



Sonoma Technology, Inc.
Air Quality Research and Innovative Solutions

INTERAGENCY FUELS TREATMENT DECISION SUPPORT SYSTEM SOFTWARE DESIGN SPECIFICATIONS

**Software Specifications
STI-909029.03-3664-SS**

By:

Neil J. M. Wheeler

Judd E. Reed

Kevin D. Unger

Sean M. Raffuse

Steven A. Ludewig

Tami H. Funk

Eric A. Gray

**Sonoma Technology, Inc.
1455 N. McDowell Blvd., Suite D
Petaluma, CA 94954-6503**

**Prepared for:
The Joint Fire Science Program
3833 South Development Avenue
Boise, ID 83705**

May 21, 2010

ACKNOWLEDGMENTS

Many individuals have contributed to the Interagency Fuels Treatment Decision Support System (IFT-DSS) software development effort. We are greatly appreciative of the support and guidance provided by John Cissel, the Joint Fire Science Program (JFSP) program manager. Erik Christiansen, Chair of the Fuels Management Committee (FMC), provided key help and support for analyzing the fuels treatment process that specialists from all agencies struggle with. The JFSP Fuels Treatment Working Group (FTWG)—Michael Beasely, Dennis Dupuis, Mark Finney, Glen Gibson, Randi Jandt, David Peterson, Tessa Nicolet, and Brad Reed—has guided our work and represented the first line of critique and innovative ideas for the project. The JFSP Software Tools and Systems Study Advisory Committee—Pat Andrews, Nate Benson, Mike Hilbruner, Mike Hutt, David Peterson, Carol Saras, Paul Schlobohm, Shari Shetler, and Tim Swedberg—has kept the study headed in the right direction and made sure that all the various components of a successful solution were considered.

We would like to specifically acknowledge the fuels treatment specialists who have agreed to participate in this effort to serve as the IFT-DSS proof of concept (POC) Test User Group—Brad Reed, Brenda Wilmore, Eric Miller, Gary Curcio, Gary Fildes, Gwen Lipp, Jim Roessler, Jon Wallace, Jonathan Olsen, Karen Folger, Mack McFarland, Nikia Hernandez, Perry Grissom, Randi Jandt, Randy Stiplin, Scott Weyenberg, Sean McEldery, Tessa Nicolet, Alison Forrestel, Brian Sorbel, Daniel Rasmussen, Dennis Fiore, Dennis Page, Glenn Gibson, Jeremy Spetter, Joshua Keown, Kim Kelly, Kyle Jacobson, Loretta Duke, Mike Uebel, Sam Amato, Anna Payne, Dave Pergolski, Jennifer Croft, Jeremy Bennet, John Washington, Ken Rodgers, Mathew Weldon, Paul Maday, Rance Marquez, Albert Savage, William Aney, Yanu Gallimore, Lauren Miller, and Kristen Allison—who have provided, and will continue to provide, valuable feedback regarding the functionality and usability of the IFT-DSS. We also thank the broader group of 49 field fuels treatment specialists who helped develop and refine the fuels treatment decision support process.

We would like to acknowledge our appreciation of the fire and fuels science and software development community—Eric Twombly, Mark Finney, Joe Scott, Alan Ager, and Nick Crookston—for their cooperation and feedback related to integrating data and software applications that will be a part of the IFT-DSS. The Fire and Environmental Research Applications (FERA) team of Dave Petersen, Roger Ottmar, Susan Prichard, and Paige Eagle provided support and assistance implementing new software modules into the IFT-DSS. We would also like to acknowledge the managers of the Wildland Fire Decision Support System (WFDSS)—Tom Zimmerman, Rob Seli, and their development team—and the BlueSky Framework—Sim Larkin—for their willingness to work collaboratively to develop software systems that can communicate with one another to create efficiencies in the fire and fuels domain.

We would also like to thank the Information Technology (IT) specialists who helped ensure that we have considered agency IT requirements—John Gebhardt, John Noneman, Laura Hill, and Brad Harwood. All of these groups constitute the large group of stakeholders whose jobs will be positively affected by the results of this project. It will take long-term attention and energy from these groups of stakeholders and others, yet to be engaged, to create an effective community of interest that will ensure that the IFT-DSS helps all of us do our jobs better.

Finally, we would like to thank the other members of Sonoma Technology's (STI) design and development team who contributed to the IFT-DSS design, including Liron Yahdav, Alan Healy, Jason Amador, and Stacy Drury.

VERSION CONTROL

Draft Version 1	July 29, 2009
Draft Version 2	August 18, 2009
Draft Version 3	May 21, 2010

TABLE OF CONTENTS

<u>Section</u>	<u>Page</u>
ACKNOWLEDGMENTS	iii
LIST OF FIGURES	vii
LIST OF TABLES	ix
GLOSSARY	x
EXECUTIVE SUMMARY	ES-1
1. INTRODUCTION.....	1-1
1.1 Purpose	1-2
1.2 Document Organization.....	1-3
1.3 IFT-DSS Naming Convention.....	1-3
1.4 References to Other Relevant Documents.....	1-4
2. SYSTEM OVERVIEW.....	2-1
2.1 Service Oriented Architecture	2-1
2.2 Rationale for a Service Oriented Architecture Approach.....	2-1
2.3 IFT-DSS Users and Stakeholders.....	2-2
2.4 Overview of the Fuels Treatment Decision Support Process.....	2-3
2.5 Overview of the IFT-DSS Architecture.....	2-5
2.5.1 Architectural Components.....	2-5
2.5.2 Key Architectural Features and Functions	2-7
3. THE IFT-DSS USER EXPERIENCE.....	3-1
3.1 Logging into the IFT-DSS	3-1
3.2 Phase I – Project Setup and Planning	3-3
3.3 Phase II – Software Model Execution and Iterative Analysis	3-5
3.4 Phase III – Project Finalization, Documentation, and Archive	3-8
4. TECHNICAL SOFTWARE DESIGN	4-1
4.1 Components	4-4
4.1.1 Models.....	4-4
4.1.2 Model Adaptors.....	4-6
4.1.3 Scientific Database	4-13
4.1.4 Data Interface	4-14
4.1.5 Executive.....	4-15
4.1.6 Control Database.....	4-18
4.1.7 Navigator.....	4-22
4.1.8 Project and Planning Database.....	4-23
4.1.9 Planner Session Engine.....	4-24

<u>Section</u>	<u>Page</u>
4.2 Interfaces.....	4-25
4.2.1 Model Data Exchange Interface	4-26
4.2.2 Model Control Interface	4-26
4.2.3 Sequencer Control and Monitoring Interface	4-27
4.2.4 Executive Scenario Polling and Feedback Interface	4-29
4.2.5 Executive Launching and Progress Reporting Interface	4-29
4.2.6 Map and Data Presentation Interface	4-30
4.2.7 Web Page Generator and Database Interface	4-31
4.3 Behaviors and Interactions	4-31
4.3.1 Project Workflow	4-32
4.3.2 Executive Scenario Compilation	4-33
4.3.3 Executive Orchestration of Model and Scientific Database Interaction	4-33
4.3.4 User Interaction with the Graphical User Interface and Database	4-34
4.3.5 User Interaction with the Web Map Service and the Scientific Database..	4-36
5. TECHNOLOGIES	5-1
5.1 Models and Model Adaptors	5-1
5.2 Scientific Database	5-2
5.3 Data Interface and Executive.....	5-2
5.4 Planning and Control Databases.....	5-3
5.5 Navigator and Planner	5-3
5.6 Hardware.....	5-3
5.7 Future Software Topology for the IFT-DSS.....	5-3
6. REFERENCES	6-1

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
ES-1 Illustration of the five key architecture components of the IFT-DSS.....	ES-3
ES-2 Overall structure of the IFT-DSS depicting the system’s major components and software interfaces	ES-5
1-1. Naming convention for the IFT-DSS.....	1-3
2-1. Work flow diagram illustrating how the fuels treatment planning work flow scenarios fit together from a process perspective	2-5
2-2. Illustration of the five key architecture components of the IFT-DSS.....	2-7
3-1. The IFT-DSS user login screen.....	3-2
3-2. The IFT-DSS POC user home page.....	3-2
3-3. The “My Profile” screen where users manage their profile information.....	3-3
3-4. The options available on a user’s home page	3-3
3-5. The IFT-DSS POC “Manage Projects” project list and project setup screen.....	3-4
3-6. The Project Details screen in the IFT-DSS.....	3-4
3-7. The Run Details screen of the IFT-DSS	3-5
3-8. The dynamically generated tabbed navigation bar and Action Graph for a sample modeling scenario	3-6
3-9. Sample outputs from FlamMap in tabular form	3-7
3-10. Sample outputs from FlamMap in graphical form.....	3-7
3-11. Screen shot of the map viewer within the IFT-DSS GUI and the map viewer within the IFT-DSS displaying fire behavior output data from FlamMap.....	3-8
4-1. Overall structure of the IFT-DSS depicting the system’s major components and software interfaces	4-2
4-2. Screen shot of the CDL tool.....	4-5
4-3. Illustration of the three model hosting methods by which models can be integrated into the IFT-DSS and function as services	4-7
4-4. UML diagram of the IFT-DSS model communication infrastructure	4-9

<u>Figure</u>	<u>Page</u>
4-5. Conceptual ERD for the Control Database.....	4-19
4-6. Conceptual ERD for subset of the Control Database to facilitate efficient model execution.....	4-20
4-7. Conceptual ERD of user interface HTML templates.....	4-21
4-8. Subcomponents of the Navigator.....	4-22
4-9. Conceptual ERD for Project and Planning Database.....	4-24
4-10. Overview of run execution.....	4-32
4-11. Swim lane diagram of the Navigator, Executive, and Control Database interaction	4-33
4-12. Swim lane diagram of Executive, Data Interface and the Scientific Database, and models interaction.....	4-34
4-13. Swim lane diagram of the user, Navigator, page creator, and control database interaction	4-35
4-14. Swim lane diagram of the user, WMS, and Scientific Database interaction	4-36
5-1. Software topology for the first production release of the IFT-DSS scheduled for release in June 2011	5-4

LIST OF TABLES

<u>Table</u>	<u>Page</u>
1-1. IFT-DSS naming convention during development.	1-4
4-1. Pseudocode for key member functions to the output stream class.	4-10
4-2. Pseudocode of the readObject member function of the input stream class.	4-11
4-3. Pseudocode of the model hosting program.	4-12
4-4. Brief pseudocode for the Executive program.	4-15
4-5. Pseudocode for compiler component of the Executive.	4-17
4-6. Messages between the Executive and the Model Adaptors.	4-27
4-7. Messages between the Executive and Data Interface.	4-28

GLOSSARY

TERM	DESCRIPTION
API	Application programming interface
Area of interest (AOI)	For IFT-DSS, a scale-independent unit of area defined by a user. Within an area of interest, project areas and vegetation units can be defined for analysis. There is no minimum size unit but a maximum unit will be limited to one million acres (approximately 400,000 hectares)
BlueSky Framework	An SOA system to facilitate predictions of smoke emissions and air quality impacts from fires
Consume	A fire effects prediction model. It uses fuel loadings, fuel moisture, and weather variables to predict fuel consumption, particulate emissions, and heat energy released under prescribed fire and wildfire conditions.
DAG	Directed Acyclic Graph (used to represent scenarios in the IFT-DSS)
.dll	dynamic link library
Distributed system	A system whose components can be distributed across a computer network in geographically different locations
ERD	Entity Relationship Diagram
Executive	The component that compiles scenario segments and directs the execution of models
FCCS	Fuel Characteristic Classification System
FOFEM	First Order Fire Effects Model; a set of fire effects prediction models. FOFEM uses fuels and vegetation information to provide estimates of fuel consumption, tree mortality, soil heating, and particulate emissions.
FRAMES	Fire Research and Management Exchange System
FSVeg	A USDA Forest Service database that contains point and plot vegetation data from field surveys such as Forest Inventory Assessment (FIA) exams, stand exams, forest inventories, and regeneration surveys. It includes data for trees, surface cover, understory vegetation, and downed woody material.
FVS	The USDA Forest Service's Forest Vegetation Simulator, a framework for modeling forest growth
Fan-out/fan-in	The maximum number of digital inputs that can be sent (fan-out) or accepted (fan-in)
FlamMap	A fire behavior mapping and analysis program that computes potential fire behavior characteristics (spread rate, flame length, fireline intensity, etc.) over an entire landscape for constant weather and fuel moisture conditions.
Fuels treatment	For the IFT-DSS, any mechanical, silvicultural, or burning activity whose main objective is to reduce fuel loadings or change fuel characteristics to lessen fire behavior or burn severity.
GIS	Geographic information system

TERM	DESCRIPTION
GUI	Graphical user interface
IFT-DSS	Interagency Fuels Treatment Decision Support System
I/O	Input/output
Interface	The programming mechanism that allows software applications to communicate with one another
IT	Information technology
JFSP	Joint Fire Science Program
LANDFIRE	Landscape Fire and Resource Management Planning Tools Project, a mapping project and database of vegetation, fire, and fuel characteristics
.lcp	Landscape File format
Multiplex	To send multiple signals simultaneously in one complex signal and recover the individual signals at the receiving end
NEPA	National Environmental Policy Act
NEXUS	Crown fire hazard analysis software that links separate models of surface and crown fire behavior to compute indices of relative crown fire potential.
NIFCG	National Interagency Fuels Coordinating Group
NWCG	National Wildfire Coordinating Group
Navigator	The component that allows users to operate the system from a web browser
POC	Proof of concept
Parent class	A generalized “superclass” containing specialized subclasses or “child” classes
Project and Planning Database	A database for storing administrative data about fuels treatment projects
Pseudocode	An outline of step-by-step computer programming instructions that is not written in a particular programming language
RDBMS	Relational database management system
RPC	Remote procedure call
SQL	Structured Query Language, used in managing databases
STS Study	Software Tools and Systems Study
Scenario Database	The component that stores representations of the scenarios that are run
Service Oriented Architecture (SOA)	An underlying structure that allows loosely connected components of a computer system to communicate, thus allowing services to be added or changed without creating a completely new structure
Software interface	The mechanism by which software components interact and communicate with one another.
Spatial Data Interface	The component that relays data between the Scientific Database and models
UML	Unified Modeling Language
WFDSS	Wildland Fire Decision Support System
WMS	Web Map Service

EXECUTIVE SUMMARY

ES.1 INTRODUCTION

In May 2009, the Joint Fire Science Program (JFSP) initiated Phase III of the Software Tools and Systems (STS) Study. Phase III of the STS Study involved the development of a software design for the Interagency Fuels Treatment Decision Support System (IFT-DSS) and the implementation of a proof of concept (POC) system for the IFT-DSS. This document describes the software design for the IFT-DSS and will remain a living document throughout the development of the IFT-DSS. The IFT-DSS POC is available on the Internet¹ and provides a user-friendly software system to manage a subset of the most commonly used software tools and data to perform fuels planning scenarios. A key goal of the IFT-DSS POC is to demonstrate that a well-designed, extendable, service oriented architecture software (SOA) framework can help organize and manage the many existing data sets, software models, and tools in the fire and fuels domain and can help foster collaboration within a community of stakeholders. Ultimately, it is the goal of the IFT-DSS program to change the software development and deployment process within the fire and fuels domain to create efficiencies and leverage services among several large distributed SOA systems (e.g., the BlueSky Smoke Modeling Framework (BlueSky), the Wildland Fire Decision Support System (WFDSS), and the Fire and Fuels Application (FFA)).

ES.2 THE FUELS TREATMENT PLANNING PROCESS

During Phase II and early in Phase III of the STS Study, many efforts were made to understand the decision support needs and the workflow processes involved in fuels treatment planning and management. As a result of these efforts, the following six workflow scenarios were identified:

- **Data acquisition and preparation** involves collecting and preparing the vegetation data needed for input into fire behavior and fire effects models.
- **Strategic planning** involves identification of high fire hazard areas within an area of interest. The focus is to identify where further treatment analysis may be warranted on the basis of potential fire hazard.
- **Spatially explicit fuels treatment assignment** involves (1) simulating fuels treatment placement in areas of high fire hazard within an area of interest; and (2) simulating post-treatment influences on fire behavior and fire effects potentials. The spatially explicit fuels treatment assignment extends the strategic planning analysis to applying treatments on the landscape.
- **Fuels treatment effectiveness over time** involves the evaluation of the temporal durability of fuels treatments; that is, how long, in years to decades, a treatment will continue to affect potential fire behavior and fire effects within an area of interest. This workflow scenario naturally follows the strategic analysis and fuels treatment assignment workflow scenario.

¹ <http://iftdss.sonomatech.com>

- **Prescribed burn planning** involves preparing the information needed to plan, document, and conduct a proposed prescribed fire.
- **Risk assessment** involves conducting a probabilistic risk assessment for fuels treatment planning.

ES.3 IFT-DSS SOFTWARE ARCHITECTURE

The findings of the work performed during Phase I of the STS Study indicated that an SOA approach would best serve the fuels treatment community. Several key strategic-level requirements warrant a web-based, distributed SOA approach:

- The need for a system that can be easily accessed and used by fuels treatment specialists from a variety of different agencies.
- The need for a system that can organize, integrate, and manage existing and future fire and fuels software applications.
- The need for a system that will support distributed collaboration and allow fuels treatment planners to perform ad hoc analyses customized for a particular location and/or situation.

The core of the IFT-DSS software architecture consists of five elements: (1) a multi-layered graphical user interface (GUI), (2) a control database, (3) an executive application, (4) data and software services, and (5) a scientific database. **Figure ES-1** illustrates the five key components of the architecture and their relative architectural arrangement. Each component contains interconnected subcomponents, or layers, that perform specific functions within the system. The GUI and the back-end system components and their behavior are described in Sections 3 and 4 of this document.

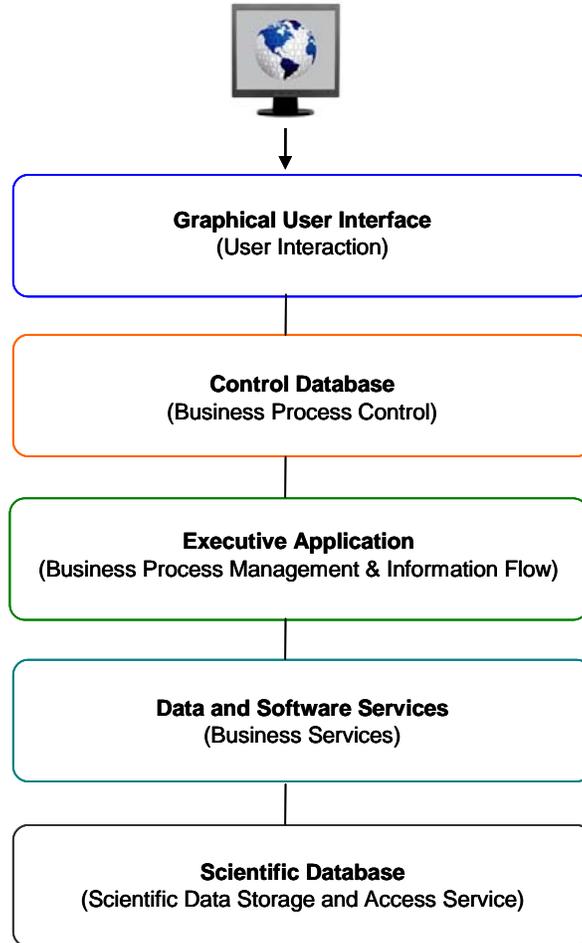


Figure ES-1. Illustration of the five key architecture components of the IFT-DSS.

The IFT-DSS architecture was designed to support the following key features:

- The IFT-DSS should make fuels treatment planning easier by
 - allowing users to acquire, create, and transform input data easily;
 - providing data choices: treelist, LANDFIRE grids, user supplied;
 - allowing users to view and edit spatial and tabular data (inputs and outputs);
 - organizing fuels treatment planning analysis steps and software tools; and
 - recognizing user errors and explaining alternate action.
- The IFT-DSS should make fuels treatment planning more scientifically robust by
 - providing guidance regarding data and model choices based on the scale and type of analysis being performed;
 - allowing users to publish and share analysis methods and algorithms;
 - providing a mechanism to perform sensitivity and iterative analyses;

- providing a mechanism to easily incorporate new models and tools as they are developed; and
- providing quality control, documentation, and audit-trail information to meet regulatory reporting requirements.

ES.4 IFT-DSS SOFTWARE DESIGN

The IFT-DSS will serve as a software framework to integrate vegetation data, vegetation simulators, fire behavior and effects models, and risk analysis tools. It will support the reuse of IFT-DSS applications and services over the Internet, and it will be a flexible, modern, web-friendly system. In designing the IFT-DSS, both current and future needs of the user communities have been considered, and therefore the IFT-DSS can be implemented as a general framework for scientific modeling and analysis. The IFT-DSS is designed in a manner that is generic and almost completely independent of programming language, hardware, and operating system implementation decisions; that is, the design can be realized in a variety of ways using any one of many hardware and operating system configurations. It is important to realize that this design was constructed in a way so that system components can be developed in different languages and use different operating foundations while still conforming to this design.

