and aid in extracting the data from the region. They also help in navigating through the data associated with the region. Figure 10 shows a typical region operator.
3.6 Adapter

Each tool can be attached directly only to a set of pre-defined regions. For example, a vector tool can be attached directly to a OneD region since the OneD region has data in the structured format required by the Vector tool. In order to attach tools to semantically different regions, the data has to be modified into the required structured format. Adapters perform this role of converting the data into various structured formats. Tools are connected to each other through adapters. Adapters are invisible to the users and do the conversions behind the scene. Adapters are also responsible for converting the data back into the original format before updating the parent. The adapters together with the exchangers maintain the data consistency in the Data Exchange framework through a data-caching algorithm. Figure 11 shows the role of adapters in the data exchange
mechanism. In the figure, a tool is attached to the text region and another tool is attached to the table region. The two regions hold data in two different formats. An adapter does the data translation between the two formats so that the tools can communicate with each other. When the tool on the text region requests (GET) data from the table region, the adapter translates the data in the table region to a format acceptable to the requesting tool. Similarly, when the tool on the table region sets (PUT) the value of data in the text region, the adapter converts the data in table region into the format in which it is stored in the text region.

Figure 11. Adapters
3.7 FIPER Parameter

These are typed containers for data. The contents of a region can be copied into a FIPER parameter and likewise the value of the FIPER parameter can replace the contents of the region. The only caveat involved is that the dimensionality of the FIPER parameter and the region should match.

3.8 Conceptual Example

Having looked at the various components of the data exchanger, a conceptual example of how each of the components fit into the total picture of data exchange is in order. Consider a text file to be parsed for information. Since it is a text file, a file exchanger can be attached to the file in order to get a hook on to the file contents.

In order to extract useful information from the file, it is necessary to break up the data in the file into manageable chunks. This can be achieved by attaching a partitioner on the exchanger to partition the file into smaller regions. Figure 12 shows the partitioner being attached to the exchanger.
In order to extract small chunks of data from the file, the sub-region operators of the tool are employed. In keeping with the above concept, in order to extract data from a list of elements in a file, a Vector tool can be attached to a sub region created by the Partitioner tool attached to the File Exchanger. This is graphically displayed in Figure 13.

Figure 12. Partitioner tool on the Exchanger
Similarly, if there were name-value pairs in a particular chunk of data associated with the file, it can be extracted using a Name-Value tool attached to a region of the Partitioner tool, which in turn is attached to the File Exchanger. This is shown in Figure 14.
Figure 14. Name-Value Tool in action
4 Vector

The Vector tool processes data as a one-dimensional array of strings. The Vector tool uses the OneD Region, which is a single dimensional array of contiguous regions, to store elements. The vector of elements may be constructed by splitting text based on delimiters or by applying row or column operations on a table. An example of a simple vector would be a list of elements separated by an element delimiter. The details of the Vector tool are stored in the Vector Details object.

4.1 Vector Details

This object acts as a data structure for storing the various details related to the data in the region associated with the Vector tool. There are 11 possible specifications for the data contained in the region. These are split into three groups.

4.1.1 Tool Specifications

This group of details includes the aspects of the tool such as

- startIndex – represents the initial index of the first element in the region. Typically it is defaulted to 1, but the user has the provision to specify any integer as the start position.

- resizable – This is a Boolean field, which indicates if elements can be added to or deleted from the region. If the field has the value true, additions and deletions of elements are permitted. The default value for this detail is true.
4.1.2 Delimiters

This group of specifications deals with the various delimiters permitted in the data associated with the region. They include

- elementDelimiter – represents the string that separates the various elements in the region. The default value is a string comprising ‘,’ (comma), ‘;’ (semi-colon), or ‘\t’ (tab).

- escape – represents the character that denotes that the character appearing after it does not have any special meaning. The default value is backslash (‘\’).

- wsChars – represents a string of characters, which are accepted as white space characters in the data. The default value is a string comprising ‘ ‘ (blank space), ‘ \t’ (tab), ‘\r\n’ (carriage return), and ‘\f’ (line feed).

- whiteSpaceIgnored- represents a Boolean variable that specifies whether the white space characters occurring in the data are to be retained or ignored. The default value is true which implies that the white space characters are ignored.

- eolIsDelimiter – is a Boolean variable that specifies whether the system-dependent end of line character(s) can act as an element delimiter. The default value is true which indicates that the end of line character(s) is also an element delimiter.

4.1.3 Groupers

Groupers represent the set of specifications that concern the parenthesis and quote characters. They include
• openChars – represents the set of acceptable characters that can assume the characteristics of an open parenthesis or braces. The default value is a string comprising ‘(‘, ‘{‘ and ‘[‘.

• closeChars – represents the set of acceptable characters that can assume the characteristics of a close parenthesis or braces. The default value is a string comprising ‘)’, ‘}’, and ‘]’.

• quoteChars – represents the set of characters that are acceptable as quotes. The default value is set to double quotes (‘”’).

• doubleQuote – represents a Boolean variable that specifies whether two consecutive quote characters represent an escape behavior. The default value is true.

4.2 Region Operators (ROPs)

Every tool has region operators (ROP). These are operators that help subdivide the region into sub-regions and aid in extracting the data from the region. They also help in navigating through the data associated with the region. The Vector Tool has three ROPs associated with it. Each of these ROPs takes different arguments and returns different types of regions.

