Modeling and visualization of automated feature extraction workflows

Scott W. Bouma
The University of Montana

MODELING AND VISUALIZATION OF AUTOMATED FEATURE EXTRACTION WORKFLOWS

by

Scott W. Bouma

B.Sc., Montana State University, 1998

presented in partial fulfillment of the requirements

for the degree of

Master of Science

The University of Montana

May, 2006

Approved by:

[illegible]
Chairperson

[illegible]
Dean, Graduate School

5-31-06
Date

Bouma, Scott W., M.S., May 2006 Computer Science

Modeling and visualization of automated feature extraction workflows.

Chairperson: Dr. David Opitz DWO

Automated feature extraction (AFE) from digital imagery is practical technology. An offshoot of artificial intelligence research, AFE software tools have been helping to solve real-world geospatial problems for several years already. Smart AFE software has been shown to drastically reduce the time needed to extract features from digital imagery. Nevertheless, current AFE methods focus on extracting features from imagery a single image at a time, or at best applying a group of AFE processes to multiple images on a very limited and inflexible scale. Since similar workflows often need to be applied to more than one image, a new method is needed which leverages knowledge gained from feature extractions to increase the efficiency of further (similar) extractions. The development of such a method is the focus of this paper.

Current theory from the field of visual programming, as well as existing image-processing workflow visualization software, forms the basis for the development of a new technique which graphically represents AFE workflow models. The new method - implemented as a graphical user interface and named "Feature Modeler" - provides a powerful and flexible, yet easy-to-use tool for applying AFE workflows to multiple images in a robust and efficient manner. In addition, it also provides a vehicle for knowledge-transfer from more experienced AFE analysts to less experienced ones. It may also help foster global collaboration in the arena of automated feature extraction.

Anecdotal evidence from results generated by the Feature Modeler suggests that this new technique may increase the speed of large-scale automated feature extraction by an order of magnitude, at no cost in terms of quality. Further rigorous testing is needed to determine more precisely the expected efficiency gains for various feature extraction problems, but initial results are encouraging.

ACKNOWLEDGEMENTS

No man is an island, and I suppose the same can be said of a thesis. I owe a debt of gratitude to several who have helped me achieve this milestone. First and foremost, I would like to acknowledge my wife for her selflessness and sacrifices over the past two years as I juggled school, work, and family.

I would also like to thank my employers at Visual Learning Systems, particularly Dave and Stuart, for allowing me to do most of the work for this thesis as part of my employment duties. I am also indebted to my colleagues at VLS who helped with suggestions, improvements, and testing of the Feature Modeler.

Finally, I thank my thesis committee members for the time and effort that each has taken in supporting the work of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Geographical information is of vital importance in our knowledge society. It has been estimated that approximately eighty percent of all information involves a spatial component [1, 15]. The acronym GIS (or Geographical Information Systems) has come to represent the broad spectrum of tools and techniques used to gather, organize, and analyze spatial information. GIS, whether in commercial settings, local governments [2], or national security, has been shown to be an effective aid for informed decision-making [9]. Presenting geospatial information to decision-makers in a meaningful way, however, is not an easy task. Broadly speaking, geospatial information must usually pass through three stages in order to become useful information: data collection (by satellites, GPS devices, etc), organization and storage (e.g. image formatting and compression, geo-referencing, insertion into geo-databases), and finally, visualization and analysis (usually with dedicated software packages). The subject of this thesis is to improve the efficiency of one aspect of the final stage, namely automated feature extraction (AFE).

AFE is a technique used to automatically detect features from digital imagery (see Section 2.1.1). This usually involves more than one step, and often the application of a standard workflow. Software tools are available for automatically extracting features, such as Visual Learning System's Feature Analyst. For various reasons, Feature Analyst has emerged as the leading software in the AFE field [10, 11]. However, the current software tools available in Feature Analyst for reusing these standard workflows are limited and inflexible. This thesis addresses this problem by implementing a new graphical user interface (GUI) for visualizing and modeling these workflows, leveraging their inherent knowledge to speed up future extractions.

Thesis Statement: *Modeling AFE workflows will increase the speed of future feature extractions and decrease the user input required, at no cost in the quality of results.*

1.1 Motivation

Development of a GUI to visualizc AFE workflows is motivated by the current lack of any such methods, and the premise that such a tool will be of real practical value in the GIS world. In particular, two practical needs in the current Feature Analyst AFE software solution provide motivation for this thesis:

- The need for a method of applying standard AFE workflows to multiple images, while maintaining enough flexibility to account for variation between images.
- The need for an intuitive mechanism for transferring knowledge from more experienced AFE analysts to less experienced ones.

Advancements in satellite remote sensing are such that high quality color imagery is freely available for most parts of the globe (and several other planets). If these advancements are not mirrored by equal advancements in imagery analysis, however, the dazzling salmagundi of available digital imagery will be of little use in the real world. Since feature extraction is a major part of image analysis, the success of AFE is critical to the continued advancement and usefulness of GIS. "Automated feature extraction is a challenge that must be addressed if LIDAR data are to be widely adopted," says one GIS expert [14].

AFE is more than just a single-step, "one size fits all" process. Each feature extraction problem has its own intricacies, and requires its own solution. A road extraction, for instance, requires a training and classification step, followed by a conversion to lines, and possibly a line-smoothing step to cull extraneous vertices. An inner-city pervious/impervious surface layer analysis, on the other hand, requires an entirely different workflow. These differing requirements mean that an analyst must use professional judgment to develop a workflow which produces the desired results on a single image. Once this is accomplished, however, that same workflow could be applied to similar extraction problems, to produce similar results. In theory, an expert analyst could develop workflow models to be used by junior analysts for high-output production work. Such a technique would take the already-substantial efficiency gains of AFE and increase them by orders of magnitude in many situations. In order for such a goal to be realized, however, a method of modeling and visualizing these workflows needs to be developed which combines flexibility for workflow development and ease-of-use for non-expert analysts. Such a method should be implemented as a GUI, since most GIS analysts are adept at processing visual information. Developing such a GUI for Feature Analyst is the goal of this programming thesis project.

Best-practice techniques in the field of visual programming provide an excellent basis for developing a method to visualize AFE workflows with adjustable parameters and inputs. Feature Analyst AFE workflows can be visualized as individual process modules linked in a linear program, with the inputs and outputs of each module visualized as user-definable parameters. Basic control sequences, such as allowing an analyst to decide if a particular step was successful, can also be modeled using visual programming techniques, to provide a powerful and interactive AFE "program."

1.2 Goal

The goal of this thesis is to implement a GUI which allows user interaction with existing AFE models, as well as development of new models. The goal of the GUI (hereafter referred to as the **Feature Modeler**) is to be intuitive, have simple-to-understand base functionality, and be flexible enough to handle a wide variety of extraction problems. Feature Modeler should work with existing AFE models and be architectured in a "forward-compatible" manner, which will allow the development of new image processing modules that can still be modeled and visualized in the same manner, and can even be integrated into existing AFE models. Such a software tool will go a long way towards addressing efficiency issues associated with AFE production workflows.

1.3 Thesis Organization

The remainder of this thesis is organized as follows:

- **Chapter 2** discusses related literature and software, explains the concept and structure of a Feature Analyst AFE model, and outlines the general steps of the software development process used to develop Feature Modeler.
- **Chapter 3** steps through the details of the Feature Modeler's development, and gives an overview of the finished product.
- **Chapter 4** discusses the strengths and weaknesses of the Feature Modeler as they relate to the original goals and requirements. The chapter ends with suggestions for future work in the area of AFE workflow modeling and visualization.

CHAPTER 2: RELATED KNOWLEDGE

2.1 Literature and Software Review

This section provides knowledge reviews for three areas: (a) AFE workflows, (b) visual programming techniques, and (c) current image-processing workflow visualization software.

2.1.1 AFE Workflow Modeling in Feature Analyst

The leading commercial AFE software package (Feature Analyst) currently offers thc ability to "batch process" an existing workflow to new imagery [10]. Batch processing is the ability to take knowledge gained by an intelligent software agent in one extraction problem, and apply that knowledge to a different image. At each step in an extraction process, Feature Analyst records the parameters used and appends them to any previous steps, creating a record of the workflow and settings used to arrive at the current results. This workflow can then be applied to additional imagery via Feature Analyst's batch processing functionality. However, this functionality does not allow any change to workflow parameters, and only offers basic trial-and-error modifications. Batch processing can be very effective when used on imagery sets with little or no variation. However, images collected under varying conditions (e.g., different times of year, different sensors, different sunlight conditions) usually show some amount of variation.

In these cases, batch processing cannot consistently produce quality results. In addition, Feature Analyst records workflows step-by-step as the user performs the initial cxtraction, and users are unable to see the resulting workflow; so when it comes time to apply the workflow to a new image, there is no way of knowing for sure that the workflow is correct. It's a bit like typing without ever being able to look at the screen and check for mistakes.

To solve these problems, Feature Analyst needs a graphical AFE workflow visualization tool which allows users to see what they are working with and adjust workflow parameters to account for variations in imagery.

2.1.2 Image Processing Workflow Visualization Software

Although Feature Analyst does not currently have a workflow visualization tool, several image processing software packages do. In particular, the Model Builder in ESRI's ArcGIS software [3] uses visual programming techniques to facilitate graphical development of image processing workflow "programs." Model Builder is an excellent example of workflow modeling and visualization, and forms a basis upon which the current work is loosely modeled.