Figure ES-2 illustrates the overall system design including system components and interfaces, and their relationships. The IFT-DSS is described in terms of nine components and seven interfaces. The nine components are:

- A. **Models** – The scientific and computational components of the system; all other components support the model operations
- B. **Model Adaptors** – Enable integration of different models into the system
- C. **Scientific Database** – Stores the actual data inputted to and outputted from models
- D. **Data Interface** – Relays data between the scientific database and the models
- E. **Executive** – Directs the execution of the models
- F. **Control Database** – Stores representations of how different “runs” of models are orchestrated
- G. **Navigator** – Visualizes spatial data, allows users to edit data values for model inputs, and allows users to execute runs through scenarios or parts of scenarios, all from a web browser
- H. **Project and Planning Database** – Stores administrative data about fuels treatment projects
- I. **Planner Session Engine** – Enables users to manage fuels treatment projects from a web browser

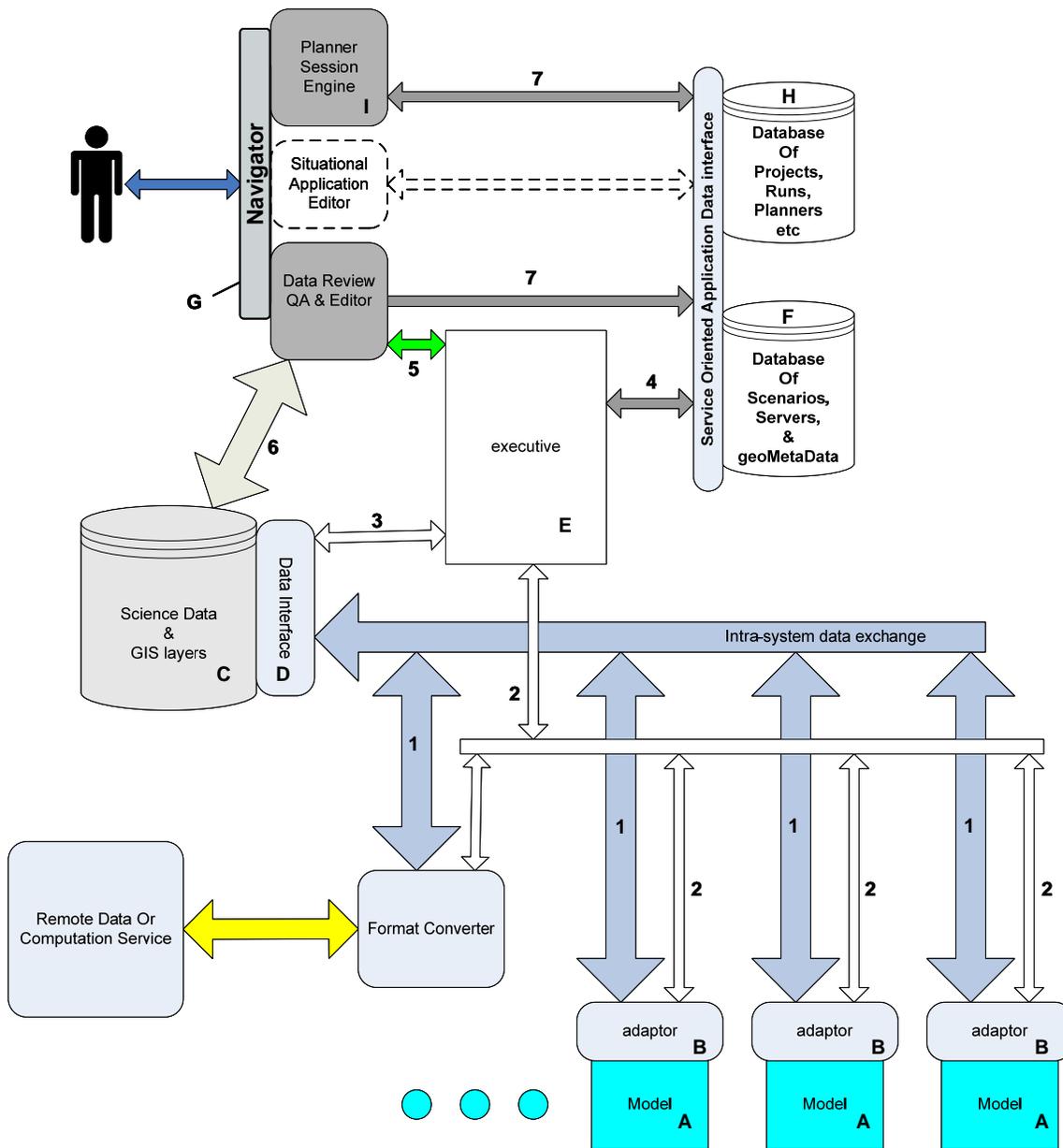


Figure ES-2. Overall structure of the IFT-DSS depicting the system's major components (with lettered indicators) and software interfaces (numbered).

The seven interfaces are:

1. **Model Data Exchange** – Connections between machines running models to relay model inputs and outputs
2. **Model Control** – Connections between the Executive and model adaptors to prepare the model data exchange connections

3. **Sequencer Control and Monitoring** – Interaction between the Data Interface and Executive to manage scenario execution
4. **Executive Scenario Polling and Feedback** – Executive queries of the Control Database for required data, and feedback about model and machine execution performance
5. **Executive Launching and Progress Reporting** – Web application signaling of the Executive to compile and run a scenario segment and the Executive reporting progress back to the web application
6. **Map and Data Presentation Interface** – Communication between the Scientific Database and the web application components
7. **Web Page Generator and Database Interface** – Web application queries of the Control Database to produce the web pages displayed to the user for model inputs and outputs

This design supports not only the immediate needs of the fuels treatment planning community but also supports two key requirements: (1) fuels treatment planners must be able to collaborate and share analysis methods, and (2) the science and model development community must be able to dynamically publish new data and applications to the system. In addition, this design is flexible and extensible to support future planning needs, new sources of data, and new applications as they evolve.

1. INTRODUCTION

In May 2009, the Joint Fire Science Program (JFSP) initiated Phase III of the Software Tools and Systems (STS) Study. Phase III of the STS Study involved the development of a proof of concept (POC) system for the Interagency Fuels Treatment Decision Support System (IFT-DSS). The IFT-DSS POC provides a user-friendly software system to manage a subset of the most commonly used software tools and data to perform fuels planning scenarios with the goal of demonstrating the usefulness and feasibility of the IFT-DSS. The development period for the IFT-DSS POC (Phase III of the STS Study) spanned approximately one year beginning in May 2009. This document contains the software design specifications for the IFT-DSS POC system, Version 0.3.0, as of May 2010.

The increasing operational complexity and urgency in fire and fuels management, coupled with a proliferation in the number of decision support tools available, have driven the need for a transformative solution. A distributed Service Oriented Architecture (SOA) approach that is readily accessible to users across different agencies has the potential to positively transform the development and deployment of data and software tools for fire and fuels management by organizing and coordinating the interaction of independently operating data services and software tools.² When developed with input and acceptance from all stakeholder communities, SOA systems can improve the exchange of information and promote collaboration among stakeholder communities. These systems can streamline the decision support process, facilitate improvements and advancements in fire and fuels science, and reduce the barriers that hinder the adoption of sophisticated risk management science concepts and practices.

Several SOA systems currently exist in the fuels treatment planning domain, and they are “distributed” and service-oriented to varying degrees. While these existing systems all contain useful science and link disparate software applications, some of the systems are inaccessible or require expert knowledge, and none of them individually supports the full range of fuels treatment planning activities. In addition, these systems have not been widely adopted; thus, their user groups are relatively small, and they have not been deployed in a way that facilitates interagency collaboration.

The overall IFT-DSS development effort is intended to demonstrate that a well-designed SOA approach, combined with community development efforts, can improve the decision support process and underlying science in the fire and fuels domain. The overall goal of the IFT-DSS is to develop an extendable software framework for organizing and managing the many existing data sets, software models, and tools available for fuels treatment planning and analysis and to foster collaboration within a community of stakeholders. Ultimately, it is the goal of the IFT-DSS program to change the software development and deployment process within the fire and fuels domain to create efficiencies and to leverage services among several large distributed SOA systems including the BlueSky Smoke Modeling Framework (BlueSky), the Wildland Fire Decision Support System (WFDSS), and the Fire and Fuels Application (FFA).

² Refer to the Carnegie Mellon Software Engineering Institute (SEI) final report prepared for the JFSP for a definition and discussion about SOA at http://frames.nbii.gov/documents/jfsp/sts_study/palmquist_sei_report_2008.pdf

The IFT-DSS software architecture was designed to provide the following benefits to its various stakeholder communities:

- Integration, guidance, and collaboration regarding the use of existing data, software models, and tools for fuels treatment analysis and planning.
- Increased productivity and efficiency in the fuels treatment planning process through a system that can greatly reduce the time associated with preparing and manipulating data and software applications.
- A framework that facilitates peer review and model validation for scientific algorithms, applications, and vegetation data, along with a more efficient review, critique, and feedback mechanism to improve the scientific work flow and decision support process.
- A central framework that meets the needs of users, scientists and software developers, IT security specialists, and managers equally well, organizing a myriad of software systems in a functionally effective, user-friendly manner while allowing IT administrators to provide appropriate security and access.
- Another vehicle to enhance interagency functionality and collaboration and to serve as a proving ground for identifying and testing acceptable governance issues that best support interagency operations.

To achieve these goals, an SOA approach was taken in designing the IFT-DSS architecture. SOA facilitates the integration of disparate software systems by separating functions into distinct units, or services that can be made accessible across a computer network (distributed) so that users can combine and reuse individual services as needed. A key characteristic of a distributed SOA is the ability of users to have choices and control over the data and software applications that are applied to address a specific situation.³

1.1 PURPOSE

This document is intended to serve two purposes: (1) to provide background and an overview of the IFT-DSS, and (2) to provide technical design specifications for the software implementation of the system. Sections 1, 2, and 3 of this document are intended for the end user of the IFT-DSS, with Sections 1 and 2 providing an introduction to and background of the STS Study and an overview of the IFT-DSS, and Section 3 containing a discussion of the graphical user interface (GUI) for the system. Sections 4 and 5 are primarily for the use of the IFT-DSS software development team and contain the technical design specifications for the IFT-DSS.

This document will be updated throughout the IFT-DSS development process to reflect design changes that may occur. It is a living document and will eventually serve as the final technical documentation for the IFT-DSS.

³ Palmquist, 2008 (pages 17-18), http://frames.nbii.gov/documents/jfsp/sts_study/palmquist_sei_report_2008.pdf.

1.2 DOCUMENT ORGANIZATION

This document is organized into six main sections.

- **Section 1 – Introduction:** Provides background on the IFT-DSS and describes the purpose of this document.
- **Section 2 – System Overview:** Provides an overview on the user community, anticipated work flows, and the system architecture.
- **Section 3 – The IFT-DSS POC User Experience:** Provides examples and a discussion of the IFT-DSS GUI.
- **Section 4 – Technical Software Design:** Provides a description of the IFT-DSS software design.
- **Section 5 – Technologies:** Provides a discussion of technologies used in the development of the IFT-DSS.
- **Section 6 – References:** Provides a list of references cited in the document.

In addition to these sections, this document includes an executive summary and a glossary. Because this document, and especially Section 4, contains many technical terms, the reader may find the glossary particularly helpful. The glossary is located near the beginning of this document just before the executive summary.

1.3 IFT-DSS NAMING CONVENTION

There will be several versions of the IFT-DSS as development progresses. **Figure 1-1** illustrates and describes the naming convention for the IFT-DSS.

$$\text{IFT-DSS v } n_1.n_2.n_3.n_4$$

Figure 1-1. Naming convention for the IFT-DSS where n_1 indicates a proof of concept (POC) or production release (POC releases are indicated with a 0 and production releases are indicated by 1, 2, 3, etc.); n_2 indicates a test user group release; n_3 indicates an internal test release; and n_4 indicates an agile cycle release for the development team.

Table 1-1 defines the IFT-DSS naming convention for the past releases and for the next two years of development. Note that interim versions of the IFT-DSS may contain upgrades to the versions indicated in Table 1-1.

Table 1-1. IFT-DSS naming convention during development. Note that there will be interim releases between the major releases indicated in this table.

IFT-DSS Version	Description
IFT-DSS v 0.1.0.0	The POC system released in January 2010.
IFT-DSS v 0.2.0.0	The POC released in April 2010.
IFT-DSS v 0.3.0.0	The POC system described in this document released in May 2010.
IFT-DSS v 1.0.0.0	Initial beta release to be hosted by the Forest Service, currently scheduled for implementation in May 2011.
IFT-DSS v 2.0.0.0	Initial production version scheduled to be released in May 2012.

1.4 REFERENCES TO OTHER RELEVANT DOCUMENTS

Several documents have been produced throughout the STS Study that are relevant to this document. All STS Study documentation can be accessed through the Fire Research and Management Exchange System (FRAMES) website. The following documents are of particular relevance to this document:

- Working Summary of the SEI’s Engagement with the Joint Fire Science Program (Palmquist, 2008).
- The Interagency Fuels Treatment Decision Support System Software Architecture (Funk et al., 2009),
- Refined Work Flow Scenarios for the Interagency Fuels Treatment Decision Support System (Drury et al., 2009).

2. SYSTEM OVERVIEW

Any number of architectural approaches could have been proposed for the IFT-DSS; however, there are key requirements that the IFT-DSS must meet that warrant a web-based, distributed SOA approach:

- The need for a system that can be easily accessed and used by fuels treatment specialists from a variety of different agencies.
- The need for a system that can organize, integrate, and manage existing and future fire and fuels software applications.
- The need for a system that will support distributed collaboration and allow fuels treatment planners to perform analyses customized for a particular location and/or situation.

This section provides a strategic-level overview of the fuels treatment decision support process and the IFT-DSS architecture, including the rationale for the design approach and a discussion of the key components of the system. A complete list of system requirements for the IFT-DSS was developed as part of the architecture design process (Phase II of the STS Study) and can be found in the document entitled “The Interagency Fuels Treatment Decision Support System Software Architecture” (Funk et al., 2009),

2.1 SERVICE ORIENTED ARCHITECTURE

The term SOA is not an alias for a particular system; rather, it describes a particular way in which a system is constructed. For our purpose, SOA is defined as a generic software architecture framework designed to support a collection of services (databases and software applications) with well-defined software interfaces.⁴ SOA facilitates the integration of new and legacy software applications to streamline work processes. This architectural approach can also support inter-operability with other decision support systems in the fire and fuels domain such as BlueSky and the WFDSS. A “distributed” SOA is a system whose components are (or can be) distributed across a computer network; that is, the services within the system can reside in geographically different locations and can be accessed across a network. A web-based, distributed SOA is a system that is accessible to users and controlled through a standard web browser (e.g., Internet Explorer, Firefox) interface.

2.2 RATIONALE FOR A SERVICE ORIENTED ARCHITECTURE APPROACH

The findings of the work performed during Phase I of the STS Study indicated that an SOA approach would best serve the fuels treatment community. Several key strategic-level requirements warrant a web-based, distributed SOA approach:

⁴ A software interface is the programmatic mechanism that allows components (software applications) to communicate with one another so that data and services can be accessible and interoperable within an SOA framework.

- **The need for a system that can be easily accessed and used by fuels treatment specialists from a variety of different agencies.** Government and state agencies often have rules and regulations regarding installation of new software on government computers and workstations. These regulations make it difficult to install software applications directly onto agency desktop computers. Therefore, to make the IFT-DSS accessible across agencies, the system must be able to run on a standard desktop (or laptop) computer with an Internet connection—eliminating the need for users to install new software on their local computers. This feature requires that the system be hosted on an accessible server and developed to function within the most commonly used web browsers (e.g., Internet Explorer, Firefox).
- **The need for a system that can organize, integrate, and manage existing and future fire and fuels software applications.** Existing fire and fuels software applications perform a variety of functions and can be combined to perform complex simulations. The variety of existing applications presents an integration challenge, as these applications were developed with different goals and requirements, at different times, and by different organizations. In addition, these applications were written in different software languages, may currently run on different operating systems, have overlapping functionality, and require differing data formats. The SOA approach evolved to address the issue of integrating such disparate services and software applications. The design of the IFT-DSS software architecture is driven by the requirement to enable disparate applications to work together (including applications not yet developed) while requiring minimal additional effort from application developers to support the framework. This objective can be achieved by designing an SOA architecture that is adaptable and generic enough to accommodate a broad variety of applications and functionality.
- **The need for a system that will support distributed collaboration and allow fuels treatment planners to perform ad hoc analyses customized for a particular location and/or situation.** One of the findings from Phase I of the STS Study was that fuels management and risk mitigation require a distributed approach to collaboration and present an ongoing need for data fusion. Because of the variety of operational contexts, it is impossible to know the exact sets of models, tools, or data needed for every fuels treatment planning situation. Therefore, customizable approaches are needed, requiring collaborative tools that support web-enabled methods of analysis. A flexible and extendable software framework will allow tool developers or sophisticated users to rapidly configure, calibrate, or extend web-enabled capabilities to meet the needs of a specific operational situation.

2.3 IFT-DSS USERS AND STAKEHOLDERS

A major goal of the overall IFT-DSS is to develop a community of individuals from multiple agencies and organizations that can collaborate, exchange, and communicate science and information related to fuels treatment analysis and planning. Collaboration among the fire and fuels community is important to improve the science and understanding of fuels treatment planning and to keep the data, software applications, and the IFT-DSS updated to meet current and future needs. In addition, collaboration helps all agencies and organizations learn from each other about methods, challenges, and approaches for fuels treatment planning.

Through the efforts of Phase II of the STS Study, five stakeholder groups were identified for the IFT-DSS:

1. Approximately 1,000 **fire and fuel operations managers**, or fuels treatment specialists, at multiple federal and state agencies throughout the United States. Fuels treatment specialists will be the primary users of the system for year-round fuels treatment planning.
2. Several (20 to 30) **scientist developers** who will provide new or updated science, models, and tools to the system. Scientific collaborators will be periodic contributors to and users of the IFT-DSS.
3. **Database developers** who will provide applications and data to the system, including:
 - a few (2 to 5) institutional application and data providers who will make large-scale databases available to the system (e.g., LANDFIRE and FSveg); and
 - fuels treatment specialists who will upload or create local data sets.
4. **Information technologists and software specialists** who will operate and maintain the system over time.
5. **Agency senior management**, including the National Wildfire Coordinating Group (NWCG), who are responsible for issues related to business needs, resource allocation and prioritization, financial investments, and the operational efficiency and effectiveness of the community as a whole.

A community development effort to engage each of the five stakeholder groups listed above is being conducted in parallel with the IFT-DSS POC software development effort. The purpose of the community development effort is to gain the support and acceptance of a network of stakeholders that gain valuable services from the IFT-DSS and have responsibilities for the on-going functional and maintenance aspects of the system. In particular, the community development strategy (a) describes the various stakeholder communities and their characteristics; (b) presents a plan for enhancing awareness about and use of the IFT-DSS software by various subgroups of these stakeholder communities; and (c) provides a roadmap of how the IFT-DSS development team proposes to transition the software from the originators, JFSP and the NIFCG, to the USDA Forest Service, the Managing Partner for the NWCG.⁵

2.4 OVERVIEW OF THE FUELS TREATMENT DECISION SUPPORT PROCESS

At the most basic level, the fuels treatment analysis and planning process involves performing environmental assessments of fuels treatment options as mandated by the National Environmental Policy Act (NEPA). This decision process centers on managing outcomes by modifying vegetation. The decision support process involves preparing a detailed vegetation data set; modeling vegetation changes based on growth, treatments, and/or disturbance; and

⁵ For more specific information regarding the community development plan, refer to the FRAMES website at <http://frames.nbj.gov>.

analyzing the results of the modeled vegetation. A fuels treatment specialist then recommends which treatment option to apply.

During the past two years, many efforts have been made to understand decision support needs and the work flow processes involved in fuels treatment planning and management. These efforts included surveying the fuels treatment planning community; conducting personal interviews with several fuels treatment specialists representing different land management agencies; engaging and soliciting feedback from the Interagency Fuels Treatment Work Group (IFTWG); and conducting meetings and discussions with fire and fuels software application and data developers. As a result of these efforts, the following six work flow scenarios have been identified:

- **Data acquisition and preparation work flow scenario** provides a simple and efficient way to collect and prepare the vegetation data needed for input to fire behavior and fire effects models.
- **Strategic planning work flow scenario** enables identification of high fire hazard areas within an area of interest. The focus of this work flow scenario is to identify where further treatment analysis may be warranted on the basis of potential fire hazard.
- **Spatially explicit fuels treatment assignment work flow scenario** (1) simulates fuels treatment placement in areas of high fire hazard within an area of interest, and (2) simulates post-treatment influences on fire behavior and fire effects potentials. The spatially explicit fuels treatment assignment work flow scenario extends the strategic planning analysis to applying treatments on the landscape.
- **Fuels treatment effectiveness over time work flow scenario** enables the evaluation of the temporal durability of fuels treatments, that is, how long, in years to decades, a treatment will continue to lower potential fire behavior and fire effects within an area of interest. This work flow scenario naturally follows the strategic analysis and fuels treatment assignment work flow scenario.
- **Prescribed burn planning work flow scenario** provides the information needed to plan, document, and conduct a proposed prescribed fire.
- **A risk assessment work flow scenario** provides a probabilistic risk assessment for fuels treatment planning.

The work flow scenarios presented here can be divided into two categories: (1) fuels treatment work flow scenarios and (2) a prescribed burn planning work flow scenario. The prescribed burn planning scenario is often considered one phase of a fuels treatment scenario because prescribed burning is one way to treat fuels. However, the development of a prescribed burn plan is a long and complex process, and fuels treatment specialists have indicated that the IFT-DSS could provide a useful service by supporting a work flow scenario specifically devoted to prescribed burn planning.

The work flow scenarios defined here are not mutually exclusive and generally build on one another. The IFT-DSS architecture is designed to be flexible and scalable so that new work flow scenarios can be implemented as they evolve. **Figure 2-1** is a work flow diagram showing how the six work flow scenarios fit together from a process perspective.