4.2.1 Element

This ROP retrieves the element based on the index of the elements in the region and returns it in the form of a Scalar Region
4.2.2 SubVector

This ROP retrieves the elements from a start location to an end location (including the end Index in the subRegion) and puts it in a OneD Region. The start location is typically an integer denoting the index of the corresponding element in the list of elements associated with the region. The end location could be specified in multiple ways

- Integer corresponding to the index of the last element associated with the region.
- An object that specifies the end to be the end of the parent region associated with the tool – This is implemented by creating an instance of the Locator object described in Section 4.2.2.1.
- An object that specifies the number of elements associated with the region – This is implemented by creating an instance of the VectorLength object described in Section 4.2.2.2.

4.2.2.1 Locator

This specifies the various predefined begin and end points for a region that is associated with a tool. The available options are listed below

- SOP – This implies that the locator is the Start Of the Parent region.
- EOP – The locator is the End of Parent region, which implies that the region whose end is specified by this locator ends when the region associated with the parent tool ends.
• EOC – The locator is the End of Child region. A region with its end location specified as EOC ends when the region no longer concurs with the format of the child tool attached to it.

• CURRENT – This locator indicates the current position of the tool. When used as a start locator, it refers to the end of the previous region and when used as an end locator, it refers to the beginning of this region. This is typically used in the Partitioner tool to sub-divide its associated region.

4.2.2.2 VectorLength

This class has been derived from the Locator class and denotes the number of elements that are to be associated with the tool. Setting the value of the parameter “numOfElements” in the class specifies this number.

4.2.3 All

This ROP retrieves the entire region and puts it in a OneD Region. The begin index is the start index of the tool and the end is given by the end of the region associated with it. This region operator is implemented as a sub-vector region with the end specified using Locator object.

4.3 Tool Operators (TOPs)

The operators that retrieve characteristics of the region associated with the tool are called tool operators (TOPs). They may be also used to set the attributes of the region. There is currently only one TOP associated with the Vector Tool namely the
numElements which returns the number of elements in the region associated with the tool.

4.4 Important Methods

Some of the important methods that assist in manipulating the data associated with the Vector tool are enumerated below along with the functionality they achieve.

- `get(<int> index)`: returns the vector element at the specified index in the region.
- `set(<int> index, <object> value)`: replaces the value of the element at the specified index with the new value.
- `add(<int> index, <object> value)`: inserts a new vector element at the specified index in the region.
- `add(<object> value)`: inserts a new vector element at the end of the region.
- `remove(<int> index)`: removes the vector element at the specified index from the region.
- `clear()`: removes all the elements from the region associated with the vector tool.
- `copy(<OneDRegion> region)`: copies the contents of the given region into the region associated with this tool. It essentially replaces the contents of this region with those of the given region.
- `size()`: returns the number of vector elements in the region.

Of the above listed methods, only the set and get methods are applicable to Element region since it holds only one vector element.
5 Name Value

The Name Value tool processes data as Name-Value pairs. This tool uses the Item Region, which maps names to contiguous regions, as the data structure for storage. The name is a simple string of text and the value is also considered to be a block of text. In a line of text, the name ends at the point where the value delimiter is found and the value ends where the end delimiter is found. The name of the name-value pair is not permitted to cross multiple lines albeit this restriction does not apply to the value. The details of the name-value pairs are stored in the details object.

5.1 Name Value Details

This object acts as a data structure for storing the various details related to the data in the region associated with the Name Value tool. There are 14 possible specifications for the data contained in the region. These are split into three groups.

5.1.1 Tool Specifications

This group of details includes the aspects of the tool such as

- startIndex – represents the initial index of the first element in the region. Typically it is defaulted to 1, but the user has the provision to specify any integer as the start position.
• resizable – This is a Boolean field, which indicates if elements can be added to or deleted from the region. If the field has the value true, additions and deletions of elements are permitted. The default value for this detail is true.

• handleDuplicateName – determines the behavior of the tool when the input source has name-value pairs with duplicate names. It can take any of the following three values
  o FIRST_INSTANCE – indicates that duplicate names are permitted in the input source and the first occurrence of the name-value pair with duplicate names is considered.
  o LAST_INSTANCE – indicates that duplicate names are permitted in the input source and the last occurrence of the name-value pair with duplicated names is considered.
  o THROW_ERROR – indicates that duplicate names are not acceptable in the input source and an error is thrown if duplicate names are found.

The default value is set to THROW_ERROR so that an error is thrown if name-value pairs with duplicate names are found in the input source.

5.1.2 Delimiters

This group of specifications deals with the various delimiters permitted in the data associated with the region. They include

• delimiter – represents the string that separates two name-value pairs in the region.

The default value is a string comprising ‘,’ (comma), and ‘;’ (semi-colon).
- **valueDelimiter** – represents the string that separates the value from the name in the name-value pair. The default value is set to be white space (" ").

- **valueExpression** – represents the regular expression for the value delimiter. It is generated at runtime from the value delimiter if the user does not specify an expression.

- **escape** – represents the character that denotes that the character appearing after it does not have any special meaning. The default value is backslash ("\").

- **wsChars** – represents a string of characters, which are accepted as white space characters in the data. The default value is a string comprising ‘ ‘ (blank space), ‘\t’ (tab), ‘\r\n’ (carriage return), and ‘\f’ (line feed).