2.1.3 Visual Programming

Applying visual programming techniques to the field of image processing has been around for several years. In 1994, Koelma et al. developed a prototype visual programming environment for image processing [7]. In that paper, the authors pointed out that the dataflow paradigm is the most appropriate way to represent image processing workflows (as opposed to, for example, state transition diagrams which focus more on object status than data throughput). They also suggested that complex programming constructs such as recursion, data structures, and pointer handling should be avoided, since they would tend to over-complicate the models, and are unnecessary for developing visual image processing programs. In 2002, Keukelaar [6] presented further arguments in support of the dataflow paradigm when developing visual programming interfaces for image processing. These sentiments are echoed by others [8]. In accord with these opinions, the dataflow paradigm was used as the basis for the AFE model architecture developed in this current work.

2.2 Feature Analyst AFE Models

A Feature Analyst AFE model is a sequential record of all AFE processing steps applied to a particular file. These files are either shapefiles or images output from an AFE process. For instance, an automated building extraction performed on an aerial photo of Missoula would result in a group of 2-dimensional polygons (stored as a shapefile), which are displayed as an overlay on the original image. Associated with the shapefile would be an AFE file which records that it is the result of an automated extraction.

Next, suppose that the buildings were processed to remove all the large industrial buildings and only preserve single-family dwellings. The result would be a new shapefile which contained only the smaller polygons from the first shapefile. This new shapefile's associated AFE model would contain a record of both steps in the history of the shapefile's creation. In this way, any shapefile which is the result of feature extraction processing has an associated model that contains the entire feature extraction history for that model. For some feature extraction problems such as pervious/impervious surface classification in urban environments, these AFE workflows can contain many individual feature extraction steps (see Figure 1).

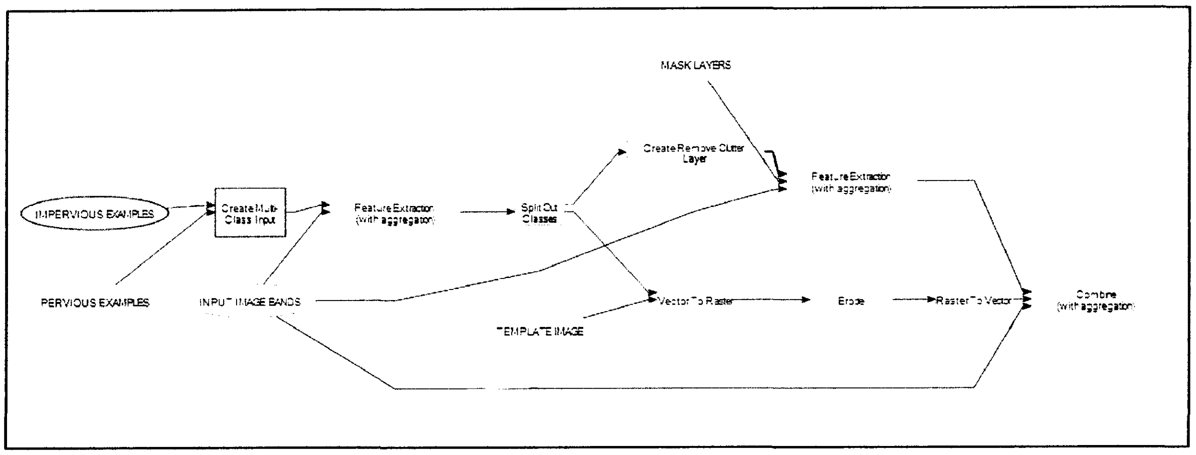


Figure 1: A typical AFE workflow

In addition to storing a linear history for each result file, AFE models also store information about the inputs and outputs of each individual process. In the building extraction described above, information would be stored about two required file inputs: a base image of Missoula, and a file containing a few training examples of buildings. Two other types of information for the process are also stored with the AFE model: (a) user-defined settings, such as the size of the smallest building, and (b) image-specific settings, such as information learned from the image by Feature Analyst's intelligent software agent.

Since the selection of these particular parameters is often a case of trial-and-error on the part of GIS analysts, there is a substantial time-commitment inherent in the information these models contain. Thus, a technique which reuses the information contained in these models could potentially result in increased efficiency when performing subsequent extractions on similar imagery.

Thc Feature Modeler provides the mechanism for this technique. AFE workflows visualized in Feature Modeler can be used as a starting point for similar feature extraction problems (e.g.. extraction of buildings from Great Falls imagery, based on a successful AFE model for Missoula building extraction), allowing an analyst to focus on production speed and quality, leveraging the built-in knowledge from an existing workflow model.

However, the image-specific settings described in (b) above may need to be re-learned on new images, to account for image variation. Thus, there is a need for two separate modes when re-using AFE workflows: silent mode, which exactly reproduces the initial workflow on additional images, and recipe mode, which allows re-learning of image-specific knowledge on new imagery (see Section 3.1 for detailed descriptions of silent and recipe modes).

2.3 Staged Implementation Software Development Process

A software development process is a structure imposed on the development of a software product. The choice of an appropriate software development process is critical to the success of any software project [5]. Many good development processes exist, each with their own particular strengths and weaknesses. Some, like extreme programming, are relatively new and are best suited for small projects that will never be subject to maintenance or integration into larger projects. Others, like the traditional waterfall method [15], have been used successfully for years in the software industry, but can be rather overbearing and inflexible.

Due to the nature of the current problem, the staged implementation process was chosen for the development of the Feature Modeler (see Figure 2). The staged implementation process emphasizes several separate and distinct steps in the development process, but it also allows revisiting of steps when necessary to correct problems that are discovered in subsequent steps. This model works well when the project requirements are well-defined, and is best suited for small development teams where communication (or lack thereof) is unlikely to cause problems. Based on these criteria, the staged implementation process is a good fit for the development of the Feature Modeler. Following is a brief description of each step in the process and how it relates to the Feature Modeler (a full account of each step is given in Section 3.1).

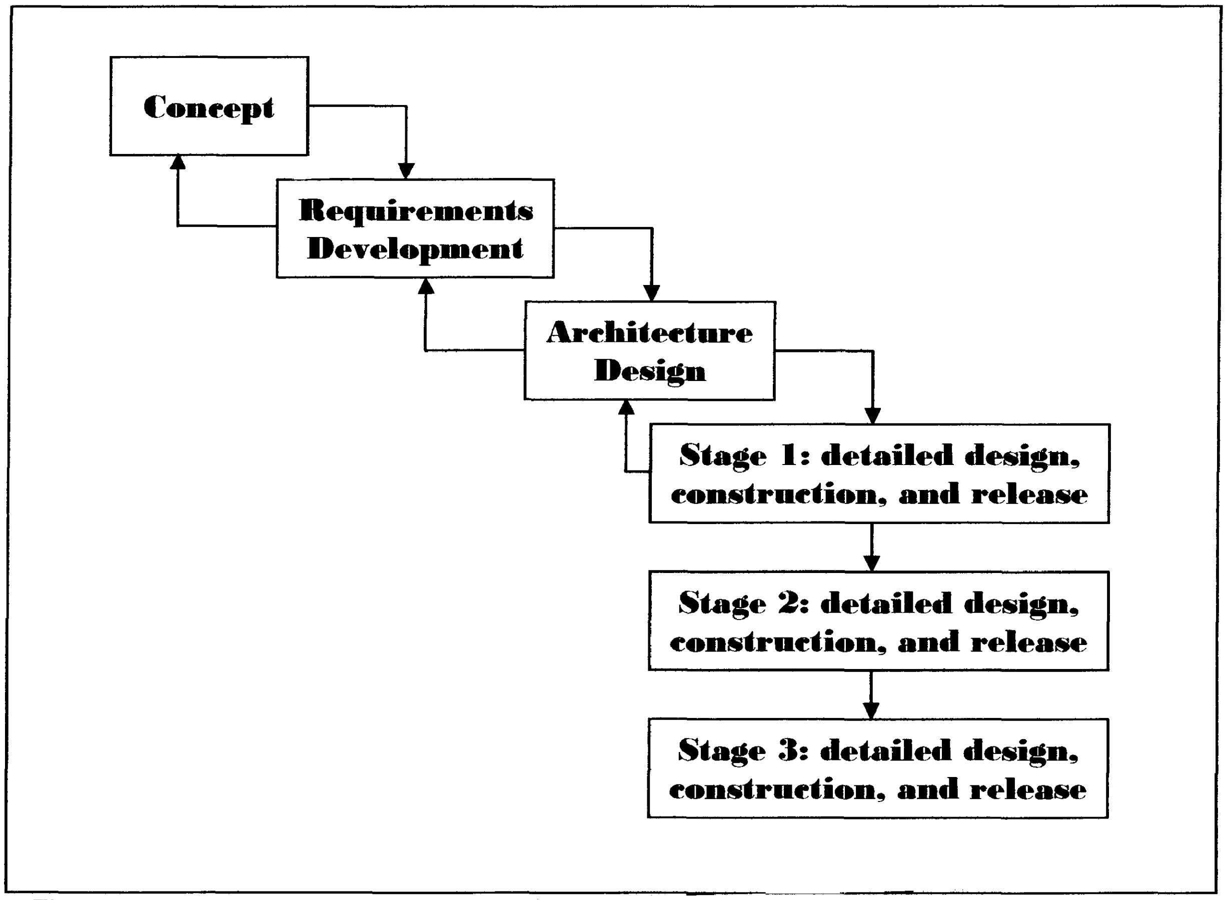

Concept
Requirements Development
Architecture Design
Stage 1: detailed design, construction, and release
Stage 2: detailed design, construction, and release
Stage 3: detailed design, construction, and release

Figure 2: Steps in the staged implementation model of software development (adapted from Henry [5])

2.3.1 Concept – In this step, a preferred concept for the software is defined and examined to determine its life-cycle feasibility and superiority to alternative concepts. In terms of Feature Modeler, this step involved an investigation of various methods for visualization and interaction with data flow models. Section 3.1.1 further discusses the details of this investigation.