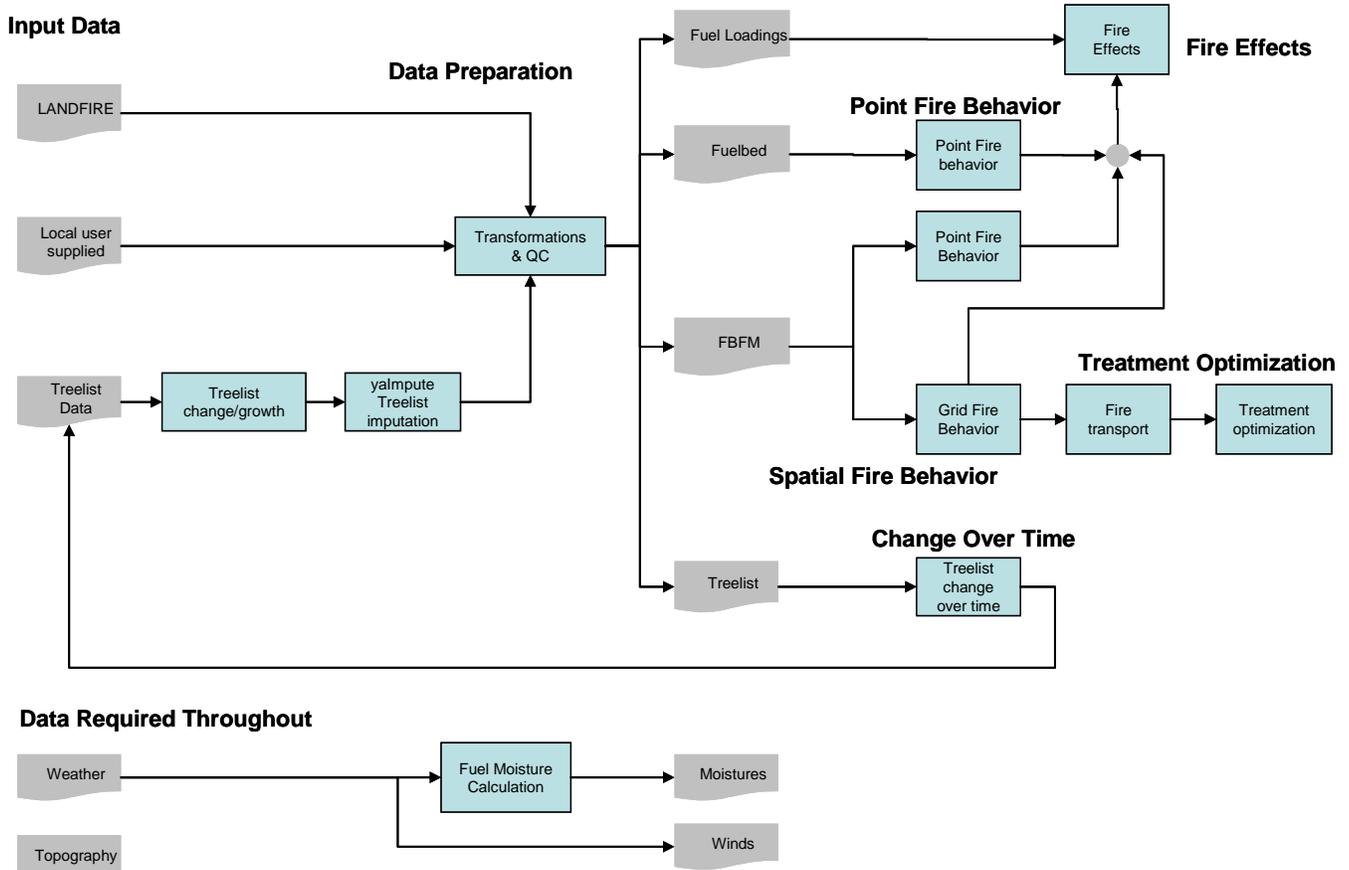


Figure 2-1. Work flow diagram illustrating how the fuels treatment planning work flow scenarios fit together from a process perspective.

2.5 OVERVIEW OF THE IFT-DSS ARCHITECTURE

This section is a strategic-level overview of the IFT-DSS software architecture. A complete description of the architecture is provided in the IFT-DSS Software Architecture Design Document⁶.

The IFT-DSS is a software framework that integrates disparate vegetation data, vegetation simulators, fire behavior and effects models, and risk analysis tools (some of which are web-based) into a common GUI. It supports the reuse of IFT-DSS applications and services over the Internet, and it is a flexible, modern, web-friendly system.

2.5.1 Architectural Components

The core of the IFT-DSS software architecture consists of five elements: (1) a multi-layered GUI, (2) a control database, (3) an executive application, (4) data and software

⁶ http://frames.nbj.gov/documents/jfsp/sts_study/ift_dss_task2_tech_architecture_draft_20090212.pdf

services, and (5) a scientific database. A description of the general function(s) of these five key architecture components follows:

- The multi-layered graphical user interface (GUI)—Provides the user with an access point into the IFT-DSS. It controls the user experience and is the single portal for all system inputs and outputs.
- The control database—Stores all information, including data and user information that is input to the IFT-DSS via the GUI. The control database manages the user experience and the decision support process.
- The executive application—Works with the control database to invoke specific functions and services. It functions as the internal system process controller and manages the flow of data and information within the system.
- The data and software services—Provides the fire and fuels treatment domain data, software models, and tools that support the decision making process.
- The scientific database—Stores all project-related input, intermediate, and output data. It is called the scientific database because it contains both spatial data (map layers) and non-spatial tabular scientific data.

Figure 2-2 illustrates the five key components of the architecture and their relative architectural arrangement. Each component contains interconnected subcomponents, or layers, that perform specific functions within the system. These components and their behavior are described in Sections 3 and 4.

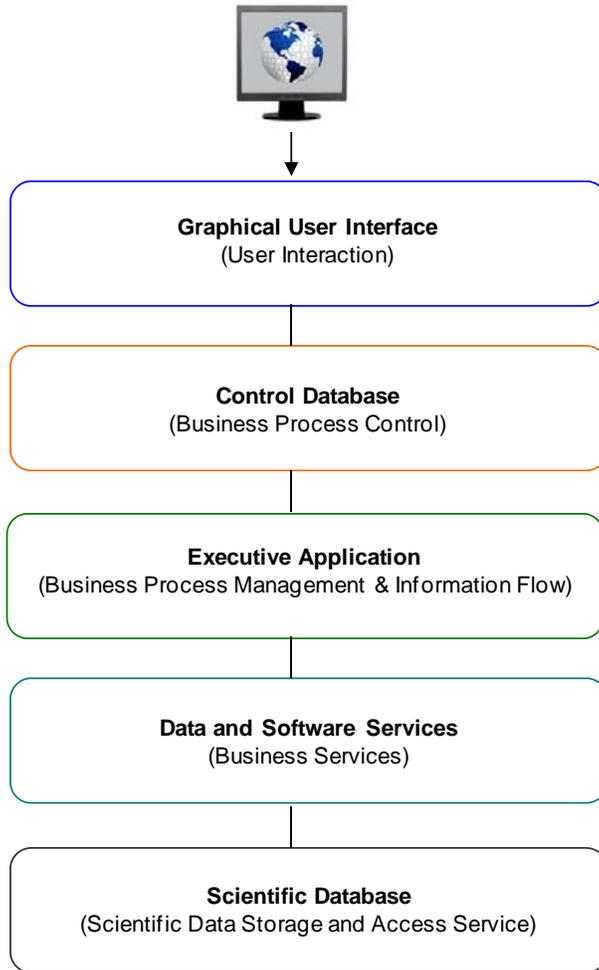


Figure 2-2. Illustration of the five key architecture components of the IFT-DSS.

2.5.2 Key Architectural Features and Functions

The IFT-DSS architecture is designed to support the following functions and features:

- organize the decision support process, analysis steps, and software tools commonly used for fuels treatment planning;
- provide data choices (i.e., standard treelist data, standard gridded data, and/or locally generated data);
- enable visualization of spatial and tabular data, editing of data, and user interaction at each processing step;
- streamline data preparation and processing by offering a mechanism to acquire, create, and transform input data (e.g., the ability to combine vector and raster data formats, perform vector-to-raster transformations, and vice versa);
- provide a quality control, documentation, and audit-trail mechanism to meet regulatory reporting requirements;

- provide guidance on user options (e.g., submodel choices) based on geographic scale and the type of analysis being performed;
- enable the stopping and starting of analyses at any processing point;
- facilitate analytical collaboration through a central system library that allows fuels treatment analysts and scientists to publish and share methods and algorithms with other system users;
- provide a mechanism to perform sensitivity analyses;
- recognize user errors and explain alternate actions; and
- facilitate and encourage scientific collaboration through an authorship and publishing mechanism that is able to incorporate new models and tools as they become available.

3. THE IFT-DSS USER EXPERIENCE

An important aspect of the IFT-DSS architecture is system behavior and user experience. This section describes the behavior of the IFT-DSS from a user perspective. Screen shots are included to show the organization and content of the GUI. For a more detailed walkthrough of the IFT-DSS GUI and how to use the system please refer to the Getting Started Guide available online within the system (http://iftcss.sonomatech.com/Getting_Started_v0.3.0.pdf).

The IFT-DSS GUI is web-based and functions within a standard web browser (e.g., Internet Explorer, Firefox). A user login screen on the IFT-DSS website is the main point of entry into the system. The first time that users log into the IFT-DSS, they are prompted to create a user profile that captures each user's email address, password, job title, organization, and other relevant information. This user profile is saved and used for subsequent logins.

From a user perspective, a project analysis will involve three phases: (1) project setup and planning; (2) software model execution and iterative analysis; and (3) project finalization, documentation, and archive. The remainder of this section describes the general experience of an IFT-DSS user during each of the three phases of a project analysis. It should be noted that the GUI screen shots are in working draft form and will evolve as feedback is received and as the system matures. The GUI screen shots contained in this section represent the latest version of the IFT-DSS v 0.3.0 as of May 2010.

3.1 LOGGING INTO THE IFT-DSS

The IFT-DSS user interface provides a portal for a user to create project analyses, execute modeling scenarios, view the results of modeling scenarios, and manage and share projects with other IFT-DSS users. The user interface is secured and accessed by using an email address and password. **Figure 3-1** shows the IFT-DSS login screen.

Once a user logs into the system, he or she can create an account and a user profile. The user profile serves as the user's central location from which he or she can create a new project analysis, revisit past analyses, manage analysis projects, and view or edit the user profile. **Figure 3-2** shows the IFT-DSS POC user home page. **Figure 3-3** shows the "My Profile" screen where users manage their profile information.

User profile information is stored in a table in the Project and Planning Database (discussed in Section 4.1.8), and is used by the system for user authentication. All project analyses created by a user are associated with a user record in the Scenario Database. The user has the ability to log out of the system at any point during an analysis session without losing information from that session. When a user logs out of the system, he or she is re-directed to the login screen.

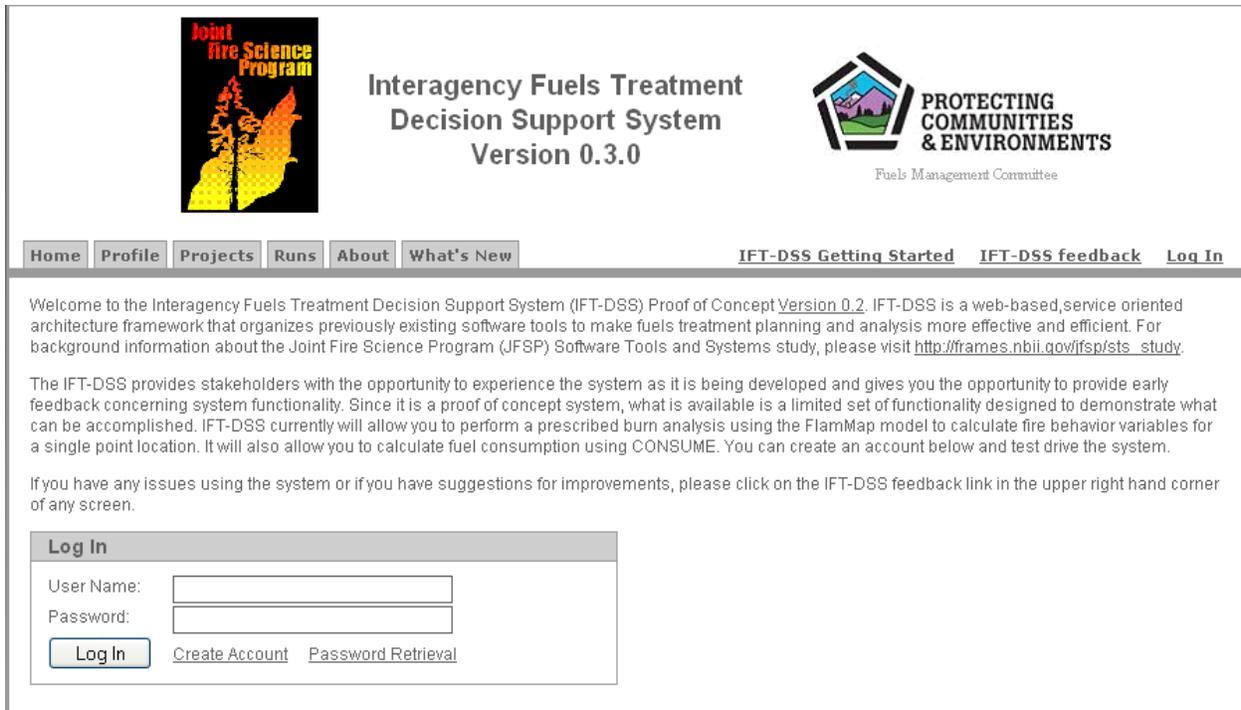


Figure 3-1. The IFT-DSS user login screen.



Figure 3-2. The IFT-DSS POC user home page.

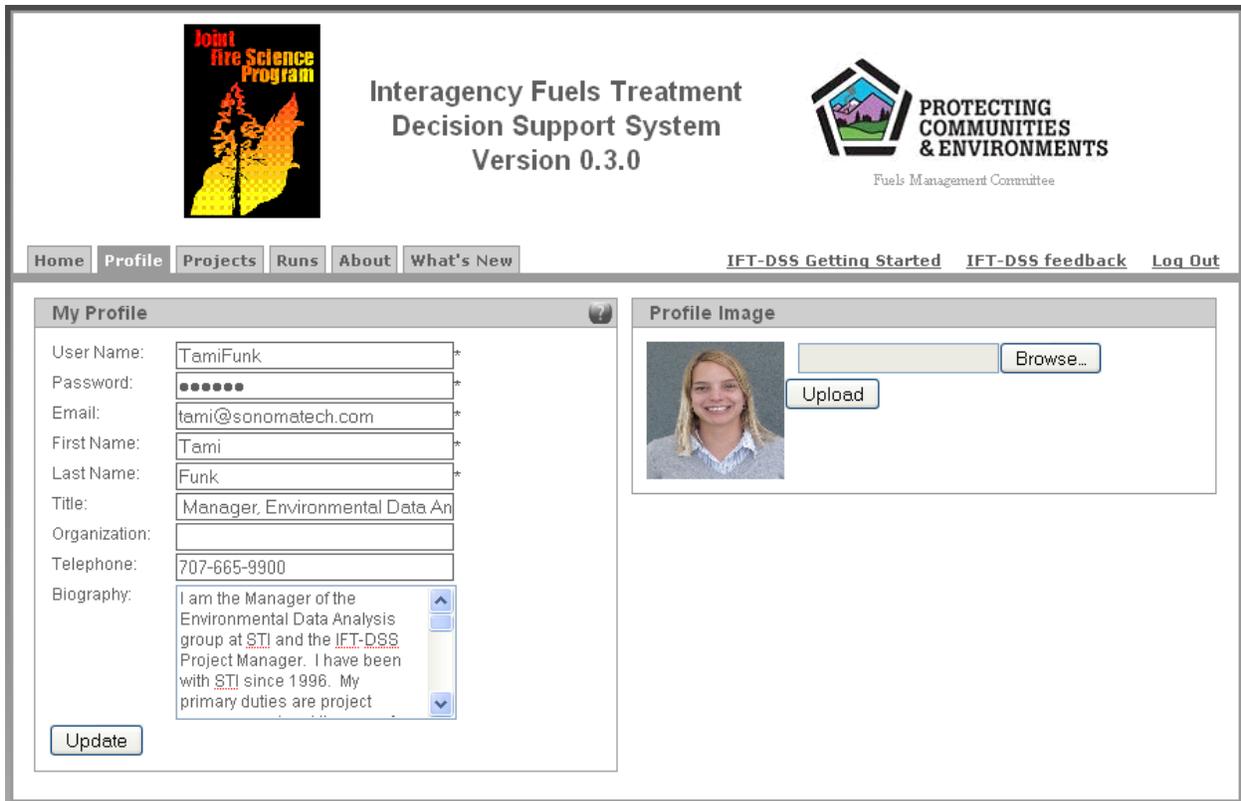


Figure 3-3. The “My Profile” screen where users manage their profile information.

3.2 PHASE I – PROJECT SETUP AND PLANNING

After a user logs into the system and reaches his or her home page, he or she is provided a list of options. From this list, the user can create a new analysis project, load an existing analysis project or modeling scenario, manage projects, and edit or view his or her user profile. **Figure 3-4** shows the user options available on a user’s home page.



Figure 3-4. The options available on a user’s home page.

Upon selecting the “Manage Projects” link, the user will be presented with a screen listing his or her projects. This is where project analyses are established and managed. **Figure 3-5** shows the project setup screen.

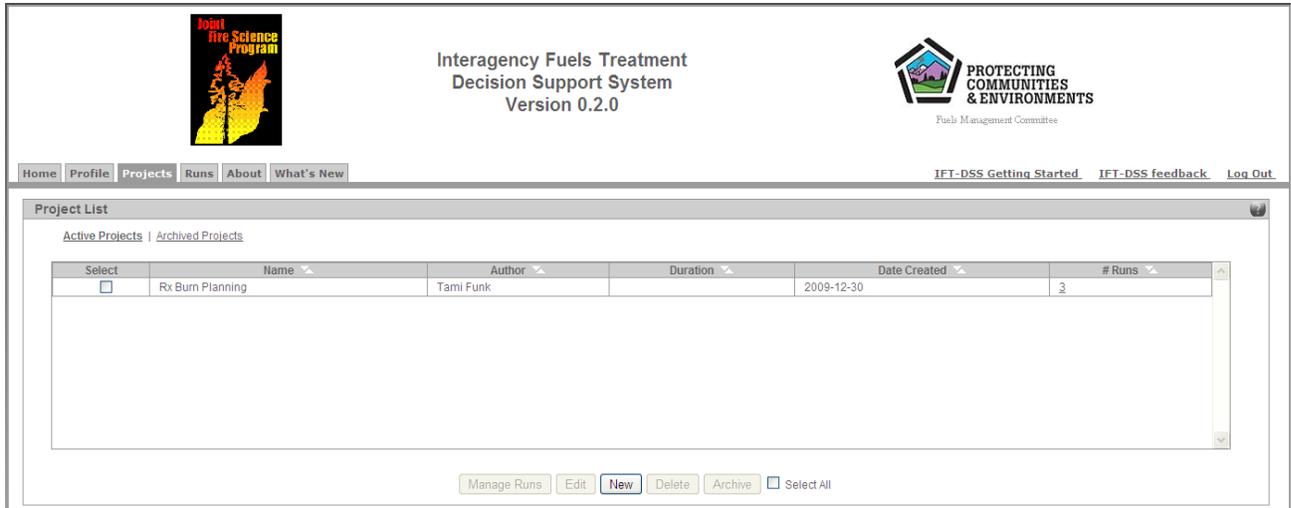


Figure 3-5. The IFT-DSS POC “Manage Projects” project list and project setup screen.

To create a new project, the New button at the bottom of the screen is selected. A user creating a new project analysis enters information about the project in the screen shown in **Figure 3-6**.

Figure 3-6. The Project Details screen in the IFT-DSS.

The user enters a name and description for the project analysis that will identify the project throughout the system. Next, the new project appears in the Project List and the user selects the Manage Runs button at the bottom of the Project List screen. A new screen appears prompting the user to set up a Run within the project. A Run is an analysis with a specific objective. **Figure 3-7** shows the Run Details screen of the IFT-DSS. The user then specifies a name for the Run, the analysis objective, and the Action Graph to be used. The Action Graph specifies the scientific models to be used for the analysis and assembles them into a process flow for the user.



Figure 3-7. The Run Details screen of the IFT-DSS.

3.3 PHASE II – SOFTWARE MODEL EXECUTION AND ITERATIVE ANALYSIS

Once a user has set up an analysis, the navigation elements of the website will change on the basis of the selected modeling scenario, and the specific pathway for the selected scenario will be displayed as a tabbed navigation bar. A dynamically generated Action Graph based on the selected pathway will also appear. The Action Graph describes the data inputs, outputs, and processes involved in running the selected modeling process. **Figure 3-8** shows the dynamically generated tabbed navigation bar and Action Graph for a sample prescribed burn planning modeling scenario using the FlamMap application to predict surface fire behavior for a point location.

The modeling scenario graph will be interactive, allowing the user to click on a step in the graph and navigate to a data input screen associated with that step. For example, if a user clicks on the “FlamMap” oval in Figure 3-8, a screen prompting the user for the information needed to run the FlamMap model will appear. The scenario graph will also provide feedback to the user by graphically depicting how far the user has progressed through the steps of the modeling scenario run at any given time. The purpose of the modeling scenario graph is threefold: (1) to visually describe the inputs, outputs, and processes associated with a selected modeling scenario pathway, (2) to allow the user to access the input screens for the selected modeling scenario pathway, and (3) to report the status of a modeling scenario run once the run has been executed.

The pathway that defines the modeling scenario will be stored in the Control Database and dynamically retrieved by a program called “pageCreator” at run time. The pageCreator’s role will be to interact with the database and retrieve pages and elements for each page, based on the selected modeling scenario. The pageCreator will also dynamically generate a scenario graph (Figure 3-8) for the modeling scenario selected.