- **whiteSpaceIgnored** - represents a Boolean variable that specifies whether the white space characters occurring in the data are to be retained or ignored. The default value is true which implies that the white space characters are ignored.

- **eolIsDelimiter** – is a Boolean variable that specifies whether the system-dependent end of line character(s) can act as an element delimiter. The default value is true which indicates that the end of line character(s) is also an element delimiter.

**5.1.3 Groupers**

Groupers represent the set of specifications that concern the parenthesis and quote characters. They include
- openChars – represents the set of acceptable characters that can assume the characteristics of an open parenthesis or braces. The default value is a string comprising ‘(’, ‘{’, and ‘[‘.
- closeChars – represents the set of acceptable characters that can assume the characteristics of a close parenthesis or braces. The default value is a string comprising ‘)’, ‘}’, and ‘]’.
- quoteChars – represents the set of characters that are acceptable as quotes. The default value is set to ‘‘’ (single quotes) and ‘”’ (double quotes).
- doubleQuote – represents a Boolean variable that specifies whether two consecutive quote characters represent an escape behavior. The default value is set to true.

5.2 Region Operators (ROPs)

The Name Value Tool has four ROPs associated with it. Each of these ROPs takes different arguments and return different types of regions. The Element operator retrieves the item based on the index of the name-value pair in the region and puts it in a Scalar Region. The Item ROP retrieves the item based on the name of the name-value pair and puts it in a Scalar Region. The All ROP retrieves the entire region and puts it in an Item Region.

The DelayedItem ROP works as a trade-off between sequential and random modes. In the event of a delayedItem being called, the operations associated with the delayedItem are executed at some point between its call and the close of the tool. It is not
guaranteed that the execution will take place at the point of invoking the method. It puts
the item in an array which is constantly checked when data is being put back into the
parent region. If an operation involving the item is present in the array when its being put
back, the operation is carried out and the changed value is put into the parent region.

5.3 Tool Operators (TOPs)

There are currently two TOPs associated with the Name value Tool namely the
numElements which returns the total number of elements in the region and the numItems
which returns the number of unique items associated with the region. This handles the
scenario where name-value pairs with duplicate names are associated with the region.
Under such circumstances only one instance of the associated name-value pairs is
considered towards the total number of unique items in the region.

5.4 Important Methods

Some of the important methods that provide specific functionality for
manipulating the elements associated with the tool are enumerated below.

- get(<int> index): returns the value of the name-value pair at the specified index.
- get(<string> name): returns the value of the name-value pair specified by the
given name.
- index(<string> name) : returns the index of the name-value pair specified by the
name in the region.
- itemCount(): returns the total number of unique name-value pairs in the region
associated with this tool.
• size(): returns the total number of name-value pairs in the region including duplicate name-value pairs if any.

• name(<int> index): returns the name of the element at the specified index in the region.

• remove(<int> index): removes the element at the specified index from the region. The element gets removed from the parent when data is flushed back to the parent in accordance with the data caching algorithm.

• remove(<string> name): removes the element specified by the given name from the region. The element gets removed from the parent when data is flushed back to the parent in accordance with the data caching algorithm.

• clear(): removes all the elements associated with this tool.

• set(<int> index, <object>value): sets the value of the element at the specified index. The changes made are transferred to the parent in accordance with the data caching mechanism.

• set(<string> name, <object>value): sets the value of the element specified by the given name. The changes made are transferred to the parent in accordance with the data caching mechanism.

• add(<int> index, <string> name, <object> value): inserts a name-value pair at the specified index.

• add(<string> name, <object> value): inserts a name-value pair at the end of the region associated with this tool.
- `copy(<ItemRegion> region)`: copies the contents of the specified region into the region associated with this tool. This effectively replaces the elements associated with the current tool.

    Of the above listed methods, only the sets and the gets are applicable to Element and Item regions since they hold only one name-value pair. Since the All region can potentially hold more than one name-value pair, the other methods are applicable to this region.
6 Adapters

Tools can attach to a pre-defined set of regions directly. For the tool to attach to semantically different regions, the data in the region must be converted into a structured format acceptable to the tool. Adapters perform the conversion of the data across formats. Tools are connected to each other by means of adapters. Since all the conversion of data is done behind the scenes, adapters are generally invisible to the user. When a structured tool like Vector or Name Value tries to attach itself on a region other than OneD region or Item Region respectively, an adapter is required to convert the data. If the region on which the Vector tool is to be attached is a text region, a Text to OneD adapter is required to convert the text into OneD format. In case the region on which the Name Value tool is to be attached is a text region, a Text to Item adapter is required.

Vectors and Name Value tools can also attach to Scalar regions by going through two adapters on top of one another. In order to convert a Scalar region to OneD region, a Text to OneD adapter is constructed on a Scalar to Text adapter. Similarly, to convert a Scalar region into Item region, a Text to Item adapter is constructed on a Scalar to Text adapter. This process of constructing adapters happens behind the scene while the tool is being constructed on a region. Depending on the type of tool and region, the particular tool constructor is chosen. In order to attach to other regions, it is up to the user to supply the adapters necessary for conversion of data.