2.3.2 Requirements Development – This step involves the development of a complete, verified specification of all required functions, interfaces, and performance for the software product. For the Feature Modeler, this step resulted in a design document developed in conjunction with the U.S. Army ERDC-TEC (who provided funding for the development of the Feature Modeler) specifying the requirements and goals for the project. See Section 3.1.2 for a discussion of these goals and requirements.

2.3.3 Architectural Design – During the architectural design step, specific software engineering patterns are investigated and applied to the problem at hand. Decisions are made about the particular patterns to be used. The type and amount of required testing should be considered, as well as how the final product will be distributed to end users. In the case of Feature Modeler, this step also involved making design decisions about how to fit the system into the existing Feature Analyst architecture. These details are discussed in Section 3.1.3.

2.3.4 Staged Implementations – Each of the implementation stages generally follow a mini-waterfall process consisting of a detailed design phase, coding phase, testing phase, and release. Two complete implementation stages were performed during the Feature Modeler development, details of which are discussed in Sections 3.1.4 – 3.1.5. A possible third iteration is discussed in Section 4.2, with a short general description as follows:

a) Detailed design – This phase is marked by the development (or modification, in later stages) of a specific UML (universal modeling language) diagram defining classes, interfaces, and their relationships to one another. Also, a testing plan should be developed adequately addressing the requirements of the project. Section 3.1.3 discusses the Feature Modeler's general architecture, as well as its relationship to the specific workflow models and image processing modules which it visualizes.

b) Coding phase – Coding is where "the rubber meets the road" in any development process, but the coding phase relies heavily on the results of the previous steps in order to produce a high-quality, maintainable product. Details of the coding stages are discussed in Sections 3.1.4 and 3.1.5.

c) Testing phase – The next phase is to perform several different types of testing as per the testing plan developed in the architecture design step. Testing of the Feature Modeler was done by the author as well as other members of the testing staff at Visual Learning Systems. Details are further discussed in Sections 3.1.4 and 3.1.5.

d) Release – The final phase involves the development of an install/distribution package for the software. The Feature Modeler was developed as an add-on to the Feature Analyst tool, and thus distribution was integrated into existing distribution methods for Feature Analyst. A brief summary of the release details is treated in Section 3.1.4.

Using the staged implementation model for the development of the Feature Modeler was simple and effective. The well-defined and unchanging nature of the requirements made for a solid starting point, and the multiple stages of implementation allowed enough flexibility to address concerns and issues that required modifications to the original design. Chapter 3 delves into the details of the staged implementation process that was used in the development of the Feature Modeler, and the results that were achieved.

CHAPTER 3: IMPLEMENTATION AND RESULTS

3.1 Development Process

This section describes in detail the steps in the staged implementation software development process with regard to the development of the Feature Modeler.

Note that although the author was in charge of Feature Modeler's development, not all aspects of the development were directly carried out by the author. The author did not write the funding proposal or the first draft of the design document. Also, the original module architecture as represented in Figure 3 was already a part of Feature Analyst, and only modified during this thesis work. Unless otherwise stated, all other work was directly performed by the author.

3.1.1 Concept

The concept for a new AFE workflow visualization technique arose from a combination of sources: originally identified as an area for improvement in the existing Feature Analyst software by VLS staff and management, then affirmed and solidified by Feature Analyst users. The concept was eventually given the impetus to proceed to actual design and implementation in early 2005, when funding for the project was secured from the U.S. Army's ERDC-TEC division.

3.1.2 Requirements Development

This phase in the development process is usually critical to the project's success, and the Feature Modeler was no exception. Significant time was spent enumerating the specific goals for the Feature Modeler, and how best to apply relevant visualization techniques to this particular problem. This work resulted in the following goals.

1) **Viewing of existing AFE models**: first and foremost, the Feature Modeler must clearly and cleanly display AFE workflow models of varying sizes, including the ability to zoom in and out and pan across models which are larger than a single screen. Color-coding of individual modules to convey information about the module's state was identified as a necessity. An auto-layout function was also identified as an important visualization aid.

2) **Interactive execution of the workflow with user-defined inputs**: the Feature Modeler's main functionality would be its ability to assist a user in production-style processing of imagery given an existing AFE workflow model. This processing would occur in two possible modes: recipe mode and silent mode.

    - Recipe mode - the user sets up the initial inputs, then steps through each module in the workflow individually, stopping after each step to allow user interaction and verification of results before continuing to the next step. This mode leverages the knowledge and workflow from the existing model, while still maintaining the flexibility needed to account for the inherent variations from one extraction to the next.

- Silent mode - users set up the initial inputs for the model, and then silently[1] run the entire model from start to finish. Silent mode allows for the greatest gains in speed and efficiency. However, this mode does not allow much flexibility in accounting for variations from one image to the next.

3) **Creation and editing of AFE models**: an essential part of AFE workflow modeling is the ability to create and modify meaningful AFE models and to store associated metadata. The ability to accomplish these tasks was an important aspect of the Feature Modeler's functionality. This goal would be accomplished by using current best-practices from the field of visual programming, including the notions of individual functions or "modules", variables as input to the modules, and a defined workflow of data from one module to the next.

4) **Tutorial integration into AFE workflows**: this functionality would mimic recipe mode, but would focus on a multimedia tutorial approach to assist users in stepping through an existing AFE model, "recipe-style", with instructions at each step. This mode was intended for use by junior analysts who could follow an AFE workflow developed by a senior analyst.

5) **Presentation of an easy-to-use interface with the outside world**: accomplishing this goal would ensure that future AFE modules can be easily integrated into the suite of modules available to developers of AFE workflow models. This goal requires forward-looking design to maximize the life-expectancy of the Feature Modeler and make it a useful and relevant tool in the AFE field for years to come.

[1] i.e., with no user interaction during intermediate steps

Several other goals were also specified, such as the ability to import, export and print models, and the ability to analyze and extract rules from the models. However, the five aforementioned goals are the most relevant to the objectives of this thesis.

In addition to the goals outlined above, the Feature Modeler was also subject to certain design constraints. Firstly, the Feature Modeler had to fit into the existing AFE model architecture, as implemented in Feature Analyst version 4.0. These models were already being generated by commercial users of Feature Analyst, and it was these models which would be displayed and manipulated in the Feature Modeler.

Related to the previous constraint, the Feature Modeler would need to be implemented as an add-on to Feature Analyst, and developed using the current .NET architecture. This constraint turned out to be more of an advantage than a constraint, since both the Feature Analyst API and the .NET architecture are well-designed and powerful aids for developing state-of-the-art software.

3.1.3 Architectural Design

This phase in the development process addressed two important issues: the actual architecture of the Feature Modeler, and the architecture of the AFE models which were to be displayed and manipulated by the Feature Modeler. The second issue was the more important design issue, since the models were to be the main focus of the Feature Modeler's functionality. In addition, any changes made to the general model architecture needed to include backwards compatibility with current AFE models already being developed by Feature Analyst users.

Design of the AFE model architecture addressed four questions:

a) How will individual modules be implemented?

b) How will module inputs and outputs be handled?

c) How will modules interact with one another?

d) What control structures will be allowed in the workflow?

Question (a) found a good starting point in existing AFE models. Each AFE module[2] was implemented using a process-view-controller (PVC) design pattern (see Figure 3). The PVC design pattern is flexible enough to allow a wide variety of implementations which can all be accessed and manipulated in the same way. Although all three parts are required for each module, only the process is actually stored on disk in the AFE file, with its associated parameters. This requires that the process have the ability to dynamically create its own controller, resulting in tight coupling of the process and controller. However, it simplifies storage of the AFE workflows, and has the added benefit of confining backwards-compatibility requirements to only one part of the module.

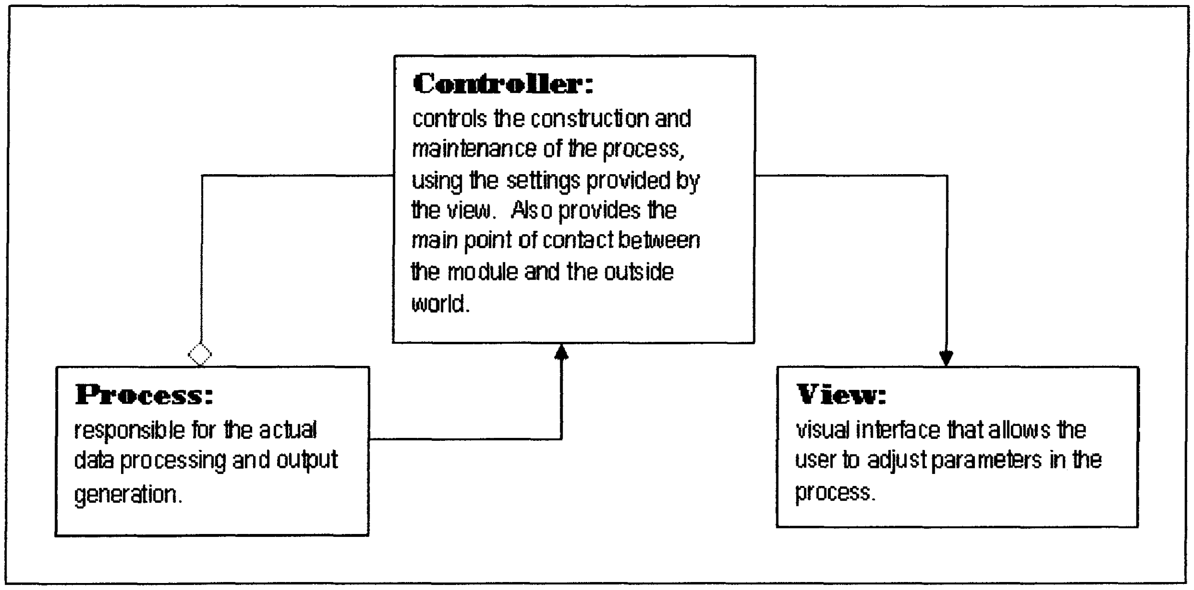

Controller:
controls the construction and maintenance of the process, using the settings provided by the view. Also provides the main point of contact between the module and the outside world.
Process:
responsible for the actual data processing and output generation.
View:
visual interface that allows the user to adjust parameters in the process.