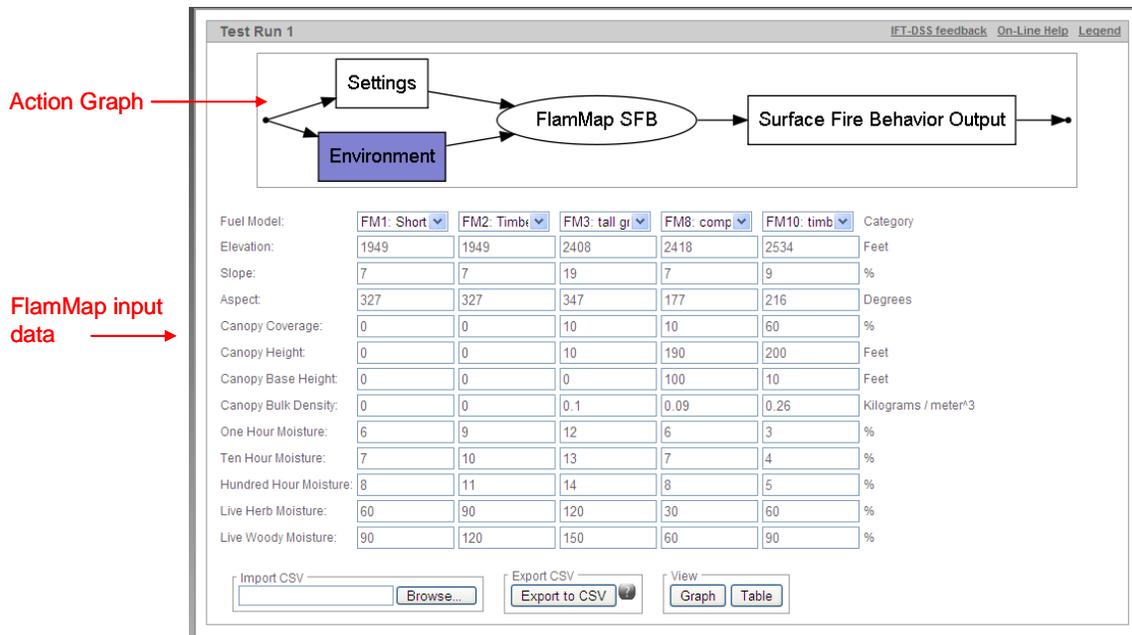


Figure 3-8. The dynamically generated tabbed navigation bar and Action Graph for a sample modeling scenario.

At this point, the user can navigate through the modeling scenario pathway by clicking through the Action Graph (Figure 3-8). This style of navigation allows the user to iterate the steps of the modeling scenario. At each step, the user can provide input data or accept default values in the text field inputs. At each step in the modeling scenario pathway, the user can execute an individual model by clicking on the oval in the Action Graph. The outputs from the model are then displayed (**Figures 3-9 and 3-10**) and will become the inputs to the next model or process in the modeling scenario pathway.

As a project analysis is performed, the IFT-DSS tracks the status of model run scenario processing time. Future versions of the IFT-DSS will allow the user to view status updates on the user home page or receive email status notifications for processes that require a long period of time to complete. The email notifications will include a link connecting the user to the completed process step, allowing the user to continue stepping through the scenario from where he or she left off.

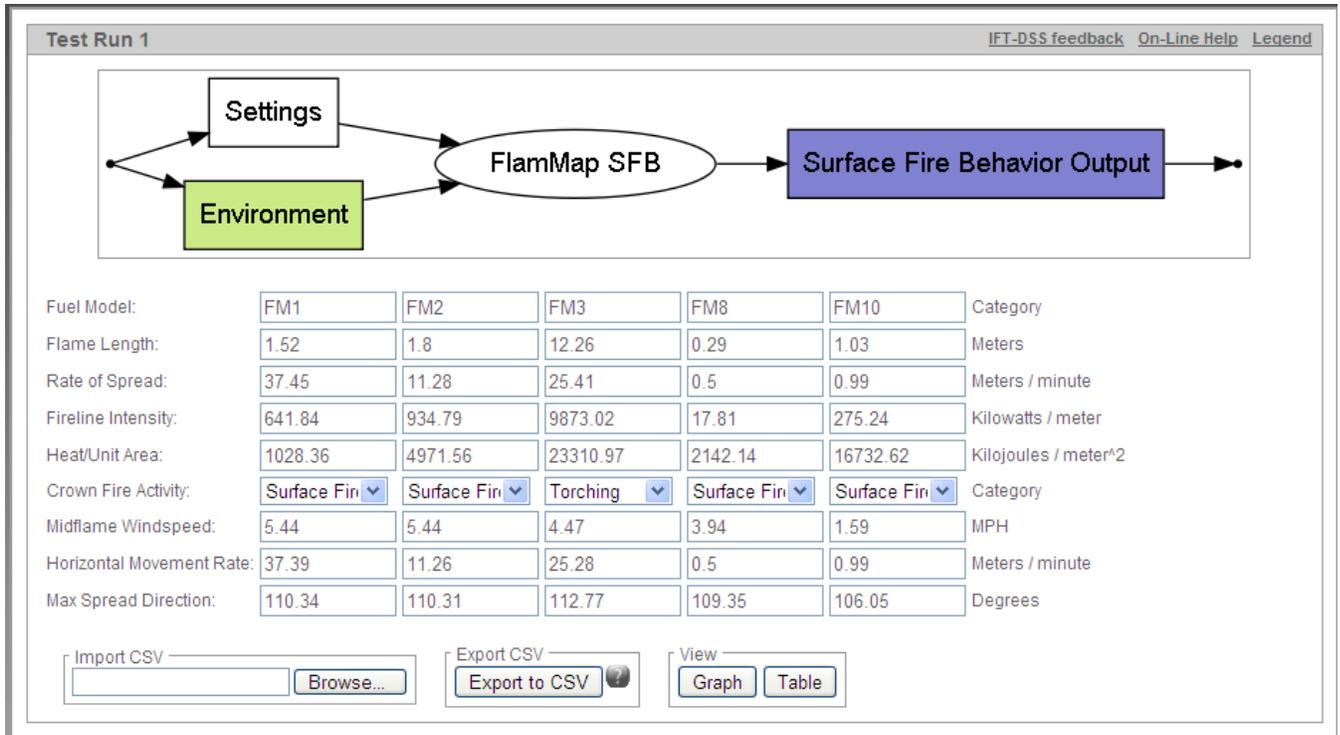


Figure 3-9. Sample outputs from FlamMap in tabular form.

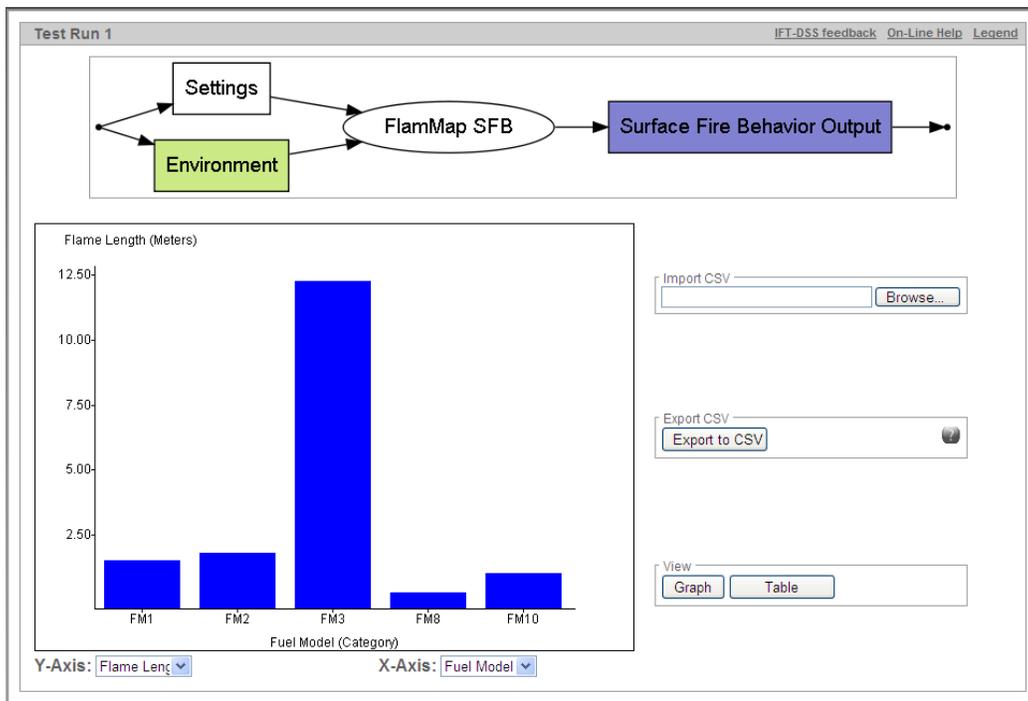


Figure 3-10. Sample outputs from FlamMap in graphical form.

The IFT-DSS allows the visualization and manipulation of spatial data. **Figure 3-11** shows a screen shot of the map viewer within the IFT-DSS GUI (left) displaying a satellite map base layer. After running the FlamMap model for the area of interest, the map window allows visualization and manipulation of the FlamMap output data layers (right). The map viewer is dynamically linked to the action graph at the top of Figure 3-11 allowing dynamic navigation among action graph data entry screens. For example, when the Wind box on the action graph is selected, the data input screen appears for entering wind data for use in FlamMap. The IFT-DSS also allows map layers to be exported and viewed in Google Earth.

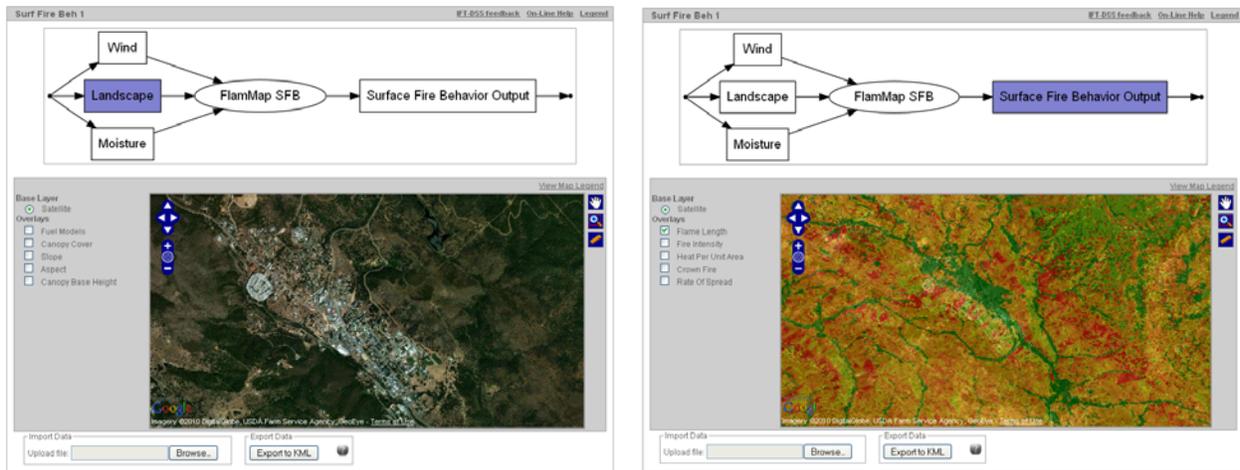


Figure 3-11. Screen shot of the map viewer within the IFT-DSS GUI (left) and the map viewer within the IFT-DSS displaying fire behavior output data from FlamMap (right).

3.4 PHASE III – PROJECT FINALIZATION, DOCUMENTATION, AND ARCHIVE

After reaching the end of an analysis, the user can save the scenario as a Run within the project analysis. The project is then accessible from the Manage Projects screen shown in Figure 3-5. The Manage Projects screen allows the user to view projects, view runs within a project, view project results, and assign sharing privileges to other users. For example, in future versions of IFT-DSS, a user could designate another user as a contributor who would have fewer (or read-only) privileges than the project owner.⁷ The user can also archive a project from this screen.

From the Manage Projects screen (Figure 3-5), the user can view and edit a project or modeling scenario run, share a project with other users and specify their user privileges, archive a project, and generate reports related to a specific project. The archived projects will continue to be available to the user but will no longer be considered active. See Section 4.1.9 for details regarding the report generation for a project.

⁷ The ability to share projects will be active in future versions of the IFT-DSS.

4. TECHNICAL SOFTWARE DESIGN

This section presents the IFT-DSS POC software design in a manner that is generic and almost completely independent of programming language, hardware, and operating system implementation decisions; that is, the design can be realized in a variety of ways using any one of many hardware and operating system configurations. It is important to realize that this design was constructed in a way that system components can be developed in different languages and use different operating foundations while still conforming to this design. Approaching the design in this manner allows the software architect to maintain a maximum degree of modularity when considering how the system will be constructed. Section 5 contains the language, hardware, and operating system details for version 0.3.0 of the IFT-DSS.

The presentation of the IFT-DSS software design in this section is separated into three parts: (1) a description of the system components, (2) a description of the connections among these components, and (3) a description of the behavior of the system as a whole. **Figure 4-1** is a diagram of the overall structure and components of the system. The structural components are labeled with letters, and the software interfaces are labeled with numbers. Each of the nine components, A through I, is described in Sections 4.1.1 through 4.1.9; each of the seven software interfaces, labeled 1 through 7, are described in Sections 4.2.1 through 4.2.7. Section 4.3 revisits the components and software interfaces and describes how they interact.

Components

- A. Models** – The scientific and computational components of the system; all other components support the model operations
- B. Model Adaptors** – Enable integration of different models into the system
- C. Scientific Database** – Stores the actual data inputted to and outputted from models
- D. Data Interface** – Relays data between the scientific database and the models
- E. Executive** – Directs the execution of the models
- F. Control Database** – Stores representations of how different “runs” of models are orchestrated
- G. Navigator** – Visualizes spatial data, allows users to edit data values for model inputs, and allows users to execute runs through scenarios or parts of scenarios, all from a web browser
- H. Project and Planning Database** – Stores administrative data about fuels treatment projects
- I. Planner Session Engine** – Enables users to manage fuels treatment projects from a web browser

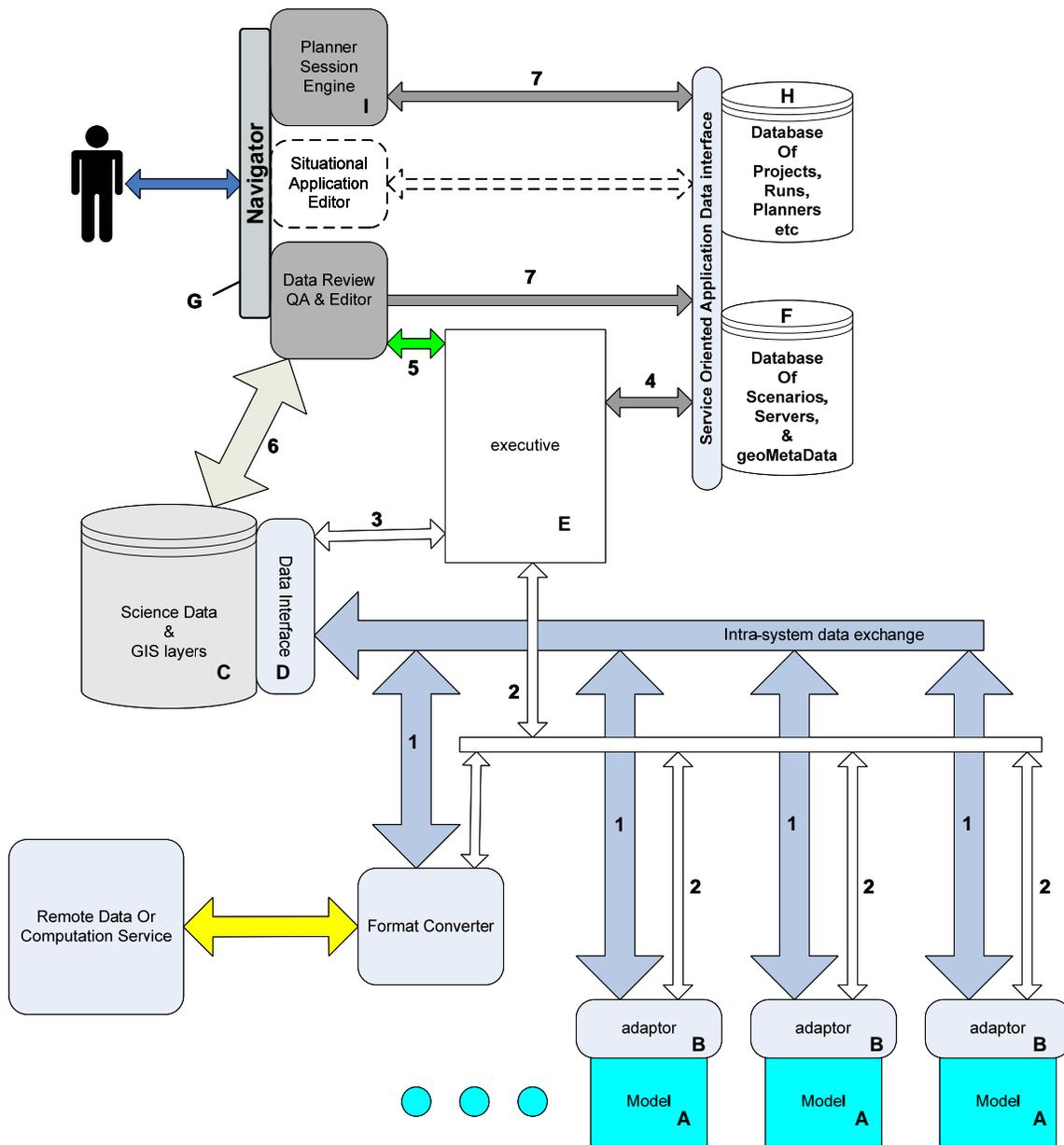


Figure 4-1. Overall structure of the IFT-DSS depicting the system’s major components (with lettered indicators) and software interfaces (numbered). Features that will be implemented in future versions of the IFT-DSS are represented with dashed borders.

Interfaces

1. **Model Data Exchange** – Connections between machines running models to relay model inputs and outputs
2. **Model Control** – Connections between the Executive and Model Adaptors to prepare the model data exchange connections

3. **Sequencer Control and Monitoring** – Interaction between the Data Interface and Executive to manage scenario execution
4. **Executive Scenario Polling and Feedback** – Executive queries of the Control Database for required data, and feedback about model and machine execution performance
5. **Executive Launching and Progress Reporting** – Web application signaling of the Executive to compile and run a scenario segment and the Executive reporting progress back to the web application
6. **Map and Data Presentation Interface** – Communication between the Scientific Database and the web application components
7. **Web Page Generator and Database Interface** – Web application queries of the Control Database to produce the web pages displayed to the user for model inputs and outputs

All of the system components have been realized in version 0.3.0 of the IFT-DSS to some extent with two exceptions: (1) the collaboration interfaces, as indicated with dashed borders in Figure 4-1, and (2) some of the more complex scenario execution optimizations described in this section.

Experience gained from using the initial versions of the IFT-DSS will help determine priorities for optimizing existing features and adding new ones. For example, it may prove to be more valuable to add new models and scenarios than to maximize resource utilization through parallelization of model execution. Some of the optimizations will become possible when system deployment configuration and run execution times are known. The features involved in these optimizations reside primarily within the executive, Data Interface, and model adaptors.

The collaboration interfaces include support for analytic collaboration among planners sharing knowledge and effort and for scientific collaboration in the authoring of models and model scenarios. The analytic collaboration interface is scheduled for implementation during year 2 of system development (2010). At that time, support for scientific collaboration will also be added by providing administrative access to the Control Database loader tool through an administrative GUI so that new scientific models and scenarios can be added to the system.

The analytical collaboration interface is the mechanism by which planners work together on projects. This includes control of shared access to project data and runs. A project owner will be able to solicit input from selected peers. Planners will also be able to discuss the preliminary results through a project-specific message relay or discussion forum. This will augment the planning project results by encouraging planners to work together within the system. It will also document quality assurance by recording peer feedback with the other project data.

The scientific collaboration interface provides a mechanism for advanced users, data providers, and software developers to add new capabilities to the system. This will include a scenario editor and a software development environment. The scenario editor is the mechanism by which advanced users can construct custom analysis pathways. Users of the existing IFT-DSS GUI are able to select an analysis objective and choose from the predefined processing

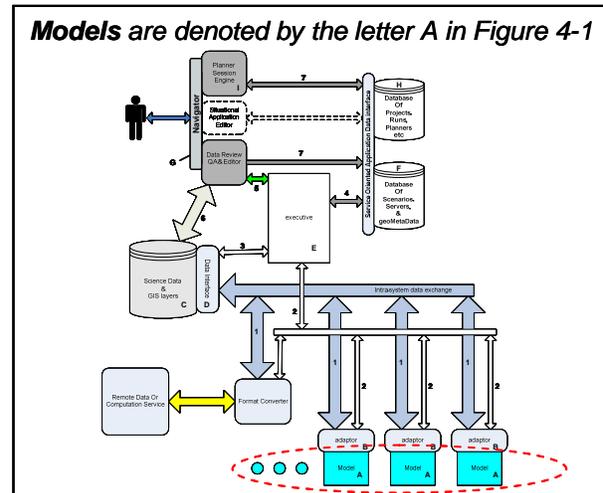
scenarios supporting that objective. The scenario editor will extend this capability by allowing users to select data and model sequences to create custom analysis pathways.

4.1 COMPONENTS

The components described in Sections 4.1.1 through 4.1.9 are software programs and subsystems.

4.1.1 Models

Models constitute the heart of the system, as they are the scientific and computational components of the system. All of the other system components were designed to support the operation of the models and the examination of the model outputs.



The first models that have been integrated into the IFT-DSS are programs (or parts of programs) that already exist and are in use in the fire and fuels community. These models include FlamMap and Consume. These models have been integrated into the IFT-DSS by the methods described in Section 4.1.2.

In the future, as protocols and guidelines for integrating models into the IFT-DSS become available to model developers, it will be much easier for a model to be integrated into the IFT-DSS. Future models will be better aligned with IFT-DSS's execution paradigm, resulting in several advantages: they will be much smaller than any standalone modeling program, they will be simpler to develop and maintain, and they will be able to take full advantage of the system's parallel processing architecture.

Developers will be able to integrate models into the IFT-DSS by using the "Model Subclass" method described in Section 4.1.2. In this simple and efficient approach to incorporating models into the system, future models may consist of small sections of programming code that define and execute specific, self-contained functions. For example, many existing standalone software applications contain programmed functions that are linked together and tightly coupled to a user interface. In the future, it is more desirable for model developers to program functions as discrete pieces of programming code that are unlinked to other functions and are decoupled from a user interface. Ideally, they will consist of a relatively small number of lines of program code embodying a single behavior or very few modeling equations. This approach will allow new functions and features to be easily integrated into the IFT-DSS without the need to redesign the IFT-DSS software framework or user interface.

The IFT-DSS contains a system administration tool, the Control Database Loader (CDL) that facilitates the implementation of existing and new models into the framework. The CDL tool provides a user interface in which a series of screens prompts the system administrator to enter information about a model, including its inputs, units, and outputs. The CDL tool also provides a model builder screen where actual programming code can be created or entered and

registered within the IFT-DSS system. **Figure 4-2** shows a screen shot of the CDL administration tool.