The generic functionality of the adapters has been implemented in the Abstract Adapter. All other adapters inherit these features and have their own specific features.
The adapter also takes care of maintaining the consistency of data when multiple tools are attached to it. This is done through a data-caching algorithm, which is explained in detail later. This algorithm basically keeps track of the tools, which have manipulated their data and correspondingly updates the parent regions when another tool requests for the data in the same region. Some of the important methods in the Abstract Adapter have been enumerated below with a description of the functionality they achieve.

- **getMoreData():** This method brings data into the current adapter’s data structure by requesting data from its parent. The data obtained from the parent is translated into a format understood by the current adapter and stored in the data structure. This method is repeatedly queried until the data requested from the tool is obtained or until the parent runs out of data.

- **getDataUntil(<object> location, <boolean>spill):** This method calls getMoreData() repeatedly until the location specified is reached. The second argument to this method specifies if the data is to be stored or thrown away as it is obtained from the parent.

- **release(<int> elements):** In case the tool is opened in a SEQUENTIAL mode, a call to this method causes the specified number of elements belonging to this tool to be released and flushed back to the parent. The pointer to the elements released is also adjusted so that further requests for elements prior to this point result in an exception being thrown. In case the tool is opened in a RANDOM mode, nothing is done since all the elements associated with the tool are held in the buffer.
• **flushBuffer():** This method writes data from the adapter back to the parent. The data is retained in the adapter though.

• **spillBuffer():** This method writes data from the adapter to the parent and deletes its data structure.

• **reacquire():** This method is called in order to obtain fresh and current data from the parent. This results in the data being deleted from all the children and also from the adapter’s data structure.

### 6.1 TextToItem Adapter

This adapter is used to convert data from a text format to an Item format. The adapter receives data from the parent region in terms of lines. Each of these lines is then parsed using the Parser in order to obtain the items. The Name-value details are used to parse the lines and retrieve the items. The items and the corresponding names are stored in an internal data structure in the form of an array list. An object of type Item is created for each name-value pair. This object stores the name and the corresponding value. The array list stores these objects for each of the name-value pairs in the region. Whenever the tool creates a ROP or a TOP, this array list in the adapter is queried for the required data. The adapter also handles the task of re-converting the data into its original format and putting it back into its parent region in terms of lines. This is achieved by using the “punctuate” facility available in the Parser. Essentially the role of the Parser in punctuating the items is to convert the items back into the line format so that the parent region can be updated.
6.2 **TextToOneD Adapter**

The Text to OneD adapter is used to convert data from a text format to a OneD format. This adapter receives data from the parent in the form of lines. The Parser is used to extract the elements from these lines. The Parser uses the Vector details in order to parse the lines and extract the elements from the text. These elements are stored in a data structure in the form of an array list in the adapter. Whenever the Vector tool creates a ROP or a TOP, the array list is queried to obtain the required elements. On the same lines as the Text to Item adapter, this adapter also handles the task of converting the data back into the original format. This is done with the help of the “punctuate” facility in the Parser. This basically converts the elements into their respective lines and updates the parent region.

6.3 **ScalarToText Adapter**

This adapter is used to convert the data from a scalar region into a text format. The adapter receives data from the parent in chunks. This is split into lines and stored in an array list in the adapter. This adapter is primarily useful for attaching structured tools on scalar regions created by the ROPs of some other structured tools. Whenever a ROP or a TOP is created, the array list is queried to obtain the corresponding lines. The adapter also handles the task of converting the data back into the original format. The data is written up in chunks as it was received from the parent region.

More adapters can be created and integrated into the environment by the user in order to incorporate custom regions and tools. This allows scope for extensibility. In the
case of custom-defined adapters, the structured tools are constructed using a generic constructor that creates an instance of the required adapter depending on the type of the region and the tool to be created.
7 Data Caching Mechanism

Data consistency and integrity within the Data Exchange framework is handled through a data-caching algorithm. It is worth recollecting that the adapters and the exchangers are the only components in the Data Exchange mechanism that actually store the data. So the efforts to maintain data consistency must be concentrated on these components. The validity of the data in these components is governed by a status variable “cacheState”. This represents the state of the data in the component at any given time. This can take the following values

- INVALID - the data stored in this component is invalid and new data must be acquired from the parent region. The data in this component may have become invalid because some tool on an overlapping region changed the values of its data.
- CLEAN - the data stored in this component is valid and can be used for further processing.
- DIRTY - the tool attached to this component has changed the data stored in it.
- >0 –the component has a number of DIRTY children. Some tool below the current tool attached to this component has altered the data stored in the component.

All the regions have a Boolean flag “hasDirtyChild” that indicates whether the region has DIRTY adapters beneath the child that has a cache state greater than zero.
Whenever the tool makes a request for data through the ROPs, the cache state of the concerned adapter is verified and only if the cache state is CLEAN or DIRTY, the requested data is directly returned. In the event of the cache state being INVALID, the data in the adapter needs to be re-validated. Valid data is obtained by reacquiring the data from the parent region. Once the data is reacquired, the cache state of the adapter is set to be CLEAN. In case the cache state of the adapter is greater than zero, it has DIRTY adapters below it and the data contained in it is not necessarily the most current. In order to ensure that the data in the parent adapter is consistent with the data in the adapters below it, the concerned tools attached to the adapters are flushed. When a tool is flushed, all the associated child tools are also flushed and the data stored in the concerned adapters is updated in the parent region of the tool. Once the tool is flushed the cache state of the concerned adapter is set to be CLEAN. The cache state of the parent is also decremented by one to reflect that there is one less DIRTY adapter below it.