Figure 3: Process/View/Controller architecture for a single AFE module.

[2] Examples include a training/classification module, a convert-to-line module, etc.

Answering Question (b) also involved reliance on existing model design. The inputs and outputs of these modules are shapefiles or images, so it was deemed sufficient for each module to include a mechanism which allowed the Feature Modeler to access its parameters. In theory, the Feature Modeler could simply query the module for all parameters which contained shapefiles or images, and treat these as the inputs and outputs of the module. In practice, however (see Section 3.1.4) this approach was overly simplistic and not flexible enough for many modules; in response, the module design was modified to include an interface for inputs and outputs. In the final design, each module controller was made responsible for providing input views (and output views, although output views are as yet unimplemented at the completion of this paper), which are represented as input and output nodes in the Feature Modeler, and presented to the user for input and output setup. The user's choices are then returned to the controller by the Feature Modeler, allowing the controller to maintain complete control over the type, number, and setup details for each input and output. In addition, these input and output views are designed to be comparable. This helped answer Question (c) regarding module interaction by providing the Feature Modeler with a mechanism for deciding whether the output of one module is a suitable input for another module. Question (d) was originally answered by permitting both linear and cyclic (loop structures) workflow paths. Cyclic paths were to include decision points, which an analyst could use to decide whether to continue on or repeat the loop. As Sections 3.1.4 and 3.1.5 indicate, however, the inclusion of cycles and decision points was not implemented during the first or second implementation iterations, and remains on the list for a future iteration.

The attention paid to the design of the AFE workflow models has resulted in a robust and extendible framework for future development of new AFE modules which will fit seamlessly into the existing Feature Modeler environment.

Regarding the design of the Feature Modeler itself, it is designed as a plug-in to the existing Feature Analyst software. As such, it has access to all the standard functionality of Feature Analyst, such as image and shapefile handling and cross-platform compatibility. It is designed to display as a modeless window which is started from within a Feature Analyst session. Module outputs are to be passed back to the Feature Analyst framework for display. Being modeless, Feature Modeler allows users to view and interact with any intermediate outputs generated by running an individual module, before returning to Feature Modeler to continue processing further steps in the model.

Also during the architecture design phase, considerable time was spent developing a meaningful symbology for the display of AFE models in Feature Modeler. Shape and color were both utilized as effective methods of communicating type and state for each item in the display. Details and examples of the symbology are discussed in Section 3.2 below.

3.1.4 Stage 1: Detailed Design, Construction and Release

The first implementation of the Feature Modeler started with an investigation into suitable software packages to form the basis of the Feature Modeler interface. Several good software packages already exist for developing graph, chart, and workflow visualization software, so there was no need to start from scratch and reinvent the wheel. Two software packages were considered in detail, and the final choice was a diagramming package based on research into the development of a visual programming language. This fit nicely with the goal of developing Feature Modeler as a new way to visualize feature extraction "programs", complete with inputs and output variables, processing modules, and basic program control structures. The chosen diagramming package was also implemented in .NET, which allowed for convenient integration into the existing Feature Analyst framework.

Once the groundwork was in place, the first implementation of the Feature Modeler could begin. Details of the actual implementation are of little relevance to the scope of this write-up, even though pages could be filled with anecdotes about furious typing, compiler errors, devious little hidden bugs, and so on. Suffice it to say that the first implementation of the Feature Modeler finished on schedule, within budget, and with the complete implementation of Goals 1 and 2 (see Section 3.1.2). Goal number 4 was also partly achieved, in that a framework was put in place which allowed the storage and display of metadata for individual modules and inputs. This framework only supported text-based metadata, but was implemented in such a way that multi-media descriptions and recipe instructions could easily be added in during a future iteration stage.

Unit testing of this first iteration was accomplished during initial implementation by the developer, and then supplemented by integration testing after the completion of development coding. Integration testing was done by VLS testing staff as well as the developer. Of particular concern during the testing phase was the integration of existing AFE modules into the Feature Modeler's framework. Many of the modules worked perfectly well in the Feature Modeler, but some did not, highlighting the weakness of the design for input/output handling (see Section 3.1.3). As a result, the initial process-view-controller design for each module was modified to include a collection of element views which represent inputs and outputs to the module, and are handled by the controller (see Figure 4). This change (at least for inputs) was implemented during the second implementation stage, as detailed in Section 3.1.5 below.

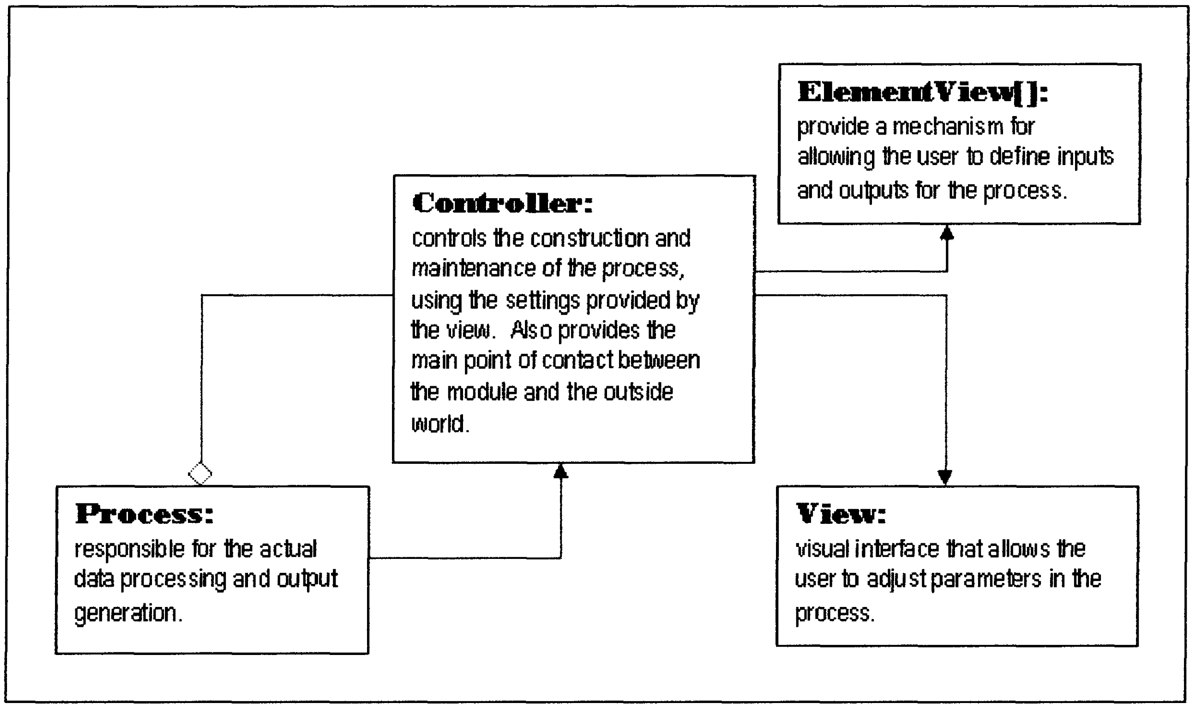

ElementView[]:
provide a mechanism for allowing the user to define inputs and outputs for the process.
Controller:
controls the construction and maintenance of the process, using the settings provided by the view. Also provides the main point of contact between the module and the outside world.
Process:
responsible for the actual data processing and output generation.
View:
visual interface that allows the user to adjust parameters in the process.

Figure 4: Modified module architecture, including a more robust mechanism for handling module inputs and outputs.

The first release of Feature Modeler was in July of 2005, when Feature Modeler was delivered to Army ERDTEC via an installation CD for Feature Analyst 4.0 which included the Feature Modeler add-on. This marked the completion of the stage 1 implementation phase of Feature Modeler.

3.1.5 Stage 2: Detailed Design, Construction and Release

The second implementation stage began with re-visiting the individual module architecture. As a result, the ElementView interface was created, which allows a standard interface for setting inputs and outputs. A few other small changes were made to the module architecture as well, with the intention of putting as much responsibility as possible into the hands of the controller object, and standardizing the behavior of each class in the module. The expectations for each class's behavior were documented for the benefit of future module developers.

Once the module changes were finalized, much of the remaining time was spent testing the Feature Modeler with existing modules to ensure that the basic design was robust, flexible, and maintainable. During this time, some existing modules were updated to ensure appropriate behavior or to fix existing bugs.