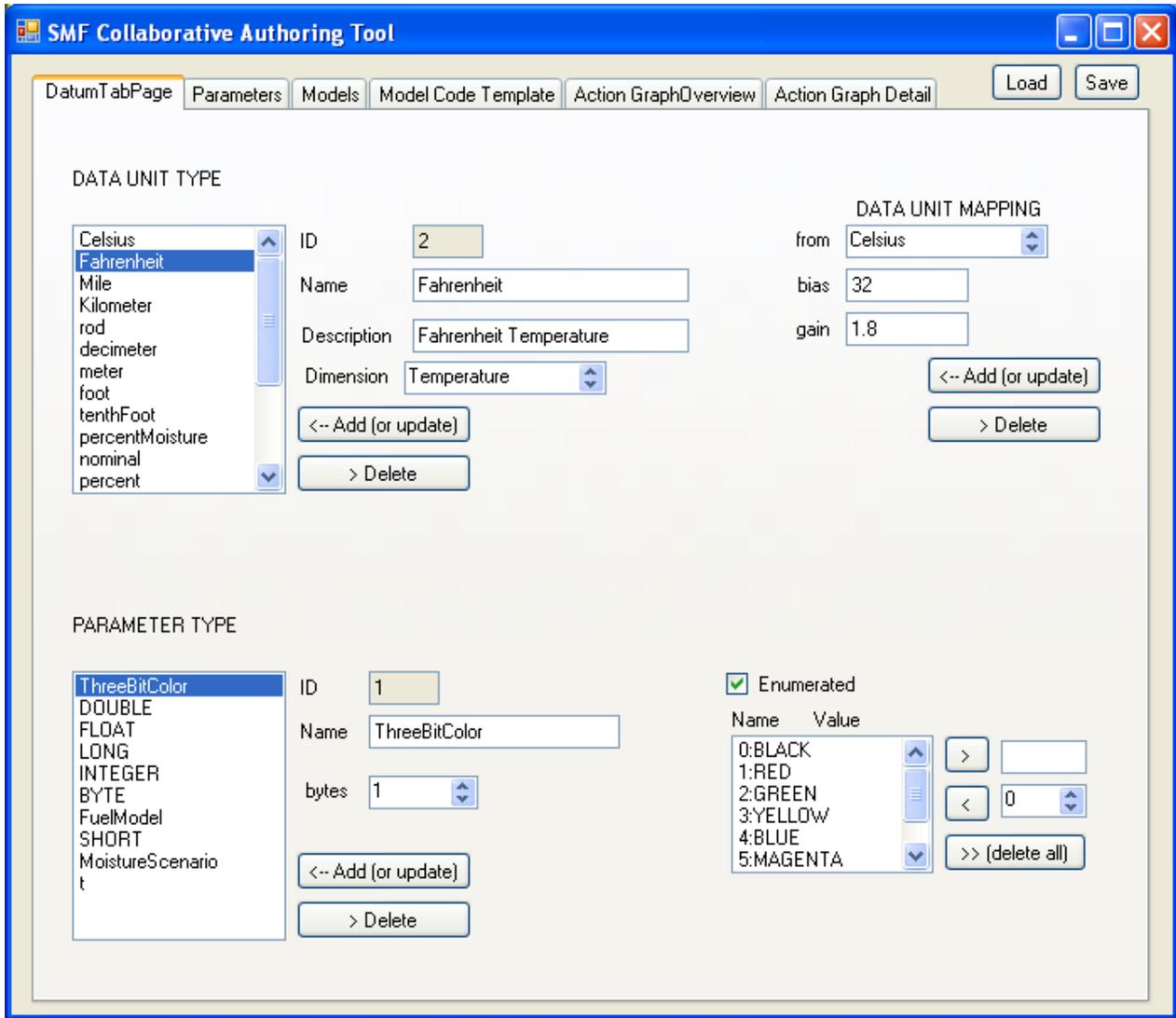


Figure 4-2. Screen shot of the CDL tool.

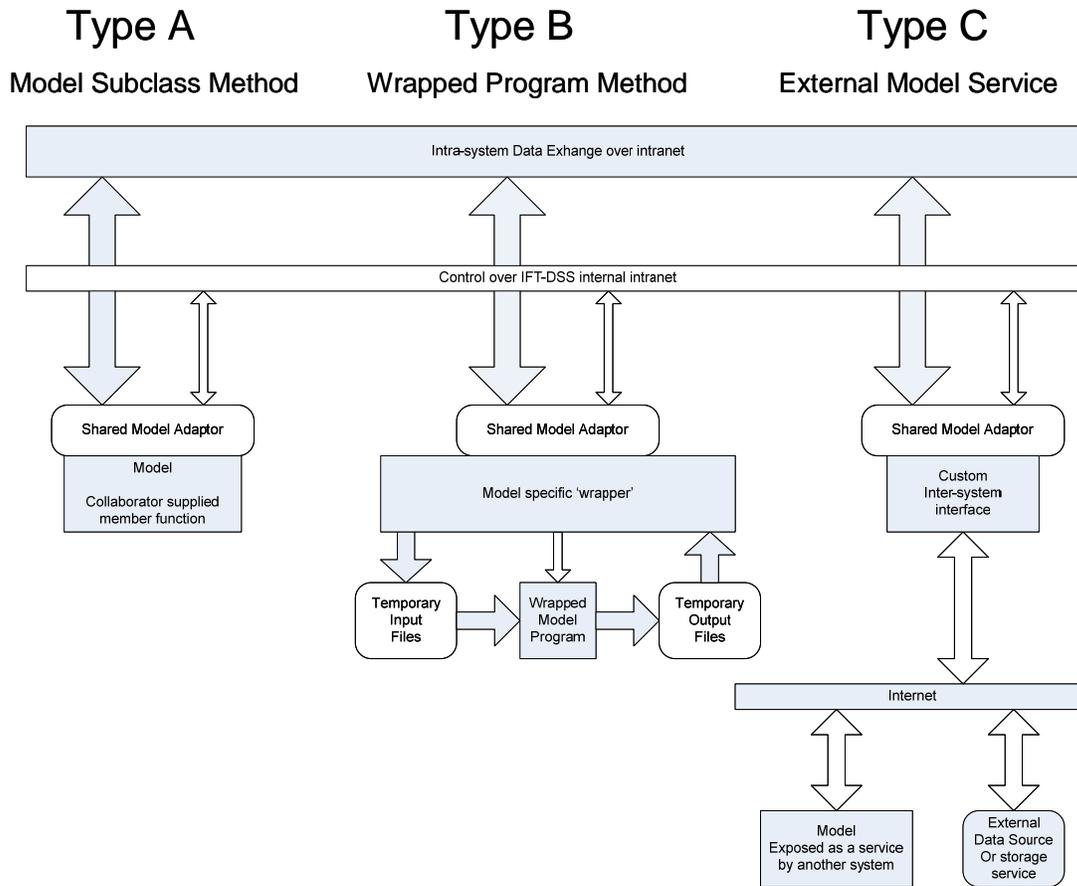


Figure 4-3. Illustration of the three model hosting methods by which models can be integrated into the IFT-DSS and function as services.

Model Integration Methods

As shown in Figure 4-3, there are three types of model adaptors, Types A, B, and C, that define three ways by which a model can be added to the IFT-DSS.

1. Type A – Model Subclass Method – The adaptor type on the left depicts the most efficient method. It consists of a standard application programming interface (API) that includes an application server, a hosting program, communication handlers, and all of the data handling code required to operate the modeling code. The model is added to this code base by extending (creating a subclass of) the model parent class. The other “wrapping” methods employ these same components (except the model parent class).
2. Type B – Wrapped Program Method – The middle adaptor type depicts a more expedient method for deploying existing model programs by providing a wrapper program that communicates with an existing model program and translates the model inputs and outputs to the format standards used by the IFT-DSS.

3. Type C – External Model Service Method – The adaptor type depicted on the right side of Figure 4-3 allows the IFT-DSS system to incorporate functionality from other systems such as BlueSky and WFDSS by utilizing web services.

These methods all share a number of important features. Near the top of each module stack depicted in Figure 4-3 are three shared features represented by the rectangles labeled “Shared Model Adaptor.” Below each of these are various forms of “Model” components.

Model Control Communications

The first component required for service-oriented model hosting is an application server. The IFT-DSS employs an off-the-shelf (OTS) solution for this requirement. The purpose of this component is to receive messages from an external control, interpret them as requests for model execution, and act on them by deploying the model host program. “Models” are computational objects, at the heart of each model integration method, that run on the distributed compute servers of the IFT-DSS. The OTS service provider causes the models to run, passes them arguments and commands, and relays status information. Model hosting programs receive the control messages from the Executive (E in Figure 4-1). The protocol of the commands is described in Section 4.2.2. The application server communicates status information from the model components back to the Executive through the same interface.

Model Data Communications

In addition to command and control data, the models communicate model data on a peer-to-peer basis. These data pass among models running on distributed computational servers, the Scientific Database, and other interface subsystems. See **Figure 4-4** below for a Unified Modeling Language (UML) diagram corresponding to the following description of model communication infrastructure.

Each model program has zero or more input and output streams. Each input stream demultiplexes data from one or more connections to an output stream of another model. Likewise, each output stream multiplexes or broadcasts through one or more connections to the input streams of other models. Information about the number of streams, their data type, their fan-out or fan-in, and the specific addresses of each channel socket are received from the Executive.

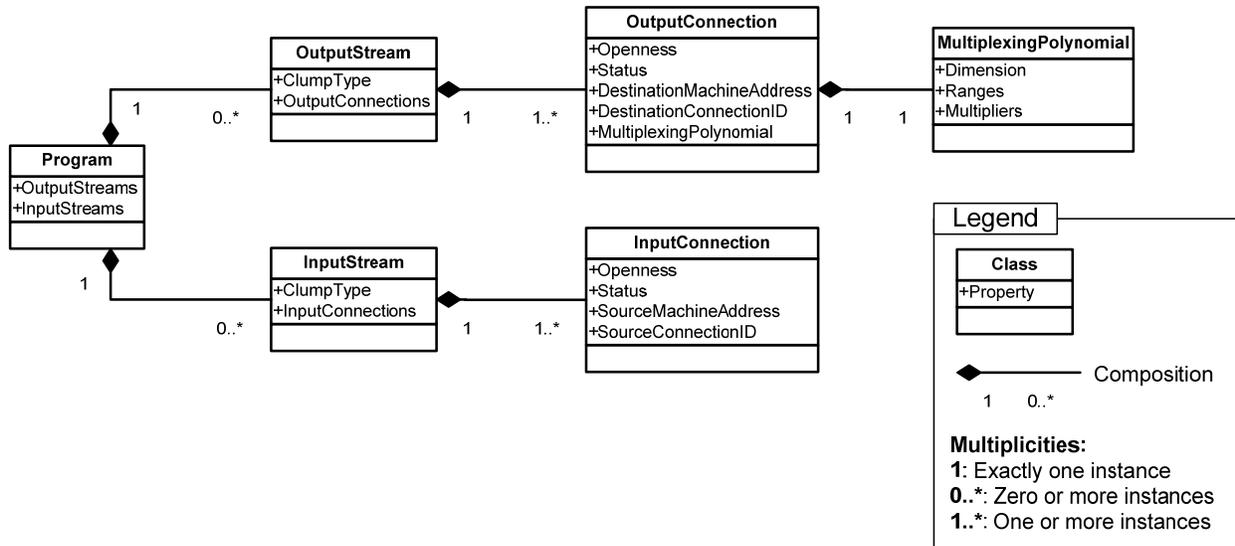


Figure 4-4. UML diagram of the IFT-DSS model communication infrastructure.

Each output connection has a “multiplexing polynomial,” which allows parts of a model to be run on different machines. It is also possible to do the same work on multiple machines or to use the output from one step as the input to several later steps. Likewise, data can simultaneously be sent to the next modeling step and to the Scientific Database for archiving. This data distribution capability is achieved by the member functions of the input and output stream classes, which use the multiplexing polynomial to sequence data from an output stream to a set of input streams. These classes have members common to stream-like objects including open(), close(), read(), and write(). However, the actual behaviors of these class members are different from those of their familiar analogues. To illustrate these differences, **Table 4-1** shows the pseudocode for the output stream members.

Table 4-1. Pseudocode for key member functions to the output stream class.

<pre> Open() For each connection Save multiplexing polynomial coefficients and ranges Fork and in the new thread Note the time Open the connection End thread End for End of open() </pre>
<pre> Close() For each open connection Terminate the connection (i.e. session) End for End of close() </pre>
<pre> writeObject() format the object into a packet for each destination connection check object index against multiplex polynomial if index is in a write window fork wait until connection is ready send the packet through the connection note the time end thread end if end for end of writeObject() </pre>

The `open()` and `writeObject()` member functions use separate threads for each connection because each of these can require some time to synchronize with the other end of the connection. In the pseudocode this is represented with “fork” and “end thread” clauses. None of the actual programming languages used to implement these behaviors has an analogous construct.

Input streams have `open()` and `close()` behaviors similar to the corresponding behaviors of output streams. However, there is no need for a channel select polynomial. All of the data that arrive at an input channel are processed and the input streams have a `readObject()` behavior instead of a `writeObject()` behavior. `ReadObject()` reads one object from any of the one or more open connections. **Table 4-2** shows the pseudocode for the input stream member.

Table 4-2. Pseudocode of the readObject member function of the input stream class.

```
readObject()
  for each input socket
    check for pending data
    note how long it's been waiting
  end for
  read the packet that has been waiting the longest
  extract the data object from the packet
  if an object with this packets index has already been received
    discard packet
    start over from the top
  else
    return the received object
  end if
end of readObject()
```

The stream classes handle the peer-to-peer flow of science data. In addition, the model host program must communicate control and status information with the Executive and cause the model to be performed for each set of data inputs. This part of this program is the same for all models.

The processing here is independent of output multiplexing, but it must take into account an analogous property of inputs. Each input stream must have a known “front aperture” description. The front aperture is a specification of the neighborhood of locations at which parameters must be supplied to the model in order for the model to calculate its result. The front aperture of many (aspatial) models is a delta function. In other cases, it may span the entire domain of the input. Both of these conditions can be indicated with flags. In other cases, ranges of values will have to be specified for each input coordinate.

Processing entails running a model many times. The sequencing of model execution is determined by the parameters of the program. One form of sequencing is to execute the model once for each record in a specified (“indexing”) input stream. Indexing streams may have to be constrained to have only delta function entrance pupils. In some cases, there may be no suitable input stream. When this occurs, a special form of input stream is required. This pseudo-input stream would have no connections but would respond to readObject() calls with indexed null data rows.

The model hosting program is responsible for configuring peer-to-peer connections with other models, managing all communications, and executing a model as needed. A very brief pseudocode of this component is presented in **Table 4-3**. The data handling for each model is divided into phases. Only a single phase is required, but generally, three I/O phases correspond to prerequisite, concurrent, and retrospective communications. More phases are possible, and there is no benefit in constraining the range of supported I/O phases. Zero or one model functions are executed during each model phase.

Table 4-3. Pseudocode of the model hosting program.

```

Model Hosting Program:
  Parse all control parameters from dispatcher
  For each phase of the model
    Open each output steam
    For each input stream
      Fork and in the new thread
        Open
        WHILE (still objects to read)
          Read in an object
          Unpack
          For each effected ingest queue object (including new ones)
            Distribute data into ingest queue object
            If ingest queue object is 'complete' (i.e. aperture full)
              Send message parent thread
            End if
          End for effected queue entries
        END WHILE
        Close
      End of thread
    End for all input stream
    WHILE (more 'aperture full" messages expected)
      Wait for 'aperture full' message from daughter thread
      Do the model operation(s)
      For each output stream
        Write a result object
      End for
    END WHILE
    Close everything
  End for phase
END of model hosting program

```

During each processing phase, all of the outputs are initiated, and a separate processing thread is produced to handle each input. The input threads each initiate the communications associated with a single input stream, receive and process each data object, and finally terminate the communications. The processing done by these threads is limited to unpacking received data objects and distributing their content into the appropriate areas of large, queue-like data structure where model input objects are staged. Whenever the data insertion results in the completion of a staged data object, a message is sent to the main thread. The main thread, upon receiving notification from one of its daughter threads, causes the model's routine to execute and process the staged input data. Any output objects that result are transmitted. Monitoring of staging queue object completeness continues until all of the input threads terminate. Then the main loop finishes and either moves on to the next phase or terminates.

4.1.4 Data Interface

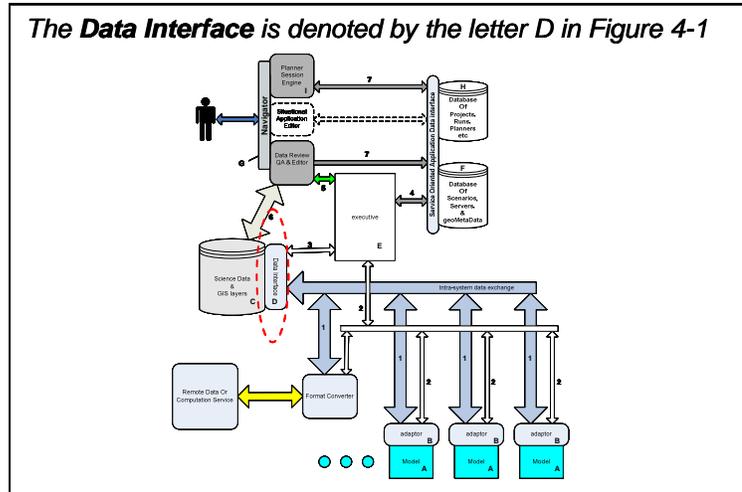
The Data Interface is an interface between the models and the geographic information system where the modeling inputs and outputs are stored. Since the sequencer determines what data are processed, it is responsible for managing key aspects of system load.

Beyond simple speed considerations, this component has the opportunity to maximize performance by limiting the waste and redundancy in the data processed by the models. When performance constraints merit, the component can exploit these additional opportunities. Although version 0.3.0 of the IFT-DSS does not have these features, the design does account for them, and their implementation will be prioritized along with expansion of the initial suite of modeling capabilities.

It is desirable to ensure that only required data are processed. When only a small section of the input data needs to be processed (as is the case when models must be re-executed following edits), the models' front aperture description will be used to ensure that only the required areas of the input data are reprocessed. This check requires cooperation between the Data Interface, which has access to relative data ages, and the Executive, which has the apertures and controls the topology of the compute service network.

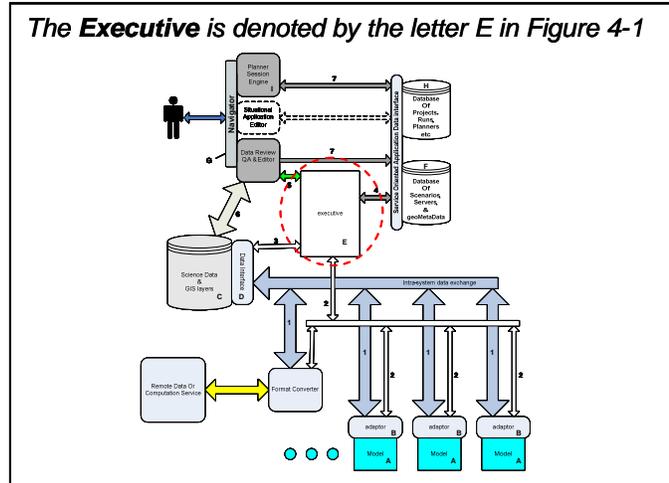
A later version of this component may further improve processing efficiency by determining which data within the edited regions are redundant and therefore do not need to be processed. This type of optimization is most easily incorporated into the model hosting programs, but the gains may be greater if implemented in the Data Interface.

Prior to process initiation, a series of messages will pass between the Executive and the Data Interface. These will include communication about masks that depict the areas affected by data updates. By working with the Executive (described in the next section) and the Scientific Database to process these data, this program will generate input and output masks for the affected areas of the scenario fragment. The Executive (after distributing the scenario fragment across the pool of computational resources) will send information to the Data Interface that will be used to make the necessary input and output connections and plan the distribution of data to the appropriate outputs. Finally, the Data Interface will receive information from the Executive upon initiation of the execution of a scenario fragment. This information will include the identity of the run, a list of the run fragments' saved results, and a list of required inputs and intermediate data sets. The inputs and results lists will trigger the generation of write and read sockets as described earlier. The interaction described above is summarized in Section 4.3.3 as a swim lane diagram (Figure 4-12).



4.1.5 Executive

The Executive component compiles a scenario segment and is invoked in response to an action by the user. A scenario is represented as a Directed Acyclic Graph (DAG). In the DAG, the models are the nodes and the data passed between models are the edges, or vectors. The Executive receives parameters defining which DAG vector of which scenario should be produced.



The Executive responds by querying the Control Database for additional data about the run's scenario. After analyzing these data, it causes model services to begin running with appropriate peer-to-peer data connections. Then, it directs the Scientific Database Data Interface to send the required input data. This, in turn, causes the models to execute and result data to be returned to the Scientific Database. This process is summarized in the brief pseudocode shown in **Table 4-4**.

Table 4-4. Brief pseudocode for the Executive program.

```

Executive program
  Receive scenario (section) from User Interface
  Get model, server, scope, etc data from scenario DB
  Compile a Network of connections and processes
  For each process
    Connect to server and start process
    Relay connections and other configuration data
  End for
  Monitor status and diagnostic feedback
  Notify user (by popup or email)
  Update resource capabilities in Control Database
End of program

```

The most complicated part of this program's work is the compilation of a scenario fragment into a set of program execution processes and connections among these processes. This compilation begins with several recursive, binary, spatial operations. As noted above, some of this work will be done by the Executive, and some of it must be delegated to the Data Interface or the Scientific Database.