If the tool desires to alter the data in the region associated with it, the behavior is governed by the current cache state of the adapter. It is imperative to ensure that the data in the adapter is current before carrying out any alteration to the data. If the cache state of the adapter is CLEAN or DIRTY, the data is current. If the cache state is INVALID, data needs to be reacquired from the parent. In case there are DIRTY children below the adapter, that is if the cache state is greater than 0, the tools attached to the lower adapters are flushed. After the manipulation of data, the cache state of the adapter has to be updated along with those of its parent. If the adapter is CLEAN, the state is set to DIRTY and the call is sent to its parent. If the parent adapter is CLEAN, its cache state is set to
DIRTY otherwise the cache state is incremented by one to indicate that there is an increase in the number of DIRTY children below it. During this operation, the cache state of adapters associated with tools constructed on overlapping regions is set to be INVALID.
8 Conclusion

The Data Exchanger tool developed currently has two plug-in tools available for parsing data viz. Vector and Name-value. Using these two tools, large text files can be parsed to obtain data, which is in the form of a single dimensional array of elements or exists as name-value pairs separated by delimiters. The value delimiters can be specified as regular expressions also. The data thus obtained can be stored in FIPER parameters in the FIPER environment, modified, and written back into the original data source. This helps the user to sort through copious amounts of information stored in large files to obtain relevant data that can be used elsewhere. This saves quality time in sifting through the large amount of information available to designers today. As a part of the FIPER framework, the Data Exchange tool enables simultaneous exchange of data across different design tools thereby reducing the design cycle times and time-to-market for new products. The FIPER framework enables distributed design and development of products within a web-based environment. It also maintains concurrency between design and manufacturing.

In order to handle some of the other formats of data available to designers like two-dimensional tables, and legacy databases, more plug-ins can be developed and incorporated into the Data Exchanger module. It is envisioned that these plug-ins would be a part of the Version 2.0 of the FIPER framework.
Bibliography


[15] Malone, B., Papay, M., “ModelCenter: An Integration Environment for Simulation Based Design,


Appendix A - A Practical Example of Data Exchanger using Vector

This example illustrates how to use the Vector tool in order to extract useful information from an input source. Although the input source could potentially be anything from a File to a database, this document looks at text files as the input source. Consider an input file named “testInput.txt”. The contents of the file are given below

```
Apple,
Ball, Cat,
Dog,
Elephant, Flower,
Giraffe,
Horse,
Indigo,
Jackal,
```

The first step is to attach a File Exchanger to this file. The File Exchanger may be constructed as follows

```java
FileExchanger fXC = new FileExchanger(
    Exchanger.PUT,
    "testInput.txt",
    "testOutput.txt";
```

where the three arguments to the constructor for File Exchanger are

- Mode – could be one of PUT or GET. Signifies if the output is to be written out into a file or not respectively.
- Input File – the fully qualified name of the source file
- Output File – the fully qualified name of the destination file.

In this case, a File Exchanger has been constructed in the PUT mode with the source and destination files being specified by “testInput.txt” and “testOutput.txt” respectively. Now,
in order to extract the individual elements of the file, a Vector tool could be attached to this region. In order to construct a Vector tool, the details of the tool have to be specified. A typical details object would be constructed as given below

```java
VectorDetail d1 = new VectorDetail();
d1.startIndex = 0;         // beginning index
d1.resizable = true;       // region is resizable
d1.elementDelimiter = ","; // delimiter is comma
d1.EOLisDelimiter = false;
d1.wsChars = " ";          // white space chars.
d1.whiteSpaceIgnored = true;
d1.escape = '\0';          // escape character
d1.openChars = "(";        // open characters
d1.closeChars = ")";        // close characters
d1.quoteChars = "";        // quote character
d1.doubleQuote = false;
```

For a Vector tool to be constructed, it is important to have a region to which the tool can attach. If the semantics of the region are different from what a vector tool can attach to, an adapter, which translates the data in the region into the required format, is constructed automatically. For example, if a vector tool is constructed on a OneD region, no adapters are required. But if a vector tool is attached to a Text region, a Text to OneD adapter is created to translate the data across formats. An instance of the Vector tool may be constructed as follows

```java
Vector v = new Vector(Tool.RANDOM, fXC, d1);
```

where the three arguments to the constructor for the Vector tool are:

- `accessType` – SEQUENTIAL or RANDOM – determines if the data can be accessed unidirectional or bi-directionally.
• region – the region to which the tool is attached. If the region is semantically different from what a vector tool can attach to, an adapter is created internally.

• details – details of the tool like the one described above.

In the above case, a vector tool has been constructed in the RANDOM mode and is attached to the File Exchanger with the details being specified by the details object.