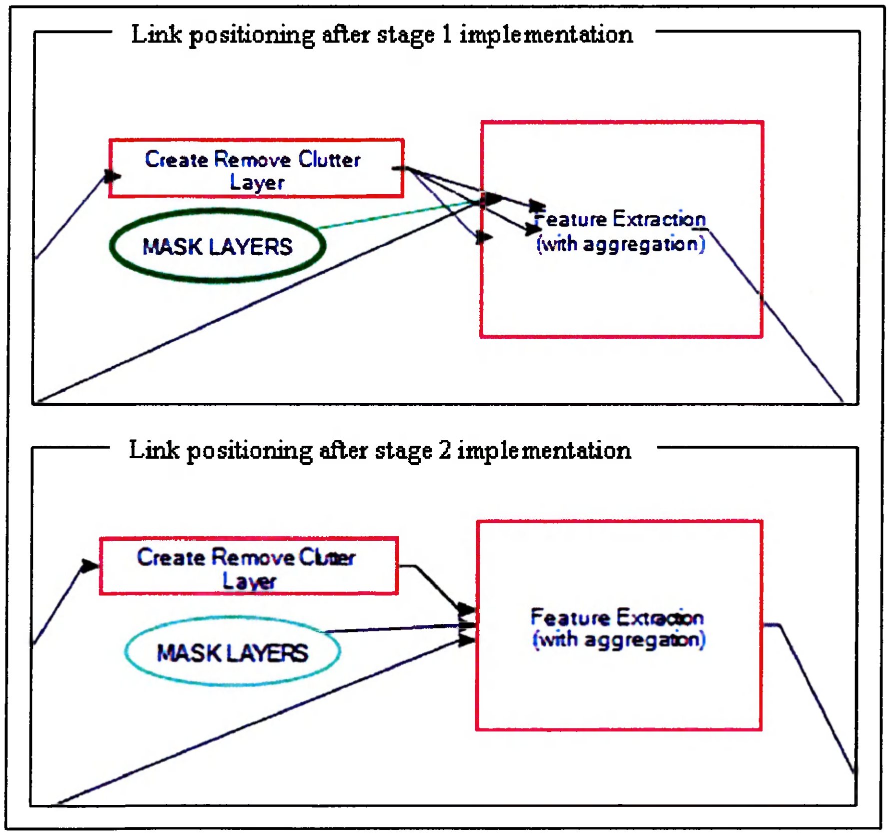

Link positioning after stage 1 implementation
Create Remove Clutter Layer
MASK LAYERS
Feature Extraction (with aggregation)
Link positioning after stage 2 implementation
Create Remove Clutter Layer
MASK LAYERS
Feature Extraction (with aggregation)

Figure 5: Comparison of link display before and after re-working.

Also during this implementation phase, improvements were made to the interface, via changes to the display and symbology. An extra color was added to module outlines to help represent current module states; the inter-modular links were redrawn, resulting in a cleaner and less cluttered model (see Figure 5); and final symbology colors were updated with the help of VLS's graphic designer.

These and many other small changes resulted in a high-quality, functional implementation of a new and effective AFE modeling and visualization technique, which is slated for commercial release as part of the upcoming release of Feature Analyst 4.1.

3.2 The Feature Modeler

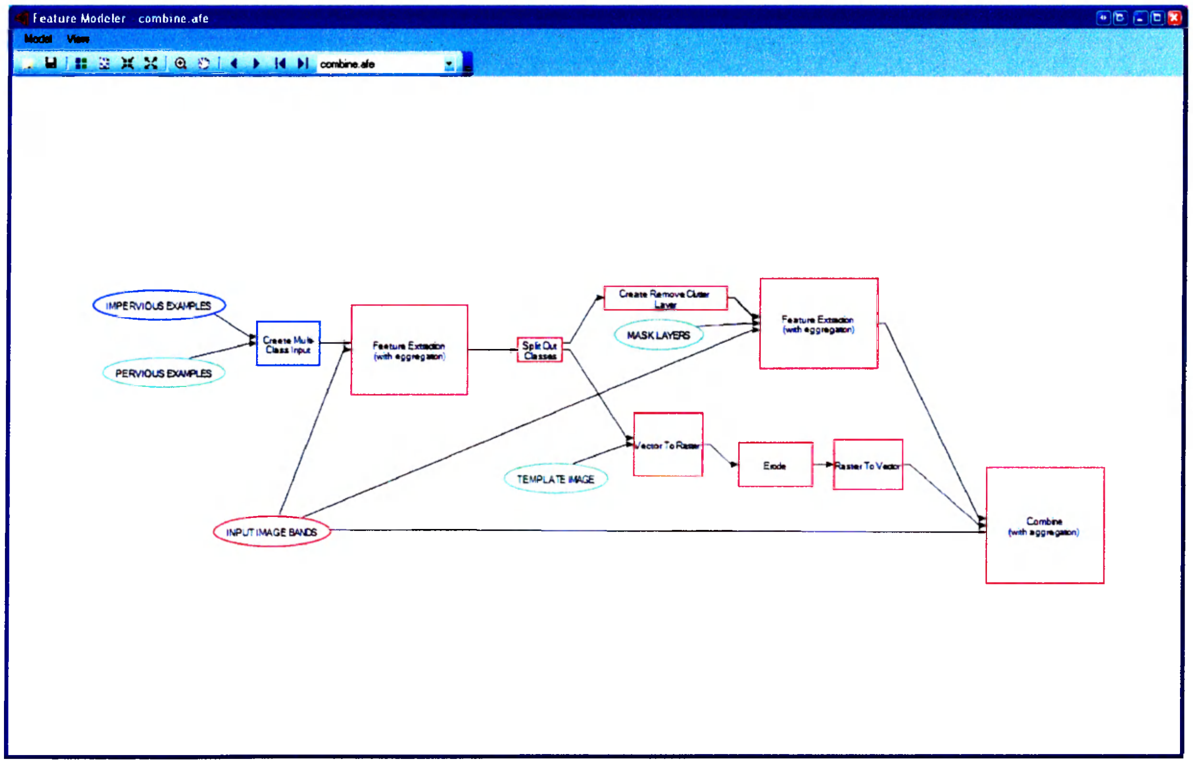


Figure 6: Visualization of a typical AFE workflow in Feature Modeler.

Figure 6 shows a typical AFE workflow as visualized in the Feature Modeler. The model in Figure 6 was developed for the automated extraction of pervious and impervious surfaces[3] from urban satellite imagery. It represents a typical AFE workflow model, and will be analyzed in detail during the following sections which describe AFE workflow visualization using the Feature Modeler.

3.2.1 Model Symbology

Webster's dictionary defines symbology as "representation or expression by means of symbols". Thus, any good symbology set should be, first and foremost, an effective means of representing or expressing information. In the case of AFE workflow visualization, the symbology was required to convey two separate sets of information to the user: what type each piece of the model is, and what state that piece is in. In the Feature Modeler, shape is used to convey type, and color represents state.

Every AFE model is made up of three main components. First, each model consists of one or more AFE modules. These various modules, although diverse in their particular implementations, all perform some kind of automated task related to GIS. They are the meat of any AFE model, and thus need a clear representational symbology. Figure 7 shows how these various

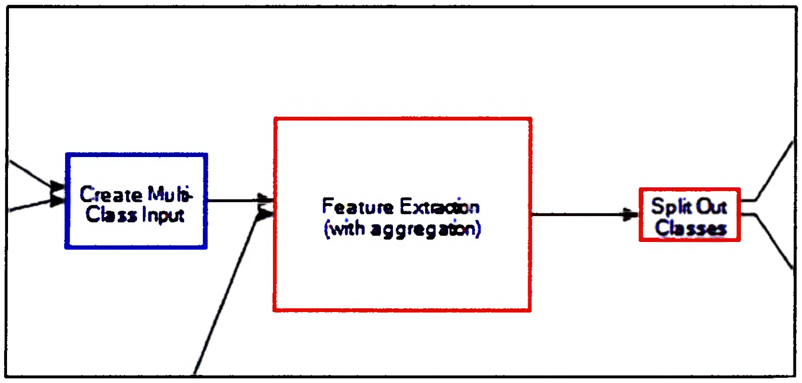


Figure 7: AFE Modules represented as colored rectangles.

[3] Pervious surfaces are those that absorb rain water, such as grass, trees, and flower beds. Impervious surfaces - like roofs, sidewalks, and roads - do not absorb water. The ratio of pervious to impervious surface is important in the planning of urban storm-water drain systems.

modules are represented as colored rectangles containing descriptive titles. The colors denote the state of the module, and are discussed in detail below.

The second major componcnt of any AFE model is user-defined inputs. Modules all require inputs. Sometimes these inputs are supplied from the output of a previous module, or inputs may be specified by the user. These user-defined input nodes are represented as colored ovals with descriptive names inside. As Figure 8 indicates, these input nodes may be displayed in various colors, to indicate their current state.

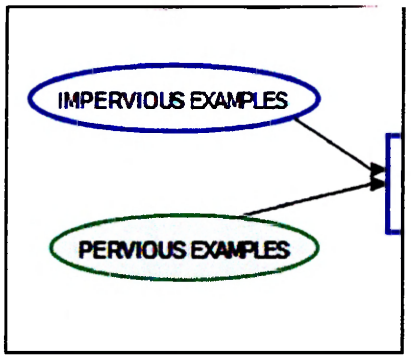


Figure 8: Input nodes.

The third major component inherent in any AFE model is the information or data flow. This data flow naturally lends itself to visualization as directional links, which are displayed in Feature Modeler as black links with arrows to represent the direction of data flow.

As mentioned previously, AFE workflow visualization involves not only displaying the components of an AFE model, but also conveying information to the user about the state of each input node or module. If these workflows are considered in visual programming terms, then it is clear that both input nodes and modules can be in various states. Color, both outline and interior fill, is used to represent these various states.

In the case of input nodes, an input can be either set or not set. Again, in visual programming terms, input nodes can be thought of as variables that either have a value assigned to them, or are equal to null. This state is represented by the symbology as grey fill or clear. In Figure 8, the bottom input node has been set, while the top input is unset.

In addition to being set or unset, an input node may also be required, optional, or irrelevant (see Section 3.2.3 for an explanation of the meaning of these three distinctions). This threefold distinction is represented in the symbology by three different outline colors: red, green, and blue, respectively. Examples of input nodes in all three states can be seen in Figure 6.

AFE modules can also be in several different states, and like input nodes, these states are represented by color. Like input nodes, modules can be filled with grey or have a clear background. However, rather than denoting whether the module has been set or not (as in input nodes), a module with grey fill represents a module that has been processed. That is, its outputs are valid and ready to be used by further modules or displayed to the user. In a sense, this is very similar to an input nodes "set" state, since both have the same effect on modules further along the workflow.