Since recursion is difficult in the relational database management system (RDBMS), the scenario's entire set of models and connection will be retrieved from the database. Then the Executive will recursively search the scenario's DAG to generate a (possibly redundant) tree-like graph of model nodes and communication vectors. Using code on the returning side of the recursive calls, the Executive will exhaustively traverse this graph, calling on the Scientific Database or Data Interface to generate an input newer than the output mask for each input to

each model. If necessary, this step can be optimized or omitted altogether in cases where enough data are modified in the high-order branches to warrant recomputation over the entire domain of the run. Each of these masks is projected forward through the model apertures and onto the output space of the scenario fragment's final results. This combined, dilated mask identifies the part of the results layer that must be recalculated. A final recursion across the DAG in which the output mask is back-projected (with another binary dilation) from each model output to the corresponding inputs is performed. This final mask-building process will use model latency to take advantage of any temporary, intermediate result layers that may have been saved.

Next, any redundancies in the masks are removed (by "or" operations), and the area of each mask is integrated. These areas, multiplied by difficulty estimates for each model, give the system fairly accurate estimates of the size of the modeling tasks that are to be done.

If no single task is very large and the number of computers available is greater than the number of tasks, or if all of the tasks combined add up to a small amount of work, then each model is assigned to the computer most able to do it in order of model difficulty.

Conversely, if it is worthwhile to use a larger network of computers to execute some parts of the scenario fragment in parallel, the work is divided into a large number of approximately equal-sized small tasks. These tasks can be assigned to the computers best suited for them based on task difficulty and bandwidth limitations among computers.

The pseudocode in **Table 4-5** shows the compilation steps. The compiler in the initial version of the Executive is not as complex as the compiler described here. Aperture calculation with a rudimentary implementation will be adequate because all of the models to be incorporated in the initial versions of the IFT-DSS will have delta function or space-spanning apertures.

Table 4-5. Pseudocode for compiler component of the Executive.

```

MakeEffortTreeNode(target)
  For each parent of target
    MakeEffortTreeNode(parent)    // RECUR
    getMaskOf(parent younger than target)
    project parent mask through aperture onto target mask
  end for
end MakeEffortTreeNode

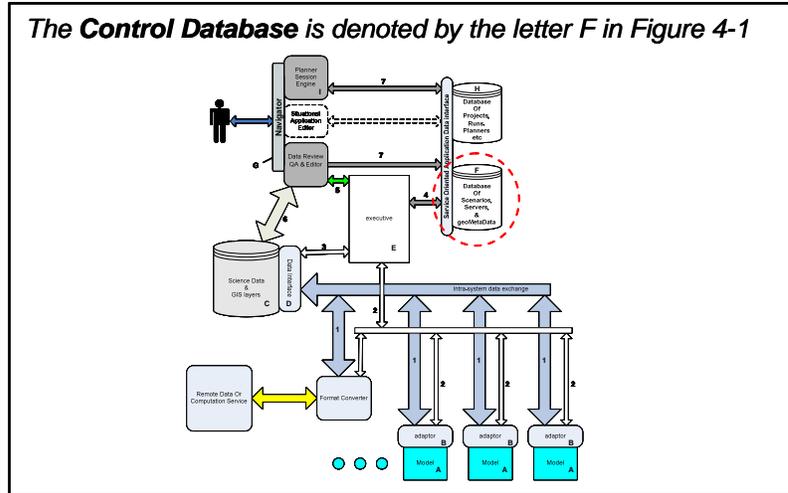
BackProjectMaskOf(Target)
  For each parent of target
    project target mask through aperture onto parent mask
    BackProjectMaskOf(Parent)    // RECUR
    Assign parent work phase
      based on phases of connection between target and Parent
  end for
end BackProjectMaskOf

COMPILER(run, target)
  Get scenario for run
  Make effort tree node(target)
  BackProjectMaskOf(target)
  For any redundant nodes
    Join mask by OR
  End for
  For each node
    Count mask size
    Estimate effort
  End for
  For each work phase
    Use approximate histogram equalization
      on the distribution of effort estimates
      to divide the work into multiple, small tasks
  end for
  Partition work space and assign data connections among tasks
  For each task, in inverse order of complexity
    Assign task to the most competent machine
    Remove machine from work phase's resource pool
  End for
END of COMPILER

```

4.1.6 Control Database

The Control Database stores relational database representations of scenarios and their accompanying DAGs. The Control Database provides data to control GUI screen formatting and model execution and is vital to the behavior of the system. The conceptual data model and its influence on system behavior will be discussed in this section.



The data model's purpose is to represent a DAG for each scenario (see the conceptual Entity Relationship Diagram (ERD) in **Figure 4-5**). By representing a scenario as a DAG, the Executive can resolve how and when to execute models. The DAG will also be represented in the GUI to facilitate user navigation through a scenario.

Models are fundamental building blocks of a scenario and are represented as nodes in a DAG. A Model entity has inputs and outputs, which are linked together. These links are represented as the data model Vector entity. To ensure that models' inputs and outputs are validly linked, each input and output maps to a ParameterGroup entity. A ParameterGroup is a collection of related Parameter entity data. An Input ParameterGroup entity must match its Output ParameterGroup entity to provide a valid link or vector.

Model inputs and outputs are fully defined before their use in a scenario by authors who will register their models into the IFT-DSS system using the CDL tool discussed in Section 4.1.1. Subsequently, authors can then link these models into scenarios. Currently, the CDL tool is an administrative tool used by system developers to add models and scenarios into the IFT-DSS. Scenarios in the initial versions of the IFT-DSS will be created, loaded, and managed by the database administrator.

The Navigator tool queries this database and uses the results to control the interactive processing of scenario runs and editing of run data. The Executive queries this database for scenario data and compiles this data to control processing. Once completed, the Executive stores performance information in the database.

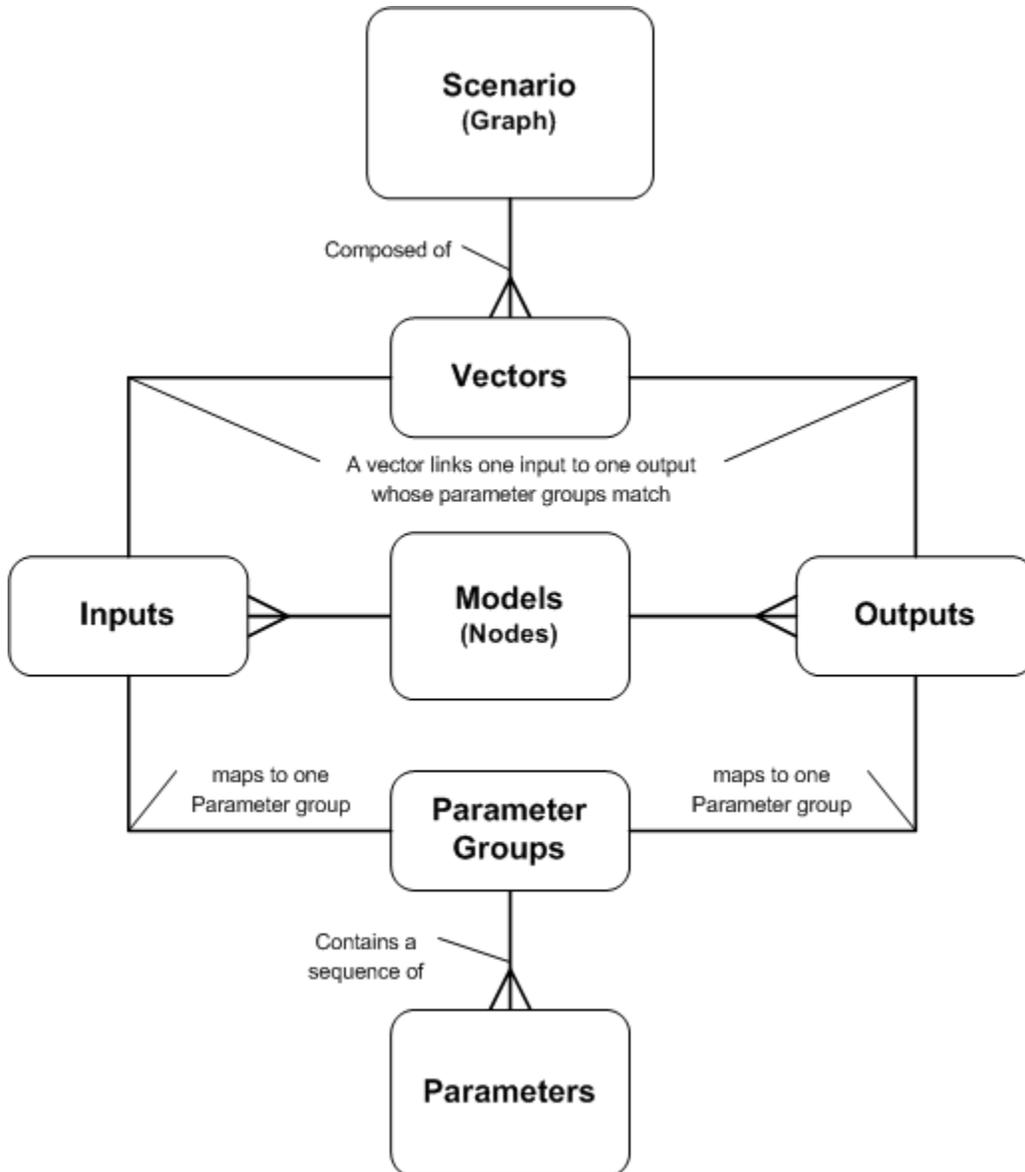


Figure 4-5. Conceptual ERD for the Control Database.

The Navigator queries this database and uses the results to control the interactive processing of runs and editing of data. The Executive queries the database for ALL of its data related to the specified scenario. The Executive then compiles a segment of these data and causes processing to be done. Once completed, the Executive updates the use counters and performance tracking data.

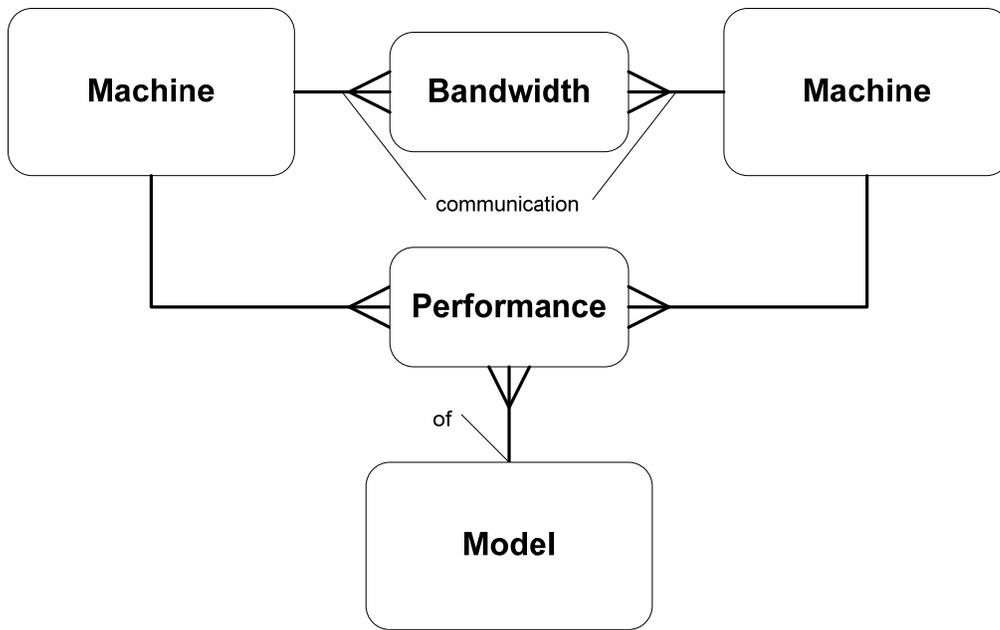


Figure 4-6. Conceptual ERD for subset of the Control Database to facilitate efficient model execution.

Machines

In order for the Executive to decide how the execution of a model should be distributed across available machines to maximize performance, the Control Database has tables (see the conceptual ERD in **Figure 4-6**) describing how well each machine can execute each model (Performance entity). The bandwidth capacity between machines (Bandwidth entity) is also used in this calculation.

Templates

To maintain control over the Navigator's GUI format, the Control Database includes entities to map the vectors, parameter groups, and their parameters in a scenario DAG to UI templates (see **Figure 4-7**). These templates are HTML documents with placeholders for parameter inputs. This arrangement makes it possible to have customized GUIs for certain scenarios. For example, given a model that takes in winds, moistures, and weather parameter groups, we may have a scenario where all three of those parameter groups appear to the user on a single screen.

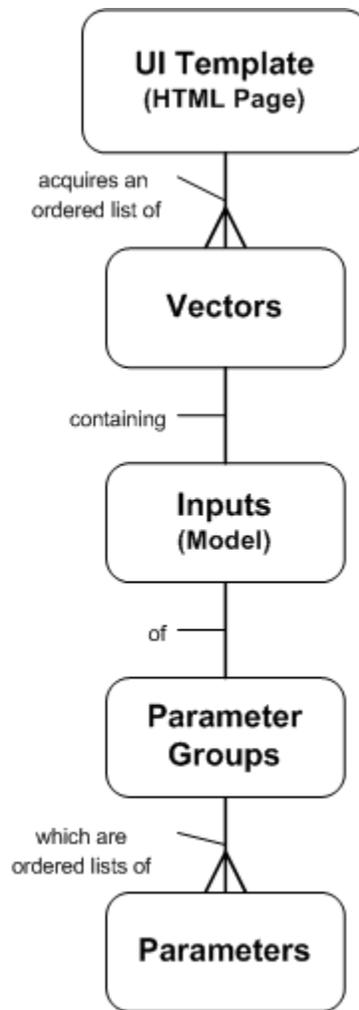
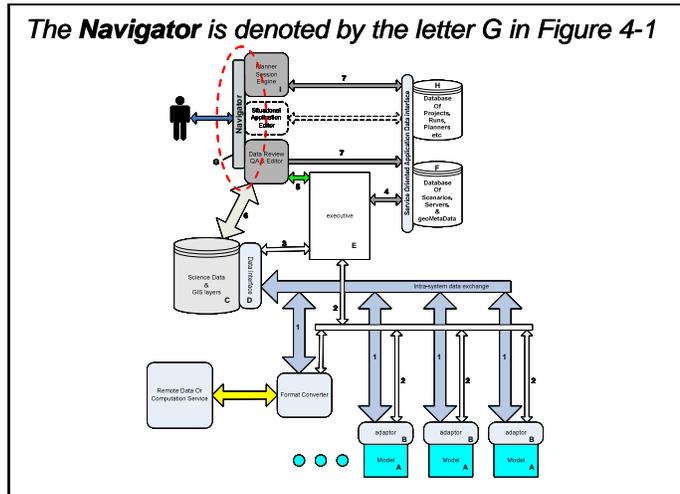


Figure 4-7. Conceptual ERD of user interface HTML templates.

4.1.7 Navigator

The Navigator is a web application that has two key functions: (1) to enable users to visualize and explore their data in the form of an interactive map, and (2) to allow users to edit data values. The tool will function as a data input mechanism (as discussed in Section 3) for user-generated modeling scenario runs. Mapping features allow users to zoom in and out, pan in different directions, switch between layers and overlays, print maps, and save map images.



In addition, users have the option to click on a point in the map or draw a polygon to view attribute data for that point or polygon area. Users will also have the ability to edit map attribute data associated with the GIS map layers in future versions of the IFT-DSS.

As shown in **Figure 4-8**, the interactive Navigator tool consists of four key subcomponents: (1) a set of underlying GIS map layers, (2) a set of software system instructions that tell the Navigator how to construct the map requested, (3) a map server software component that builds the map, and (4) a subcomponent to add interactive features.

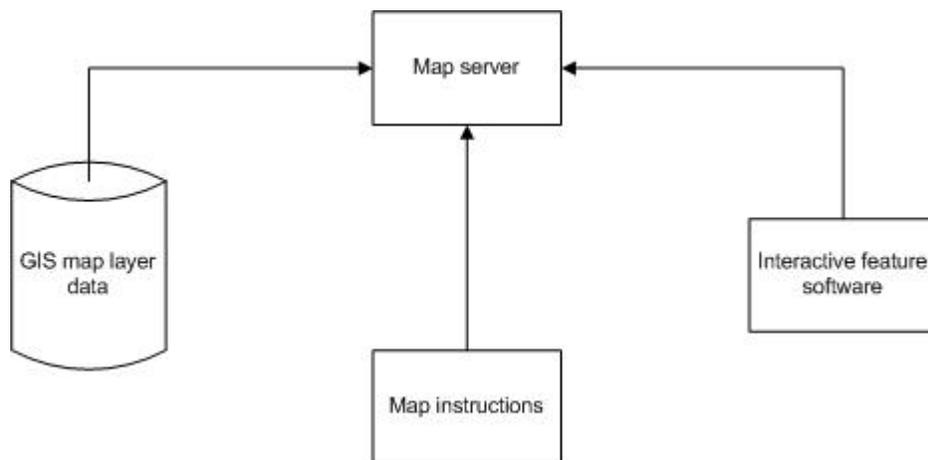


Figure 4-8. Subcomponents of the Navigator.

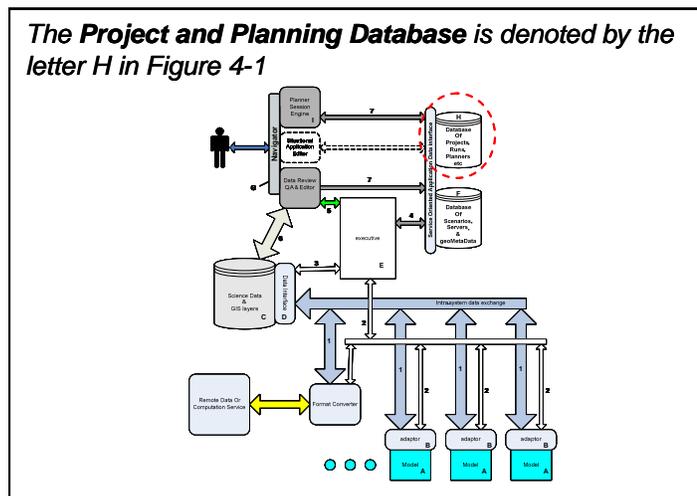
The data source provides the underlying GIS map layers (i.e., political boundaries, relief, land marks, etc.) for the base map. The map server software will support several data input formats for vector and raster data. For raster data, Tiff/GeoTif and EPPL7 are supported by

default, but other formats such as GRASS, Jpeg2000, and ArcInfo Grids are supported with the help of the Geospatial Data Abstraction Library (GDAL). In the case of vector data, ESRI shapefiles are the default; however, the software will be compiled to support and read from the Scientific Database, GML files, delimited text files, and more with the vector data access portion of the GDAL library (called OGR).

A set of instructions, in the form of a mapfile, tells the map server software component where the data source is and defines how the map should be drawn and displayed. The mapfile is where the map layers and the style of the map are specified. The map server is a software component that constructs a map image given a data source and a set of instructions. It has been implemented as a Web Map Server, generating maps in response to Web Map Service (WMS) requests. The interactive features of the map are provided by a software subcomponent. This subcomponent allows maps to be displayed in a web browser with no server-side dependencies. It also implements industry-standard methods for geographic data access, such as the OpenGIS Consortium's WMS and Web Feature Service (WFS) protocols.

4.1.8 Project and Planning Database

The Project and Planning Database stores administrative data about fuels treatment projects. The primary entities are the User and the Project. Users can author models, scenarios, and projects. Users establish permissions for authored projects to control access by other users to their projects and the reports they have generated. Users can act as planners and use projects they have access rights to. A conceptual ERD modeling these relationships is shown in **Figure 4-9**.



This database supports the Planner web application described in Section 4.1.9. Future versions of the IFT-DSS will allow Projects to be accessed by one or more planners. Projects are created by one or more authors, as are the models and scenarios that make up a project. Project authorships are used for managing three kinds of permissions: Project Owner (can do anything), Contributor (fewer privileges than an owner), and Read-Only (can view a project's details but not make changes to it).

A project will also need to reference data in the spatial data system. For example, a project has its area of interest (e.g., Yellowstone National Park). The properties of this area (e.g., elevation, slope, etc.) are stored entirely in the spatial data system, but a project must reference this area uniquely so that its properties can be retrieved from the spatial data system. See Section 3 for a description of the user experience that corresponds to this database.

After completing the scenario the user can view results, generate reports, and share the project with other users. A more detailed description of the user experience is presented in Section 3.

Report Generation

When a project reaches a certain point, planners will be ready to compile a report for the project. In the case of a Prescribed Fire Plan report, there exists a Word document template. Early versions of the IFT-DSS have limited ability to generate reports; however input and output data can be easily exported to Microsoft Office products (i.e., Excel). In future versions of the IFT-DSS, selected sections of a Prescribed Fire Plan report will be provided as blocks of free-form text. The Project and Planning Database will retain this text so that planners can enter the text while logged on to the IFT-DSS and save it for later sessions and ultimately for report generation.

Some of the sections of these reports contain values produced or consumed by certain models (e.g., flame length). Those values, stored in the spatial data system, will be retrieved at report generation time. Users can capture tables, map and graph snapshots for their reports using the Navigator. For the specific case of a Prescribed Fire Plan report, users can choose to associate such outputs with a report so that the generated Word document will include these snapshots.

4.2 INTERFACES

This section describes the communication among the principal components of the SOA. The interfaces, numbered as in Figure 4-1, are (1) Model Data Exchange, (2) Model Control, (3) Sequencer Control and Monitoring, (4) Executive Scenario Polling and Feedback, (5) Executive Launching and Progress Reporting, (6) Map and Data Presentation Interface, and (7) Web Page Generator and Database Interface. The interfaces can also be listed as follows:

- | | | | |
|----|----------------------|---|---|
| 1. | Data Interface | ↔ | Model Adaptors |
| 2. | Executive | ↔ | Model Adaptors |
| 3. | Executive | ↔ | Data Interface |
| 4. | Executive | ↔ | Control Database |
| 5. | Executive | ↔ | Navigator |
| 6. | Navigator | ↔ | Scientific Database |
| 7. | Navigator or Planner | ↔ | Control or Project and Planning Databases |

The remainder of this section describes the communications protocols for each interface, including a description of each message that passes between components.

point information required by the model adaptors for hooking up the plumbing between adjacent models in the scenario. A connection between two models is actually between specific input and output streams of the two models, each stream communicating a different data set to or from its model. Moreover, each stream may have multiple connections. When a model completes execution, it notifies the Executive and sends along performance metrics that the Executive will relay to the Control Database and use to compile future work. These messages between the Executive and the Model Adaptors are shown in **Table 4-6**.