Now, the various ROPs and TOPs can be called on the instance of the Vector tool. The first element of the vector tool may be retrieved by

\[ v.\text{element}(0)\text{.get();} \]

The ROP “element” returns a Scalar Region corresponding to the first element. In order to obtain the actual string value of the element, the method “get” is called on the Scalar Region. It is important to note that the index of the region has been set to begin with 0. It can be inferred that the output of the following command

\[
\text{System.out.println(“The first element: “ + v.\text{element}(0)\text{.get();};} \]

would be

The first element: Apple

The subVector ROP returns a OneD region on which another vector tool could potentially be constructed.

\[ v.\text{subVector}(3, 6)\text{.get(0)} \]

returns the first element of the subVector region, which is essentially the fourth element in the vector v i.e.

\[ \text{elephant} \]
It is possible to get the total number of elements associated with the vector tool using the TOP numElements.

\[ v.\text{numElements}() \]

returns the total number of elements associated with the vector \( v \), which is 10.

The all ROP also returns a OneD region that includes all the elements associated with the vector \( v \). It essentially creates a subVector that goes from 0 to the end of the parent (Locator.EOP).

It is possible to construct another Vector tool \( v_2 \) on a OneD region returned by the ROPs. The following code achieves this

\[
\text{Vector } v_2 = \text{new Vector(Tool.SEQUENTIAL,}
\begin{align*}
&v.\text{subVector(3, 7), } d1); \\
\end{align*}
\]

This essentially creates another vector \( v_2 \) on the region that corresponds to element 3 through 7. It is important to note that element 7 of \( v_1 \) is also a part of the subVector region created. Now a TOP on \( v_2 \) would return 5. That is

\[
\text{System.out.println("Total number of elements: \{} \}
\begin{align*}
&\text{ + v2.numElements());} \\
\end{align*}
\]

returns

Total number of elements: 5

Now suppose the following code is executed

\[
\text{System.out.println("Element 2 in v2: \{} \}
\begin{align*}
&\text{ + v2.element(2).get());} \\
&\text{System.out.println("Element 1 in v2: \{} \}
\begin{align*}
&\text{ + v2. element(1).get());} \\
\end{align*}
\]
\]
This would throw an exception since v2 is sequential and the element 2 has been accessed before element 1 could be accessed. The exception thrown would be

    com.engineous.datex.BackwardMoveException

The following code also would cause an exception to be thrown since v2 does not have more than 5 elements associated with it.

```
System.out.println ("Element 6 in v2: "+ v2.element(6).get());
```

would throw the following exception

    com.engineous.datex.EOPException

which denotes that the end of parent region has been reached and element 6 was still not found.

Elements can also be added or removed from the region using the add() and remove() methods in OneD region.

```
v.subVector(0, 4).add(0, "added element");
```

adds the element at the zeroth position in the subVector region. So the data in the file would be

```
added element, Apple, Ball, Cat, Dog, Elephant, Flower, Giraffe, Horse, Indigo, Jackal,
```

The following code

```
v.subVector(0, 4).add("added element");
```
adds the element at the end of the subVector region. The data in the file would resemble

Apple, 
Ball, Cat, 
Dog, 
Elephant, added element, Flower, 
Giraffe, 
Horse, 
Indigo, 
Jackal,

Now in order to remove an element from the region,

v.subVector(0, 4).remove(3);

The output file after the above operation would be

Apple, 
Ball, Cat, 
Elephant, Flower, 
Giraffe, 
Horse, 
Indigo, 
Jackal,

The entire contents of the region can be deleted using the clear() method.

v.subVector(0, 3).clear();

The output file after the clear operation would be as given below

Elephant, Flower, 
Giraffe, 
Horse, 
Indigo, 
Jackal,

The region consisted of the first three elements of the vector v. The current values of the 
elements can be replaced with new values using the set methods.

v.element (0).set("set Apple");
//third element in subvector but fourth in v
v.subVector(1, 5).set(3, “set Elephant”);
v.all().set(7, “set Horse”);

The output file can be represented as follows

set Apple,
Ball, Cat,
Dog,
set Elephant, Flower,
Giraffe,
set Horse,
Indigo,
Jackal,

The value of the elements can also be copied into FIPER parameters and replaced using the values from FIPER parameters.

v.element(1).read(FIPER Param);
v.element(2).write(FIPER param);

The read() method copies the value of element 1 into the FIPER parameter and the write() method copies the value of the FIPER parameter into the value of element 2. This can be done even for OneD region where the entire region is replaced with the values from the FIPER parameters.
Appendix B - A Practical Example of Data Exchanger using Name-value

This example illustrates how to use the Name-value tool in order to extract information from an input source that contains name-value pairs. Consider the source to be an input file named “testInput.txt”. The contents of the file are given below

A = apple,
B = Ball, C = Cat,
D = Dog,
E = Elephant, F = Flower,
G = Giraffe,
H = Horse,
I = Indigo,
J = Jackal,

As in the earlier example, a File Exchanger is attached to this file.

```java
FileExchanger fXC = new FileExchanger(
    Exchanger.PUT,
    "testInput.txt",
    "testOutput.txt");
```

In order to extract the values of the name-value pairs, a Name-value tool has to be constructed. The associated details object also has to be created.

```java
NameValueDetail nvd1 = new NameValueDetail();
nvd1.wsChars = " ";
nvd1.openChars = "";
nvd1.closeChars = "";
nvd1.quoteChars = "";
//""="" separates name and value
nvd1.valueDelimiter = "=";
nvd1.escape = '\0';
nvd1.whiteSpaceIgnored = true;
nvd1.doubleQuote = false;
nvd1.EOLisDelimiter = true;
```
//comma separates two name-value pairs
nvd1.delimiter = ",";
nvd1.resizable = true;
//throws an error when name-value pairs with
//duplicate names occur in the region.
nvd1.handleDuplicateName = Locator.THROW_ERROR;