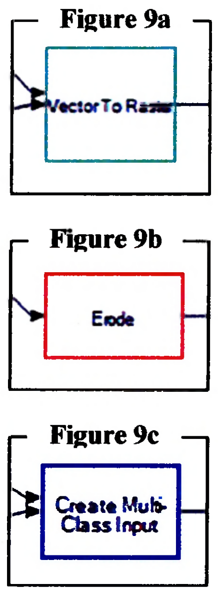


Figure 9: Modules in varying states of readiness.

The effect that modules and input nodes have on their following modules is evidenced by the outline color of the following module. For modules that are unprocessed, the outline color denotes whether they are ready to run (i.e. have all required inputs), do not have the needed inputs, or are irrelevant. A module that has access to all its required inputs will be outlined in green, as in Figure 9a. Figure 9b, c show modules that are lacking inputs or are irrelevant (unneeded by further modules), outlined in red and blue respectively. Section 3.2.3, dealing with silent mode execution of the workflow models, explains in more detail the meaning of each of the three states.

All these colors, shapes, and lines together form a comprehensive symbology for displaying AFE workflow models. Taken as a whole, the symbology gives a user quick and easy access to relevant information about the model's structure and state. The usefulness of this information is discussed further in Sections 3.2.3 and 3.2.4.

3.2.2 Visualization Aids

In addition to displaying a static model, Feature Modeler also includes several tools and menu options to aid in convenient model visualization. AFE workflows can easily fill more than a single screen, so the standard navigation tools are present, including zoom in, zoom out, zoom to full extent, pan, and zoom to selection. In addition, three different options are available for displaying links. These are standard (as in Figure 6), orthogonal, and Bezier. These three different display methods for links can be used to clear up screen clutter arising from the organization and placement of modules and input nodes on various different models.

Perhaps the most useful visualization aid, though, is the AutoLayout tool. The AutoLayout tool performs an automatic directed sort on the model and attempts to arrange its components in an ordered and sensible way on the screen. Figure 10 shows a model before and after AutoLayout, portraying the usefulness of this tool for effective AFE workflow visualization.

Feature Modeler also includes functionality for AFE model I/O, including importing, opening, saving and printing models, but since these functions do not directly pertain to AFE visualization and modeling, a detailed explanation has been omitted. The next two sections, dealing with different methods of using these AFE models, are the real keys to the success.

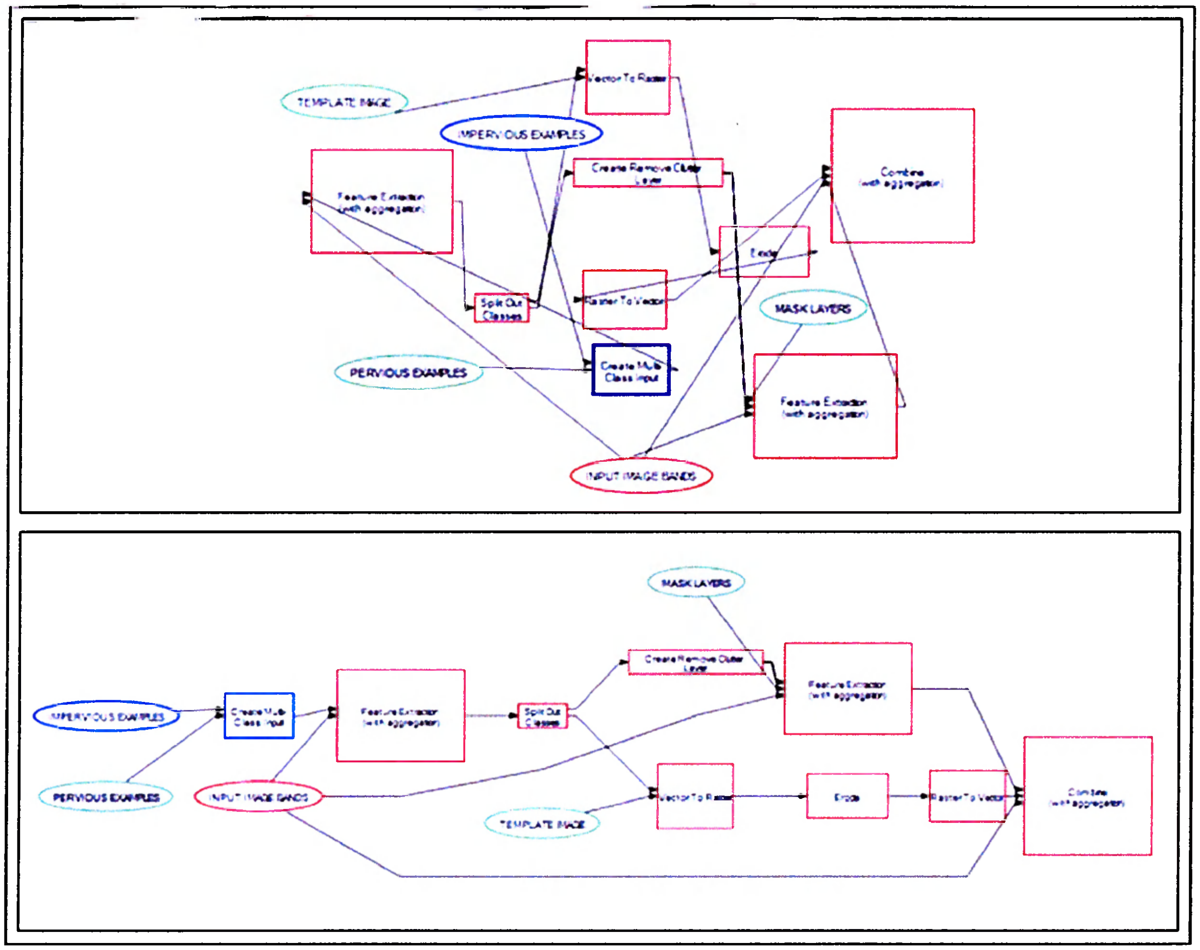


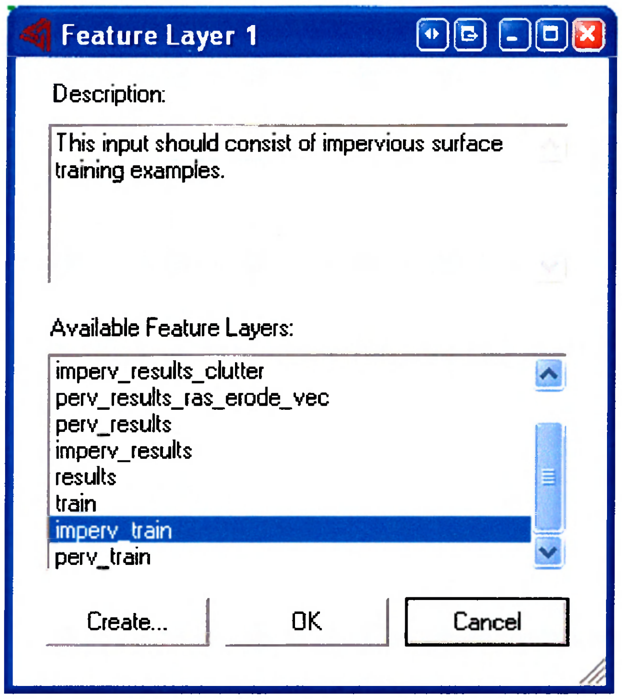
Figure 10: An AFE model, before and after AutoLayout.

3.2.3 Silent Mode

Visualization of AFE workflows is one thing, but the visualization is of limited usefulness unless it includes a mechanism for applying the workflow to new imagery. Silent mode provides this mechanism, and transforms the AFE model from a static display to an interactive visual program. Silent mode allows a user to apply all the settings and parameters from a previous AFE workflow to new imagery. It is the quickest and easiest way to apply an AFE model to new imagery, since it is designed to require no interaction from the user other than setting up the initial inputs.

In order to run an AFE model in silent mode, the model must first be opened in Feature Modeler. If the model was created on the fly by Feature Analyst, then its modules will all be in a processed state (represented by grey fill, see Section 3.2.1). In order to run a model, it must first be reset for silent mode. Pressing the silent mode reset button on the toolbar accomplishes this. When a model is reset for silent mode, only certain parts of the model are reset. That is, some internal settings (such as trained learners) are retained, allowing these settings to be applied to new imagery. So for example, if the model was developed to extract roads from IKONOS satellite imagery, then the model will "remember" what it learned from the last time it was run, and so will not need to be re-trained with new examples of roads each time it is run. Thus, when the AFE model is reset for silent mode, some modules may not be reset, or they may become irrelevant (denoted by blue outlines) because they are only used to train the model. In short, resetting an AFE model for silent mode does a smart reset, resetting only the parts of the model necessary to process new imagery, while retaining all the settings it needs to process new imagery with no user input during the process.

Once an image is reset, all required inputs need to be reset. Each input node displays a context menu when right-clicked, which allows (among other things) the user to define the value of the input. Choosing this option brings up an input dialog. Figure 11 is an example input dialog for a training example input. When the required inputs have been defined, all modules in the AFE model should have white backgrounds with green or blue outlines. This confirms to the user that the model is ready to run in silent



Description:

This input should consist of impervious surface training examples.

Available Feature Layers:

imperv_results_clutter
perv_results_ras_erode_vec
perv_results
imperv_results
results
train
imperv_train
perv_train

Figure 11: An example input dialog.

mode. (If one or more modules have a red outline, then these modules are missing some required input.) Once the inputs have been set up, clicking on the "Run in silent mode" toolbar button begins execution of the AFE workflow model. From this point, no user input is required. This may seem unimportant, but many automated feature extraction algorithms are computation-ally expensive (not to mention that the shear amount of data in multi-gigabyte imagery takes time to analyze), so a complete AFE workflow executed on a large image can take hours to process. Thus, hands-free image processing is an important aspect of efficient feature extraction.