Table 4-6. Messages between the Executive and the Model Adaptors.

<u>Messages from Executive to Model Adaptors</u>	
RunModel (inputs, outputs) – invokes the model.	
Parameters:	
<u>inputs</u> – list of input model stream connection info	
per each input:	
stream ID, IP addresses and port numbers, phase	
<u>outputs</u> – output model stream connection info:	
stream ID, IP addresses and port numbers, phase,	
multiplexing polynomial	
Returns: status message	
GetProgress – requests progress report	
Returns: progress message	
<u>Messages from Model Adaptors to Executive</u>	
NotifyComplete (runstats) – reports runtime performance measurements.	
Parameters:	
<u>runStats</u> – runtime performance metric for this model on this network node	

4.2.3 Sequencer Control and Monitoring Interface

The sequencer control and monitoring interface allows communications between the Executive and the Data Interface. The interactions between the Data Interface and the models (included in Section 4.2.1) are complex and critical to the operation of the system. The compiler process and execution of the scenarios are realized by the conversation between the Executive and the Data Interface. The protocol that enables this conversation is described in **Table 4-7**, separated into execution messages and compiler messages.

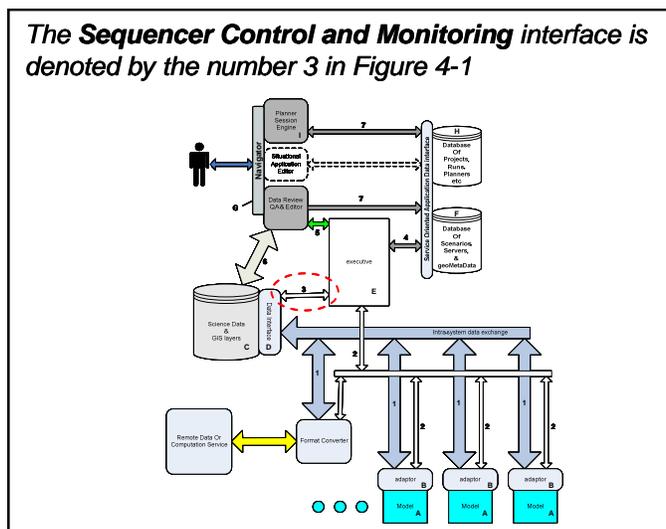


Table 4-7. Messages between the Executive and Data Interface.

<u>Messages from Executive to Data Interface</u>
<p>Startup - gets Data Interface ready to receive messages. Returns: status message</p>
<p style="text-align: center;">--- SCENARIO EXECUTION MESSAGES ---</p>
<p>EmitLayerRegion (maskID, layerID, connections, MPs) - tells Data Interface to send the part or all of a map layer that is to a set of streams, supplying each stream with its "fan-out" Parameters: <u>maskID</u> - the ID of the mask that specifies the region of the layer <u>layerID</u> - the ID of the layer <u>connections</u> - list of IP addresses and port numbers to emit to <u>MPs</u> - list of multiplexing polynomials, one per connection Returns: status message</p>
<p>ListenForResultLayer (maskID, layerID, connections, MPs) - tells Data Interface to send the part or all of a map layer to a set of streams, supplying each stream with its "fan-out" Parameters: <u>maskID</u> - the ID of the mask to apply to result <u>layerID</u> - the ID of the layer in which to store the result <u>connections</u> - list of IP addresses and port numbers to listen from Returns: status message</p>
<p style="text-align: center;">--- COMPILER MESSAGES ---</p>
<p>ApplyAperture (aperture, maskID) - tells Data Interface to mark, in the specified mask, the cells that surround the already-marked cells, which are necessary to accommodate the specified aperture. Parameters: <u>aperture</u> - the dimensions involved in the map layer and their windows of interest <u>maskID</u> - the ID of the mask to modify Returns: status message</p>
<p>MakeMask (input layer, output layer) - tells Data Interface to calculate and store in the Scientific Database a mask raster that specifies which cells in some data input layer are older than the same cells in the corresponding data output layer. Parameters: <u>input layer</u> - data map input layer <u>output layer</u> - data map output layer Returns: the ID of the generated mask</p>
<p>JoinMasks (maskIDs) - creates a new mask that is a combination of the specified masks. Parameters: <u>maskIDs</u> - the IDs of the masks to join Returns: the ID of the generated combination mask.</p>
<p>GetMaskSize (maskID) - queries for number of marked cells in the specified mask. Parameters: <u>maskID</u> - the ID of the mask to query Returns: the number of marked cells</p>
<p>DeleteMask (maskID) - removes the specified mask from the Scientific Database. Parameters: <u>maskID</u> - the ID of the mask to delete</p>
<p><u>Messages from Data Interface to Executive</u></p>
<p>GetProgress - requests progress report</p>
<p>Returns: progress message</p>

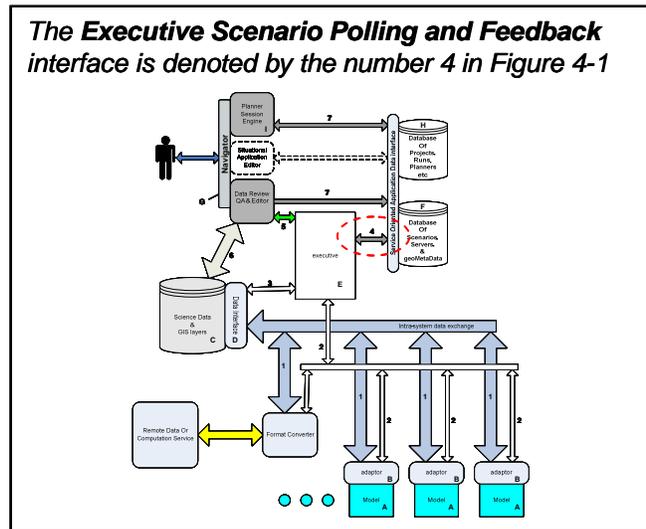
4.2.4 Executive Scenario Polling and Feedback Interface

The Executive scenario and polling interface allows communication between the Executive and the Control Database. When a user selects the output of a model in the Navigator, the Executive is told the scenario in question and the model's output that was selected. Subsequently, the Executive will query the Control Database for the data it needs to create a DAG of the scenario (a node list and an edge list).

The Executive also queries the Control Database for information about machines that are available to execute the necessary models for the given scenario. With that information, the Executive compiles an optimal execution plan for the appropriate scenario segment and then initiates execution.

After execution of the segment is complete, the Executive reports statistics back to the Control Database about how well machines executed models in order to improve performance for future runs.

The protocol across this interface is SQL over the wire (e.g., JDBC). Because the Executive uses recursion, SQL is used to get a simple and complete node and edge list of the graph from the Control Database and the Executive does the rest of the work in a procedural, imperative language (e.g., C++ or Java).



4.2.5 Executive Launching and Progress Reporting Interface

The Executive launching and progress reporting interface allows communication between the Executive and the Navigator. Some user actions in the Navigator will cause the Executive to run and receive progress feedback indicated by the arrow numbered (5) in Figure 4-1. This is triggered by a message from the Navigator that causes the Executive to execute a scenario segment. This process consists of the Navigator sending an RPC to the appropriate machine to launch the Executive.

This RPC also passes the scenario ID and the point in the overall process where processing is needed. Next, the Executive compiles the proper scenario segment. Once execution begins, the Executive will periodically report progress updates to the Navigator for GUI presentation. In the case of a long execution, the Executive will notify the Navigator so the user knows he or she may log off and receive notification (e.g., via email) when execution has reached completion.

The mechanism for communication between the Navigator and the user's web browser is the Persistent Communications Pattern for Asynchronous JavaScript and XML (AJAX).

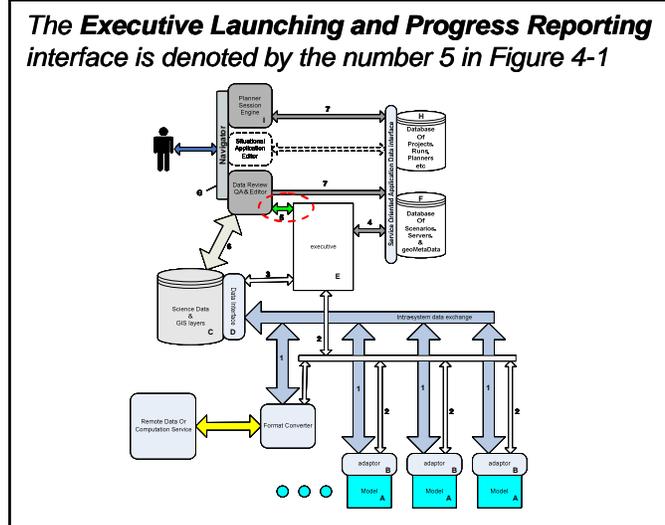
4.2.6 Map and Data Presentation Interface

The map and data presentation interface allows communication between the Navigator and the Scientific Database. Since a WMS-compliant geodata server is used, the content of this interface is WMS. WMS is a standard communications protocol for map servers. It allows the use of data from several different servers, effectively creating a network of map servers from which clients can build customized maps.

WMS servers interact with their clients using the HTTP protocol. The WMS specification defines certain request types, and for each of those requests defines a set of query parameters and associated behaviors. The WMS-compliant server is able to handle at least two types of requests, GetCapabilities and GetMap. A GetCapabilities request returns an XML document containing metadata about the map server. A GetMap request returns an image of a map based on the input information. Support exists for other requests such as GetFeatureInfo, DescribeLayer, and GetLegendGraphic. A GetFeatureInfo request returns information about features at a query location. This specific request type is used to get data when a user clicks on the map. The DescribeLayer request returns an XML description of one or more map layers. Lastly, the GetLegendGraphic request type returns a legend image (icon) for the requested layer including labels.

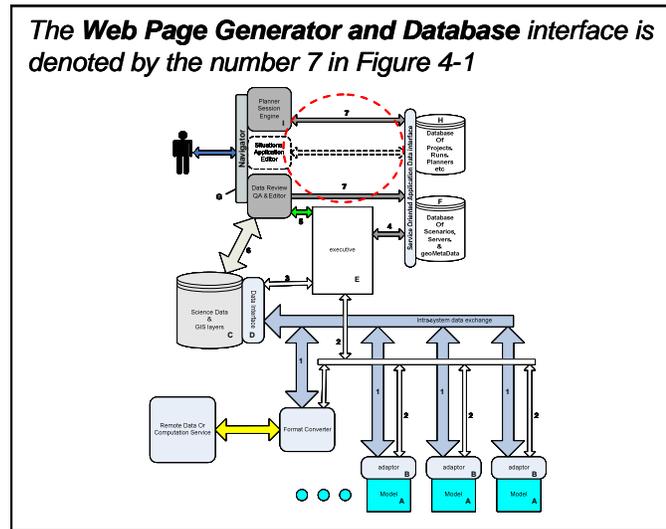
The data editor program (calibration tool) will request maps or other graphical data from the Scientific Database. The Scientific Database will act on that request and deliver a map (image) or other data. A fairly high bandwidth connection is needed because multiple users may be requesting maps nearly simultaneously. Importantly, when the user edits the map or enters data, the results are passed into the Scientific Database.

In addition to communicating map layer data, this interface must support the communication of non-spatial data and a significant amount of system control. When a new run is defined, information is sent to the Scientific Database about the run's inputs so that, if these data sets do not already exist, the Scientific Database can generate images with default data or information extracted from another source.



4.2.7 Web Page Generator and Database Interface

The Web Page Generator and Database Interface governs the communication between the application components (Navigator and Planner) and the relational databases (Scenario and Project and Planning). The Control Database stores information about each scenario. Each scenario may have one or more pages associated with it, and each page may have one or more inputs (e.g., text fields, radio buttons, checkboxes, drop-down selectors, maps) associated with it. The inputs provide the content for each page, and each page is a step in the scenario workflow.



The pageCreator is an object between the Navigator user interface and the Control database that handles the page generation for a specific scenario. On the basis of the scenario chosen by the user, the pageCreator object queries the Control database for pages associated with that specific scenario. For each page, the pageCreator object queries the database for an HTML template (controlling screen layout and design) and inputs to be included on the screen. This allows most pages of the Navigator to be dynamically generated and populated from the database based on user selections.

The design styling of each Navigator screen, including placement of inputs, is controlled by an HTML template. This template is stored in the Control database and associated with a specific scenario/page, enabling customization of individual screens. The template contains placeholder variables, which are replaced with appropriate inputs at runtime. This model allows for generic types of templates to be used in the future. In addition, individual input styles are controlled by values stored in the database for each input (e.g., width, font, minimum/maximum characters, etc.).

The Project and Planning database stores information about projects, runs, and associated scenarios. The Planner user interface interfaces with the Project and Planning database in a more mundane, create-retrieve-update-delete (CRUD) fashion than does the Navigator in its interface with the Control database.

4.3 BEHAVIORS AND INTERACTIONS

This section presents the principal interactions among components and, in combination, the behavior of the system as a whole, in the form of swim lane diagrams. The swim lane is a visual element used in process flow diagrams that depicts what or who is working on a particular subset of a process. The swim lane flowchart differs from other flowcharts in that processes and

decisions are grouped visually in lanes. Parallel lines divide the chart into lanes, with one lane for each person, group, or subprocess. Lanes are labeled to show how the chart is organized.

4.3.1 Project Workflow

Figure 4-10 illustrates the interaction between the Planner/Navigator, a planner (end user), the Executive, and services (model environment, models, and Scientific Database) for an entire project.

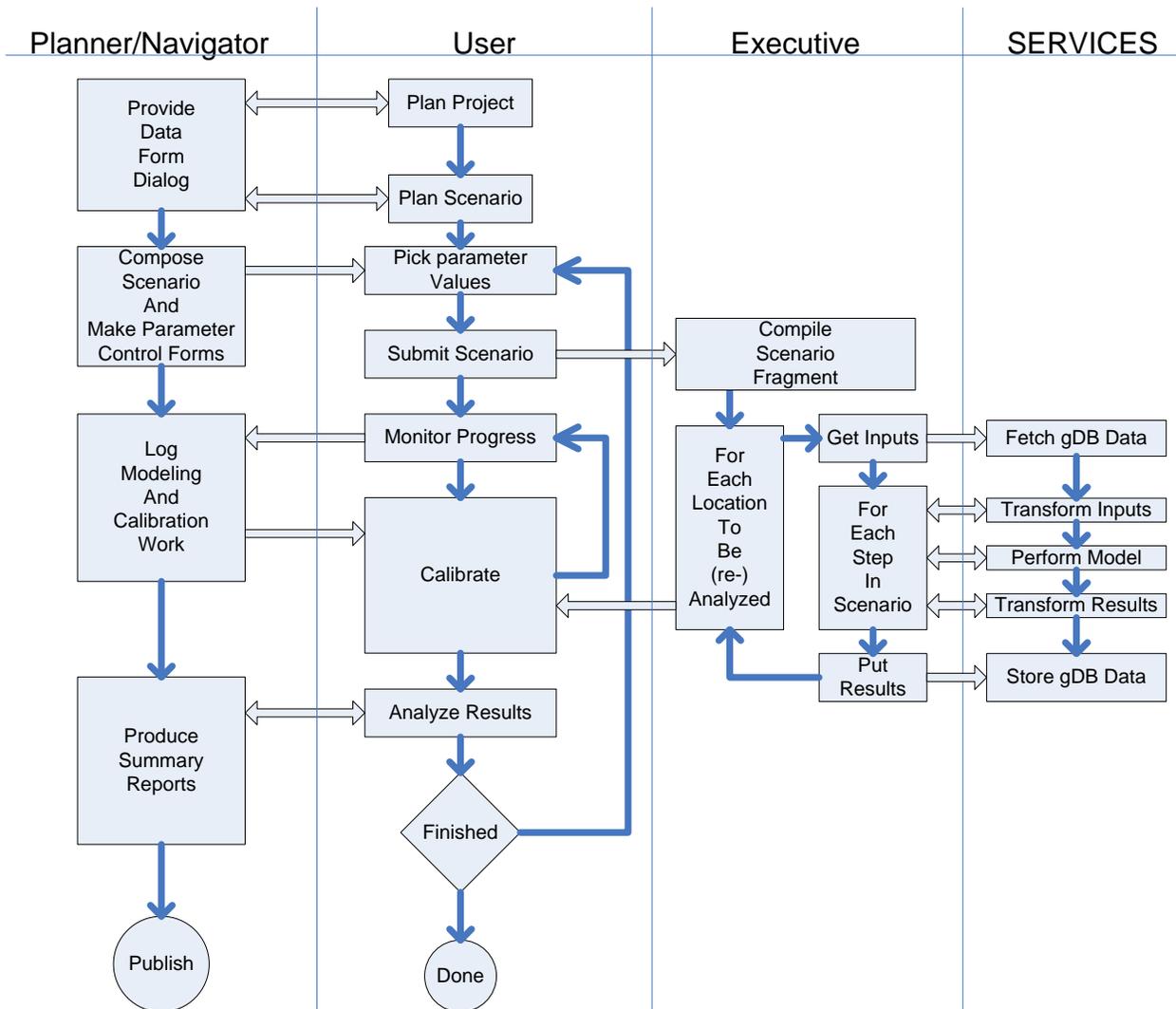


Figure 4-10. Overview of run execution.

4.3.2 Executive Scenario Compilation

The swim lane diagram in **Figure 4-11** depicts the communication between the Navigator, Executive, and Control Database when a scenario segment needs execution.

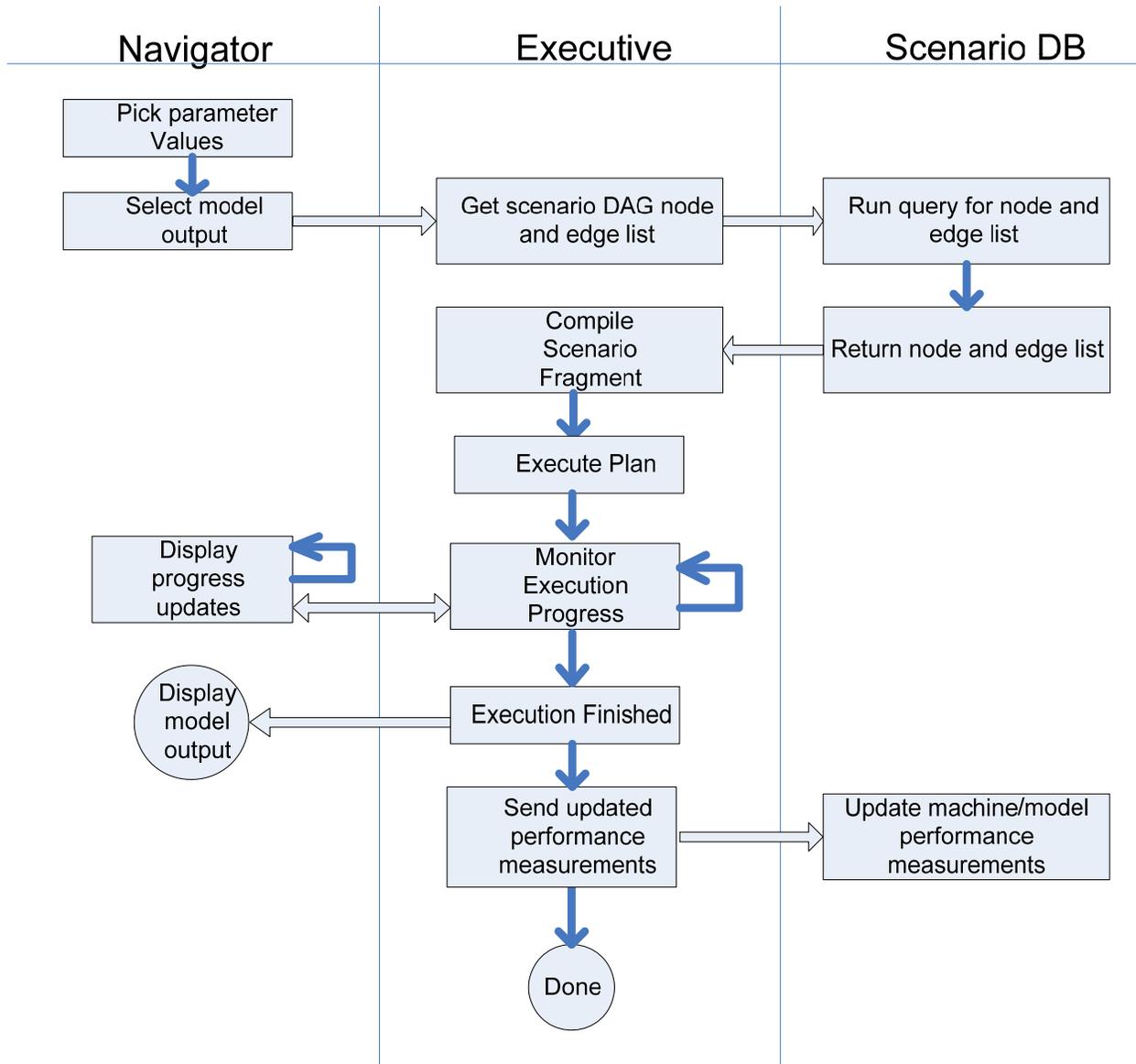


Figure 4-11. Swim lane diagram of the Navigator, Executive, and Control Database interaction.

4.3.3 Executive Orchestration of Model and Scientific Database Interaction

Figure 4-12 illustrates the interaction between the Executive, Data Interface and Scientific Database, and models, beginning with Executive compilation of a scenario segment.

This interaction occurs after user action caused the Executive to begin and is essentially a drill down into the “Execute Plan” step in the Executive swim lane in Figure 4-11.