A name-value tool can be constructed as follows

```
NameValue nv = new NameValue(Tool.RANDOM, fXC, nvd1);
```

The various ROPs and TOPs associated with name-value tool can be executed on this instance of name-value tool.

```
System.out.println("Element 2 :"
                   + nv.element(2).get());
```

returns

```
Element 2 : Cat
```

The name-value pairs can also be accessed using the names.

```
System.out.println("Element 2 :");
                   + nv.item("C").get());
```

returns

```
Element 2 : Cat
```

The all ROP in name-value returns an Item Region that corresponds to all the elements in the region associated with name-value nv. It is possible to construct another name-value tool on an Item Region.

```
NameValue nv2 = new NameValue(Tool.SEQUENTIAL,
                              nv.all(), nvd1);
System.out.println("Element 2 :");
                   + nv2.item("C").get());
```

returns
Element 2 : Cat

An exception is thrown if the specified name is not found in the list of name-value pairs.

```java
System.out.println("Element 2 :
    + nv2.item("x").get());
```

throws

```java
com.engineous.datex.EOPException
```

If the name specified is null, another exception is thrown.

```java
System.out.println("Element 2 :
    + nv2.item(""").get());
```

throws

```java
com.engineous.datex.NoSuchItemException
```

The following lines of code would throw an exception since nv2 is SEQUENTIAL and item “a” is being accessed after item “d” has been accessed. This constitutes a bi-directional movement, which is not permitted in SEQUENTIAL mode.

```java
System.out.println("Item d :
    + nv2.item("d").get());
System.out.println("Item a:"n    + nv2.item("a").get());
```

throws

```java
com.engineous.datex.BackwardMoveException
```

In order to tide over this problem, Name-value tool provides another ROP called delayedItem. This works as a trade-off between the SEQUENTIAL and RANDOM modes. The tool does not guarantee when the ROP would be executed exactly. The only thing certain about the delayedItem ROP is that it would get executed at some point after
it is called and before the tool closes. This is useful in cases where the name of the name-value pair is known but the sequence of the pairs is not known.

```java
StringBuffer buf = new StringBuffer();
nv2.delayedItem("d").get(buf);
nv2.close();
System.out.println("Item d :" + buf);
```

This prints out the value of item “d” after the tool is closed.

Elements associated with the region can be deleted or new elements may be added into the region.

```java
nv.all().remove(0);
nv.all().remove("D");
```

removes the first element in the region and the element corresponding to the name “D”.

The output file after this operation may be represented as

```
B = Ball, C= Cat,
E = Elephant, F = Flower,
G = Giraffe,
H = Horse,
I = Indigo,
J = Jackal,
```

The following code adds an element to the all region

```java
nv.all().add(0, "addedname", "addedValue");
nv.all().add("addedLast", "value");
```

The resulting output file is given by

```
AddedName = addedValue,A = apple,
B = Ball, C= Cat,
D = Dog,
E = Elephant, F = Flower,
G = Giraffe,
H = Horse,
I = Indigo,
```
J = Jackal, addedLast = value,

The clear() method removes the entire region represented by all.

    nv2.all().clear();

As a result of clear() all the elements associated with nv2 are removed and the file has no elements.

The current values of the name-value pair can be altered also using the set() methods.

    nv.all().set(0, “set apple”);
    nv.element(1).set(“set ball”);
    nv.item(“D”).set(“set dog”);
    nv.all().set(“E”, “set elephant”);

After these set operations the resulting output file can be shown as below

    A = set apple,
    B = set Ball, C= Cat,
    D = set dog,
    E = set elephant, F = Flower,
    G = Giraffe,
    H = Horse,
    I = Indigo,
    J = Jackal,

The value can also be copied into FIPER parameters and replaced using the values from FIPER parameters.

    nv.element(1).read(FIPER Param);
    nv.element(2).write(FIPER param);

The read() method copies the value of element 1 into the FIPER parameter and the write() method copies the value of the FIPER parameter into the value of element 2.
Appendix C - A Practical Example of Data Exchanger using Vector and Name-value

This example illustrates how the Vector and Name-value tools can be put to use in order to extract useful information from an input source generated by Arena®, a simulation tool from Rockwell Software used to model wide-ranging processes. A simple processing system consisting of a turning center, a milling center, a drilling center and a grinding center with parts coming into the system and leaving the system after being processed as shown in Figure 15 was developed in Arena® and the simulation results were stored out into text files. The results are for 25 replications of experiments each of duration 10 hours.