Silent mode offers GIS analysts an effective tool for addressing efficiency bottlenecks in feature extraction. It requires only one user-aided iteration (perhaps on a small image, or a small section of a large image) to develop an effective model which can then be applied to large amounts of similar data with no user input required. It is fast, and on the right imagery sets it offers extraction speeds that are orders of magnitude better than manual extraction. However, silent mode allows an analyst only very limited ability to change the actual performance of the model from one run to the next (some parameters such as minimum shape size can be changed, but no re-training is possible in silent mode). As such, it has limited usefulness in applications where the desired results are the same, but the input imagery is variable. These situations may arise when, for instance, data collection is inconsistent (i.e., some imagery was collected on cloudy days, othcrs in bright sunlight). In these cases, silent mode may produce inaccurate results. This can be solved by running a model in recipe mode, which allows users the flexibility to adjust for variable imagery without having to regenerate the entire model.

3.2.4 Recipe Mode

In a perfect world, GIS analysts would only ever need to develop and train an AFE model once to extract residential houses in American suburbia, and ever after, use silent mode with the original AFE model to extract residential houses from additional imagery. Unfortunately, the world isn't perfect, and geospatial imagery sets are subject to internal variation. Weather conditions, sensor type, and time of day are only a few of the factors that affect data collection. So the spectral signature of a suburban residence in Seattle may be quite different than that of a similar house in Auckland, New Zealand. In both cases, though, the same workflow would be applied to extract the houses from the imagery; only the actual pixel values of the imagery would be different. In this situation, the most efficient method for extracting the houses would be one that took advantage of the workflow and basic parameters used in one extraction, but retrained the model with new examples of houses in the second set of imagery. This is exactly the flexibility that recipe mode offers.

As in silent mode, a model must be open in Feature Modeler to run in recipe mode. Also like silent mode, the model must be reset before running it on new imagery. Resetting a model for recipe mode completely resets every part of the model, so that training information about any previous images is cleared. The user must also redefine the values of all required inputs to prepare the model for running. This often involves not just browsing to a new image, but also drawing training examples of the desired feature, so the AFE modules can re-learn what they need to look for.

Once the model is set up and ready to run, the user can click the "Step forward in recipe mode" button on the toolbar. This runs the first module in the AFE model, and displays the results to the user. The user then has a chance to analyze the results, and perhaps manually tweak them before continuing with the next step in the model. Or, the results may require user interaction before the next step can run. For example, if the next module is a Remove Clutter module, then the user must provide examples of correct and incorrect shapes, so the module can re-learn what clutter to remove. In this way, the user maintains control over the results at each step in the model, while still benefiting from the speed and efficiency gains inherent in using an existing workflow. This method of feature extraction, although slower than silent mode, is still faster than performing the same extraction without the aid of an existing AFE model. Since the results will be identical to redoing the extraction from scratch, recipe mode is a definite improvement.

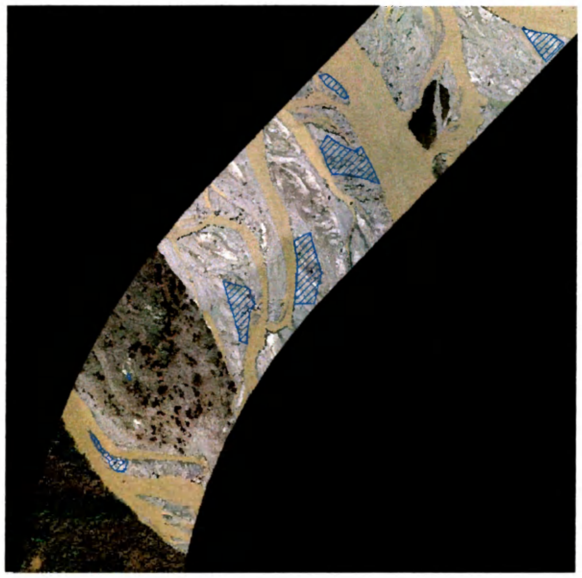
3.3 Example Results

It's easy to imagine that modeling and reuse of existing AFE workflows, whether in silent mode or recipe mode, can lead to time savings, but what about accuracy? Do the time savings come at the expense of result quality? The answer is an unequivocal no. In fact, reuse of AFE models can actually increase the accuracy of the final product, as well as decrease the time required. It's well known that manual digitizing of features by human analysts is prone to several errors, not the least of which is inconsistency. So if two analysts are each given a similar image, and asked to differentiate between pervious and impervious surfaces, one may err on the side of pervious, and the other on the side of impervious, leading to inconsistent results on almost identical imagery. Even a single analyst's work can vary from one image to the next. An existing, trained AFE model will not have these inherent inconsistencies, but will produce consistent results from one image to the next. This is especially true in silent mode, but even recipe mode could potentially produce more consistent results than redoing each extraction from scratch. As long as the original workflow was developed by an expert analyst, consistent high-quality results could even be produced by analysts with only basic knowledge of automated feature extraction. This further lowers the overall cost of feature extraction.

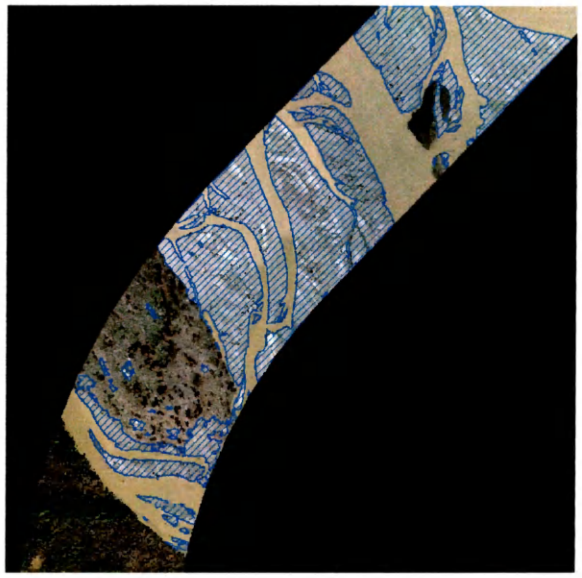
The following figures show example results obtained from silent and recipe mode runs on various imagery. Figure 12 shows the results of running a gravel bar AFE model in silent mode on new imagery that is very similar to the original. The results illustrate the usefulness of silent mode. Figure 12a shows the original image and training examples. The gravel bar extraction results in Figure 12b took approximately 20 minutes to achieve, including the time it took to draw the training examples. This is already an order of magnitude faster than hand-drawing the gravel bars, and the results speak for themselves (see Figure 12b). But even better, consider the new image in Figure 12c. This image was processed using silent mode, and the gravel bars in Figure 12d were extracted in less than two minutes. That's another order of magnitude faster than the original extraction.

Despite these promising statistics, silent mode has its limitations. Figure 13 points out the deficiency of silent mode when running on disparate image sets, and how this problem can be corrected by using recipe mode to retrain the model for new imagery. The results in Figure 13 were generated using the pervious/impervious model shown in Figure 6. Figure 13 shows the original imagery on which the model was developed. The final results of this extraction are shown in Figure 13b. Figure 13c is a new image which, although still depicting suburban houses, is quite different in terms of resolution, zoom level, and so forth. Figure 13d shows that using silent mode (with its knowledge learned from the original image) is ineffective in extracting pervious/ impervious surfaces from the new image. In this case, recipe mode is a more appropriate solution. Recipe mode allows the flexibility to re-train the AFE model for new imagery, while still retaining the information present in the workflow and associated settings. In other words, even though the images in Figure 13a and 13b are quite different, the extraction of pervious/impervious surfaces requires the same basic workflow in both cases. The only new information required to run recipe mode is some new training examples for the new image. Figure 13e shows these new training examples (displayed in grey and green), and Figure 13f displays the results of running the AFE model in recipe mode on the new image. For time comparison, the original extraction (Figure 13b) took about an hour. Using recipe mode, results were extracted from the new image in approximately ten minutes. Even allowing for the trial and error involved in the first extraction, recipe mode still resulted in a substantial time saving for the second extraction. Even better, the second extraction could easily be performed by a junior analyst with little or no experience in automated feature extraction.

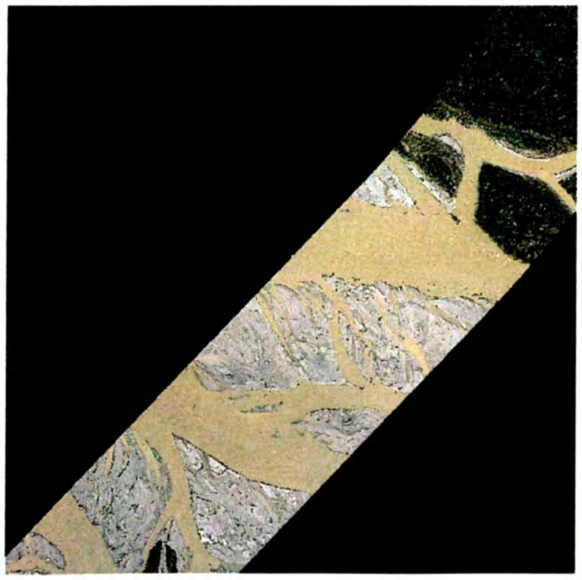
Although anecdotal, the two sets of example results described above lend support to the thesis that modeling AFE workflows and applying them to new imagery can be an effective means of increasing efficiency and lowering the cost of extracting features from geospatial imagery.