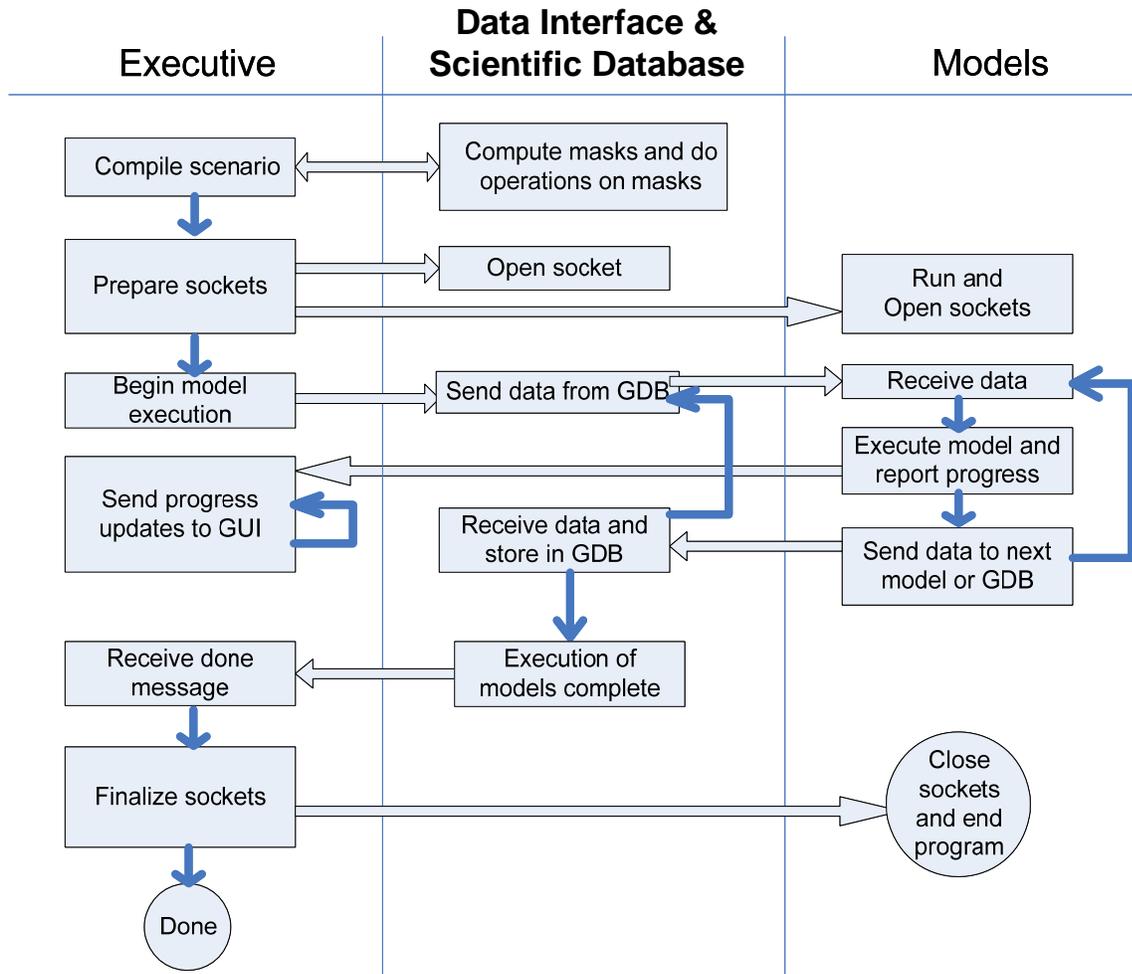


Figure 4-12. Swim lane diagram of Executive, Data Interface and the Scientific Database, and models interaction.

4.3.4 User Interaction with the Graphical User Interface and Database

Figure 4-13 illustrates the interaction between the user, the Navigator, the pageCreator object (which handles the generation of scenario specific web pages), and the database that holds the data necessary to create the web pages.

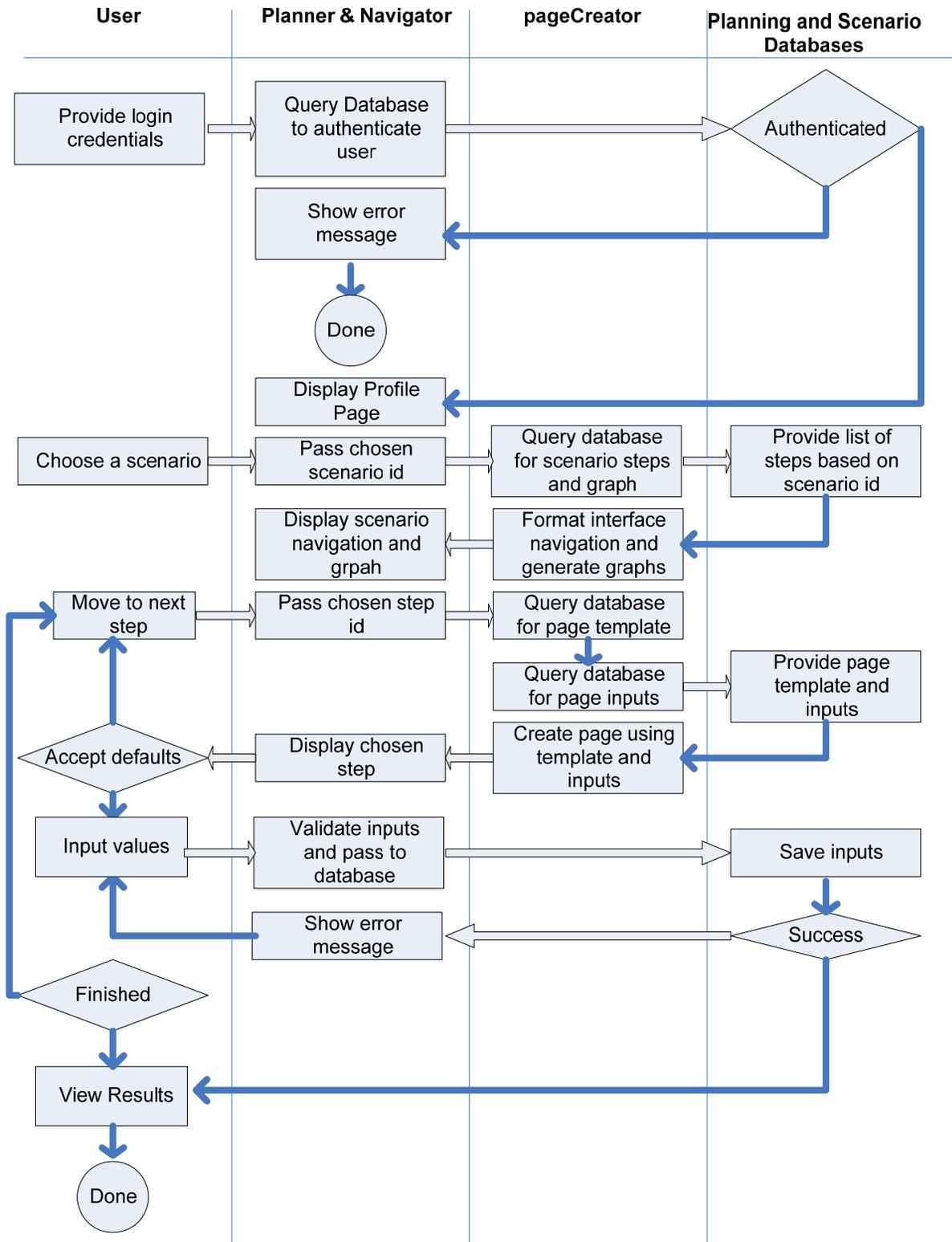


Figure 4-13. Swim lane diagram of the user, Navigator (GUI), page creator, and control database interaction.

4.3.5 User Interaction with the Web Map Service and the Scientific Database

Figure 4-14 shows how the User, WMS, and Scientific Database interact with each other.

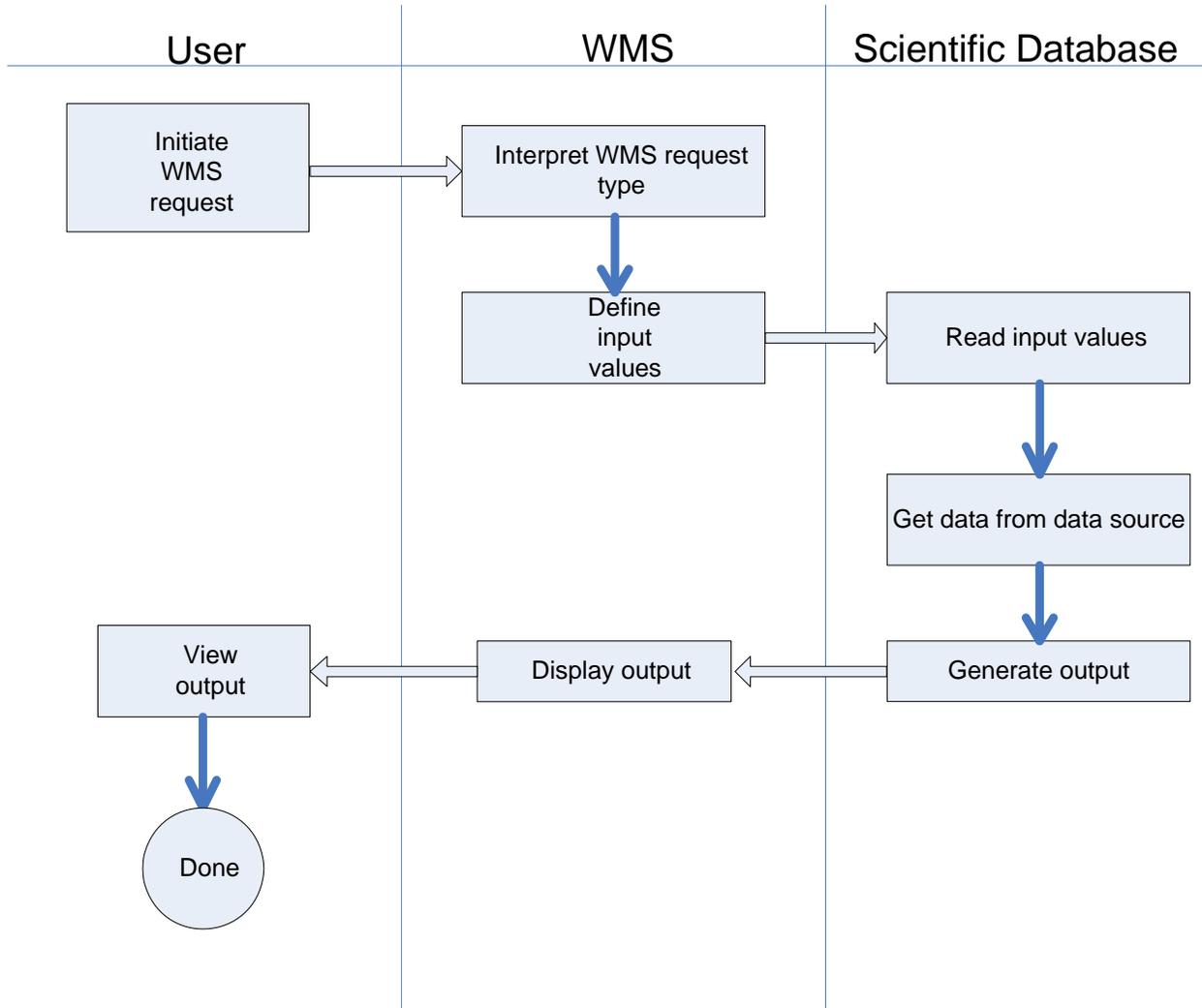


Figure 4-14. Swim lane diagram of the user, WMS, and Scientific Database interaction.

5. TECHNOLOGIES

This section discusses specific technology choices for each component in the IFT-DSS version 0.3.0. Flexibility in meeting the needs of the user communities is paramount to the success of the IFT-DSS. Therefore, the hardware and software technologies selected were primarily considered in the context of user community needs and to show the flexible nature of the system. It is intended that this document be a “living document” because the design and the implementation technologies may change during development. Such changes will be noted in revisions to these specifications.

Given the SOA of the system, technology decisions can be made separately for most components. Moreover, the architecture can accommodate a wide range of platforms and network topologies, from all components running on a single machine to a dedicated machine or cluster of machines for each component. Specific technology choices are based on a combination of criteria, including quality, ease of use, interoperability, reliability, performance, support, and cost.

We defer specifying network topology and many hardware requirements until we are closer to deployment, at which time we will have gained knowledge of optimal configurations.

5.1 MODELS AND MODEL ADAPTORS

The models are the components that process the parameter data. The Model Adaptors are the components that enable models to be “plugged in” and that direct parameter data into and out of the models. Models and Model Adaptors are described in sections 4.1.1 and 4.1.2. Each model employed in the system will always run in process with, and hence on the same box as, a Model Adaptor.

OPERATING SYSTEM: **Windows and Linux**

It is incumbent on the system to accommodate models embedded in pre-existing programs that were developed to run on specific OS platforms, so both of these platforms must be deployed to host the models.

PROGRAMMING LANGUAGE: **Java, C++**

Wrapped models (Type B in section 4.1.2) are in pre-existing executable programs, and external models (Type C) are services, so the issue of programming language for these two methods is moot. Subclassed models (Type A in section 4.1.2) require that a specific (object-oriented) programming language or languages be supported. We currently support **Java**. The IFT-DSS will also support **C++**, **Python**, and other languages in the future. This means that two versions of the model adaptor host program (which make up the model parent class) shown on the left side of Figure 4-3 have been developed using **Java**.

5.2 SCIENTIFIC DATABASE

The primary function of the Scientific Database is to manage and serve the data that are generated and used by the models. Whereas a large fraction of these data are spatially distributed, the unique features of a GIS database are needed. Conversely, a large fraction of these data are aspatial. The data server must effectively combine the features of a GIS database with those of a fast data server, and be able to present both types of data. The Scientific Database is also used to manage and display the spatial data in the form of interactive maps as referenced in sections 4.1.3 and 4.1.7.

OPERATING SYSTEM: **Linux**

There exists a large quantity of open source resources that address similar problems. We also have access to similar tools that are currently in use by members of our user community (e.g., WFDSS). **Linux** is the preferred OS for these resources and tools.

GIS SOLUTION: **MapServer, OpenLayers**

MapServer in combination with **OpenLayers** has been selected as the GIS data and mapping solution. MapServer is an open source platform used to publish spatial data and interactive mapping applications to the web. It was chosen because of its many features, which include advanced cartographic output, support for popular scripting and development environments, cross-platform support, support of numerous Open Geospatial Consortium (OGC) standards, availability of a multitude of raster and vector data formats, and map projection support. OpenLayers is a pure JavaScript library used to display map data in web browsers, with no server-side dependencies.

We have developed prototype map products for visualization in Google Earth and eventually in FireGlobe.

5.3 DATA INTERFACE AND EXECUTIVE

These components are the inner “clockworks” of the system. The Data Interface interacts with the Scientific Database to pass input data to the models and to store model output data. The Data Interface also provides data filtering calculation services to the Executive. The Executive is the program that compiles scenarios and triggers execution of scenario “runs.”

OPERATING SYSTEM **Linux**

PROGRAMMING LANGUAGE **Java**

Linux and Java are both robust, reliable, enterprise-class technologies, and well suited for these non-GUI components. Java is cross-platform, which means that Java-based components can easily be re-used for any subsequent deployments to other OS platforms.

5.4 PLANNING AND CONTROL DATABASES

OPERATING SYSTEM **Windows**

RDBMS **Microsoft SQL Server**

The Planning and Control Databases require a relational database management system that can scale with future growth. Microsoft SQL Server was selected as the database for the IFT-DSS proof-of-concept system.

5.5 NAVIGATOR AND PLANNER

These are the user-facing parts of the system; they present a browser-based, interactive graphical user interface. The Navigator and Planner interact with the control and planning databases to manage projects, work scenarios, and so on. The Navigator interacts with the Scientific Database and the Executive to view, edit, and generate new parameter data.

OPERATING SYSTEM **Windows or Linux**

WEB SERVER Apache **Tomcat**

Tomcat is a common environment for hosting web applications. STI has experience developing, maintaining, and hosting web applications in this environment.

PROGRAMMING LANGUAGE **Java, JSP, HTML**

Java was selected as the server-side language for several reasons including the following: it has become a commonly used and understood language, and this choice will facilitate future modifications of the system by various parties; it can be run on multiple platforms and web servers; and there are no software licensing fees.

5.6 HARDWARE

IFT-DSS version 0.3.0 is currently being hosted on two identical Dell PowerEdge R410 servers with the following specifications:

- Dual quad-core E5520 Xeon processors
- 2.26 GHz clock rate
- 24 GB memory
- 1.5 TB RAID5

5.7 FUTURE SOFTWARE TOPOLOGY FOR THE IFT-DSS

Figure 5-1 illustrates the software topology for the first production version (version 1.0) of the IFT-DSS scheduled for release in June 2011.

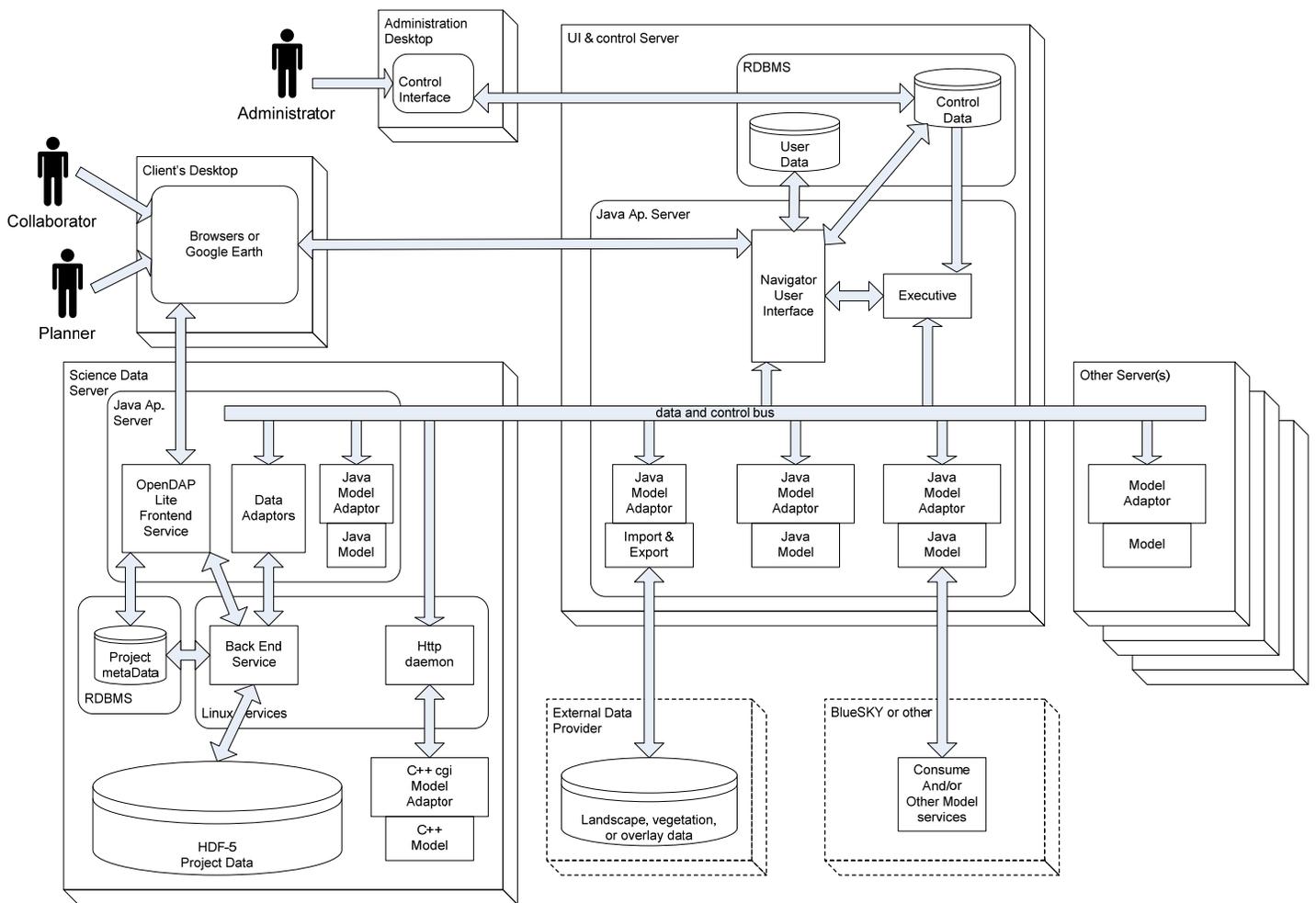


Figure 5-1. Software topology for the first production release (version 1.0) of the IFT-DSS scheduled for release in June 2011.

6. REFERENCES

- Anderson C.B., Dye T.S., Ludewig S.A., Chan A.C., Gray E.A., and Prouty J.D. (2002) Functional design specification document for the AIRNow technical user web site. Design specification document prepared for U.S. Environmental Protection Agency Office of Air Quality Planning and Standards, Research Triangle Park, NC, by Sonoma Technology, Inc., Petaluma, CA, STI-902087-2287-DSD, December.
- Drury S.A., Rauscher H.M., Raffuse S.M., and Funk T.H. (2009) Refined work flow scenarios and proposed proof of concept system functionality for the interagency fuels treatment decision support system. Final report prepared for The Joint Fire Science Program, Boise, ID, by Sonoma Technology, Inc., Petaluma, CA, and Rauscher Enterprises LLC, Leicester, NC, STI-909029.02-3655-FR, July.
- Funk T.H., Corman R.G., Reed J.E., Raffuse S.M., and Wheeler N.J.M. (2009) The Interagency Fuels Treatment Decision Support System (IFT-DSS) software architecture. Software architecture design prepared for the Joint Fire Science Program, Boise, ID, by Sonoma Technology, Inc., Petaluma, CA, STI-908038.04-3565, March.
- Gray E.A., Prouty J.D., Ovard C.D., and Wheeler N.J.M. (2004) Data management system for aerometric data. System documentation prepared for Bay Area Air Quality Management District, San Francisco, CA, by Sonoma Technology, Inc., Petaluma, CA, STI-903730-2602-SD, September.
- Palmquist M.S. (2008) Working summary of the SEI's engagement with the Joint Fire Science Program. Report prepared for the U.S. Department of Defense by the Acquisition Support Program, Software Engineering Institute, Carnegie Mellon University, April.