Figure 15 A Simple Processing System
Consider the source to be an input file named “simulation_results.txt”. The contents of the file are given below

6:57:23PM Processes                      March 2, 2004
A Simple Processing System .................Replications:25

Replication 1                      Start Time:          0.10 Stop
Time:                              10.00 Time Units:  Hours

Process Detail Summary

Time per Entity

<table>
<thead>
<tr>
<th></th>
<th>Total Time</th>
<th>VA Time</th>
<th>Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drilling Cen</td>
<td>0.22</td>
<td>0.08</td>
<td>0.13</td>
</tr>
<tr>
<td>Grinding Cen</td>
<td>0.02</td>
<td>0.02</td>
<td>0.00</td>
</tr>
<tr>
<td>Milling Cent</td>
<td>0.08</td>
<td>0.05</td>
<td>0.03</td>
</tr>
<tr>
<td>Turning Cent</td>
<td>0.04</td>
<td>0.03</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Accumulated Time

<table>
<thead>
<tr>
<th></th>
<th>VA Time</th>
<th>Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drilling Cen</td>
<td>8.25</td>
<td>13.29</td>
</tr>
<tr>
<td>Grinding Cen</td>
<td>1.63</td>
<td>0.00</td>
</tr>
<tr>
<td>Milling Cent</td>
<td>4.95</td>
<td>2.62</td>
</tr>
<tr>
<td>Turning Cent</td>
<td>3.23</td>
<td>0.99</td>
</tr>
</tbody>
</table>

Other

<table>
<thead>
<tr>
<th></th>
<th>Number In</th>
<th>Number Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drilling Cen</td>
<td>99.00</td>
<td>99.00</td>
</tr>
<tr>
<td>Grinding Cen</td>
<td>99.00</td>
<td>98.00</td>
</tr>
<tr>
<td>Milling Cent</td>
<td>97.00</td>
<td>99.00</td>
</tr>
<tr>
<td>Turning Cent</td>
<td>96.00</td>
<td>97.00</td>
</tr>
</tbody>
</table>

A file exchanger is attached to this file in a PUT mode in order to extract information from the file and possibly modify it as shown below.

FileExchanger fXC = new FileExchanger(
    Exchanger.PUT,
    "simulation_results.txt",
    "simulation_results_new.txt");
In order to extract the Start Time and Stop Time for the simulation run, a Name-value tool has to be constructed. The associated details object also has to be created.

```java
NameValueDetail nvd1 = new NameValueDetail();
//":" separates name and value
nvd1.valueDelimiter = ":";
nvd1.whiteSpaceIgnored = true;
nvd1.EOLisDelimiter = true;
//whitespace separates two name-value pairs
nvd1.delimiter = " ";
nvd1.resizable = true;
//throws an error when name-value pairs with
//duplicate names occur in the region.
nvd1.handleDuplicateName = Locator.THROW_ERROR;
```

Since the only a part of the file has the name-value format, the name-value tool is attached only to a portion of the file. This is achieved by using a Partitioner, a tool which partitions the input source into chunks based on line numbers or patterns (more description is outside the scope of this work). The name-value tool can be constructed on a partition of the file as follows.

```java
Partitioner p = new Partitioner(
    Tool.RANDOM,
    fXC, null);
NameValue nv = new NameValue(Tool.RANDOM,
    p.partition("Replication 1",
        new LineLocator(-1, "Process Detail Summary")),
    nvd1);
```

The various ROPs and TOPs associated with name-value tool can be executed on this instance of name-value tool.

```java
System.out.println("Start Time:" +
    nv.element(1).get());
```

returns
The name-value pairs can also be accessed using the names.

```java
System.out.println("Stop Time:" +
    nv.item("Stop Time").get());
```

returns

```
Stop Time: 99.19
```

In order to extract the time per entity details associated with the various machining centers, we first partition the file to obtain the corresponding segment of the file and then attach a Name-Value tool to it. The value part of the Name-Value tool can be attached to a Vector tool so that we can retrieve the VA Time and the Wait Time associated with the Drilling Center. This is achieved through the following code.

```java
VectorDetail d1 = new VectorDetail();
d1.startIndex = 0;  //beginning index
d1.elementDelimiter = \"\t\";  //delimiter is tab
d1.EOLisDelimiter = false;
d1.whiteSpaceIgnored = true;
NameValueDetail nvd1 = new NameValueDetail();
nvd1.valueDelimiter = \"\t\";
NameValue nv = new NameValue(Tool.RANDOM,
    p.partition(
        new LineLocator(1, "Total Time"),
        new LineLocator(-1, "Accumulated Time")),
    nvd1);
Vector v = new Vector(Tool.RANDOM,
    nv.item("Drilling Cen"), d1);
```

In the above case, a vector tool has been constructed in the RANDOM mode and is attached to the Scalar Region with the details being specified by the details object. Now, the various ROPs and TOPs can be called on the instance of the Vector tool. It is
important to note that the index of the region has been set to begin with 0. It can be inferred that the output of the following command

```java
System.out.println("VA Time: "+ v.element(1).get());
```

would be

```
VA Time: 0.08
```

The total number of machining centers in the system can be obtained by querying the Name-Value tool to return the number of elements associated with it.

```java
System.out.println("Total number of centers: "+ nv.numElements());
```

returns

```
Total number of centers: 4
```

Alternatively, a Table tool can be attached to a partition on the Exchanger so that, we get data in the form of rows and columns. This is shown below.

```java
TableDelimiter td = new TableDelimiter();
td.elementDelimiter = "\t";
td.EOLIsRowDelimiter = true;
td.startIndex = 1;
Table t = new Table(Tool.SEQUENTIAL,
p.partition(
    new LineLocator(1, "Time per Entity"),
    new LineLocator(-1, "Accumulated Time"),
    td));.
```

Now, the following line

```java
System.out.println("Center: "+ t.cell(1, 1).get());
```

returns
Center: Drilling Cen

And

```java
System.out.println("Total Time: 
  + t.cell(1, 2).get());
```

returns

Total Time: 0.22