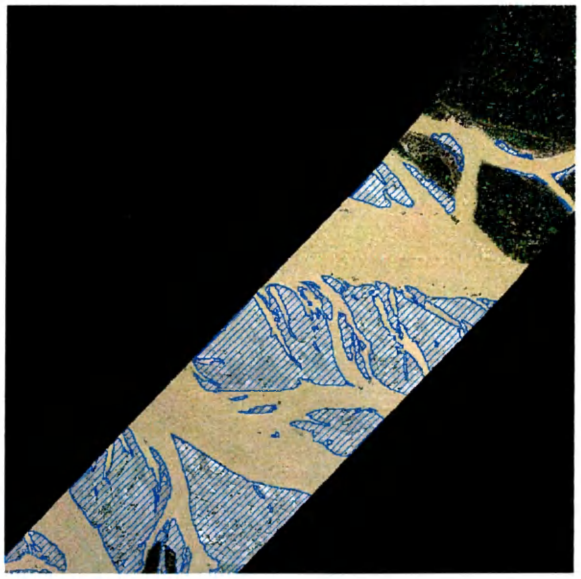
a) original imagery, with hand-digitized gravel bar training examples in blue



b) original gravel bar extraction results (in blue cross-hatched overlay)

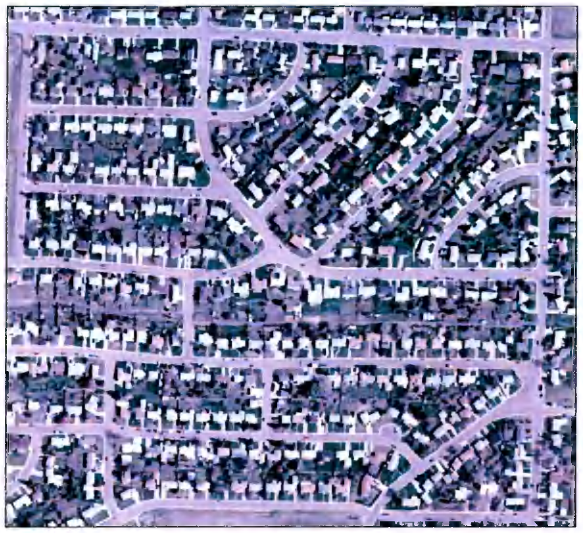


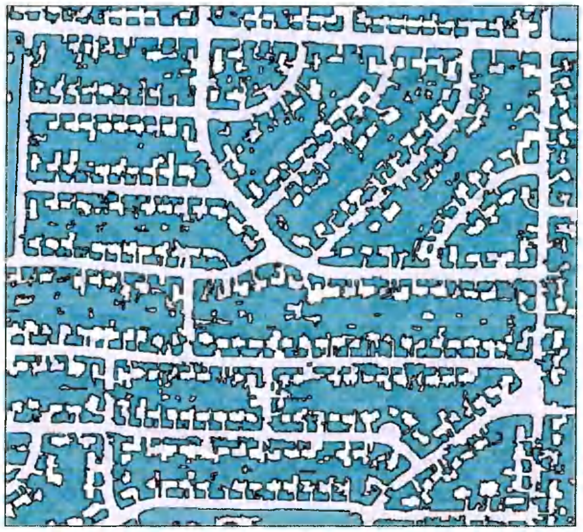
c) new imagery (no new training examples required)

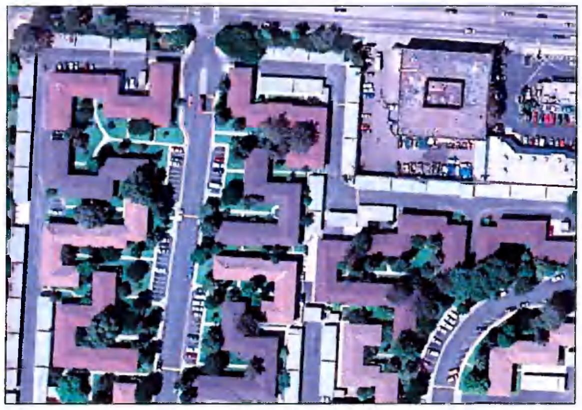


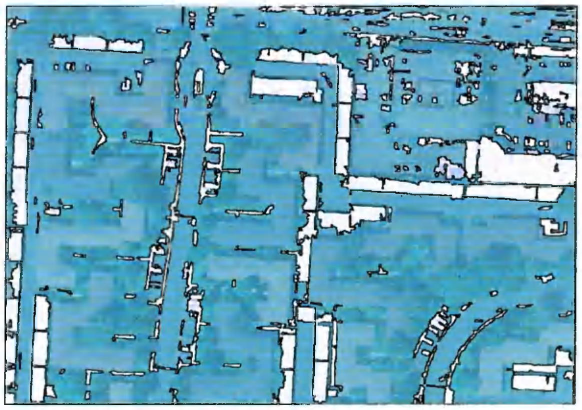
d) results of running original AFE model in silent mode on new imagery

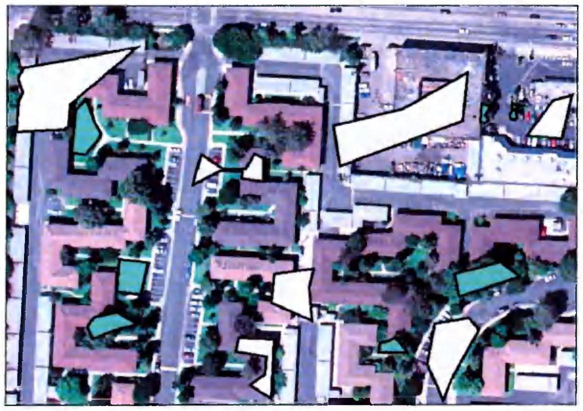
Figure 12: Effectiveness of silent mode on feature extraction from similar imagery.


a) original imagery


b) original results


c) new imagery


d) results of silent mode on new imagery


e) new training examples for recipe mode

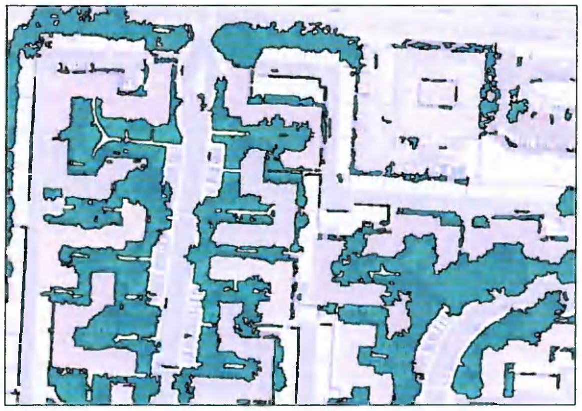

f) results of recipe mode on new imagery

Figure 13: Comparison of silent and recipe modes on dissimilar imagery.

CHAPTER 4: CONCLUSIONS AND FUTURE WORK

4.1 Critique of the Feature Modeler

Section 3.3 presents subjective evidence that the Feature Modeler successfully implements a new technique which can improve the speed and efficiency of automated feature extraction from color imagery. Rigorous testing and analysis will be required (see Section 4.2) to quantify the actual gains GIS analysts can expect from using Feature Modeler, but the preliminary results are encouraging. In particular, Feature Modeler presents the following advantages over current methods:

a) Applying AFE models to new imagery, whether in silent mode or recipe mode, is faster than re-doing the feature extraction on each new image from scratch.
b) Using recipe or silent mode on new imagery produces more consistent results than processing each new image individually.
c) Feature Modeler can be used as an effective means of leveraging the knowledge from expert GIS analysts and imparting that knowledge (in the form of AFE models) to less experienced analysts.

However, Feature Modeler is not without its weaknesses. No doubt several will be discovered once the Feature Modeler is commercially released, but the known weaknesses are listed here.

a) The process-view-controller module architecture, although flexible and powerful, certainly provides enough rope for module developers to hang themselves with. That is to say, thc module design errs on the side of flexibility and forward-compatibility, not robustness. This should not be a problem for any new modules developed by VLS software engineers, but it may be for third-party developers who design new modules[1]. Safeguards have been built into the Feature Modeler to handle any such rogue modules, but this issue may still need to be re-addressed in the future.

b) Design and development of the Feature Modeler focused extensively on making use of existing AFE models, and very little time was spent on developing ways for users to build new AFE models in Feature Modeler. As a result, goal number 3, defined in Section 3.1.2, was not achieved at all. Currently, the only way to build an AFE model is to step through an actual feature extraction on an image. Feature Analyst automatically records each step in an extraction and appends it to previous steps, forming a comprehensive AFE workflow history which can then be used in Feature Modeler. This is generally the best way to develop AFE models anyway, since the results can be verified at each step in the process. However, AFE experts may occasionally want to quickly build a workflow model without going through the actual extraction. Currently, there is no mechanism in place for doing this.

c) Input nodes provide a robust and flexible way for the user to define module inputs. However, module outputs are still handled implicitly by the Feature

[1] The process-view-controller architecture is accessible to third party developers via the Feature Analyst Plugin API.

Modeler, limiting the number and type of module outputs. Module outputs should be handled in much the same way as inputs, allowing more flexibility in module dcvelopment.

The above weaknesses are not show-stoppers, but they do pinpoint areas for future improvement and refinement. Note also that they are all issues resulting from the *implementation* of the new modeling technique; they are not weaknesses in the actual technique (modeling AFE workflows) itself. As such, they should be easier to address than flaws in the underlying method.

On the whole, it can be stated that the implementation of a new method for modeling and visualization of automated feature extraction workflows was successful, and offers the GIS world another useful tool in the quest to provide meaningful geospatial information in an efficient and cost-effective manner.

4.2 Future Work

Although the work on this thesis has advanced the cause of GIS, much work remains to be done. Some ideas related to this thesis have been listed here for the benefit of readers who may be interested in furthering the cause of efficient automated feature extraction.

a) The Feature Modeler needs at least one more implementation cycle which addresses some of the weaknesses discussed above.

b) Specific testing needs to be done to quantify the efficiency gains achieved by the Feature Modeler. Automated feature extraction in general has already been shown to be more efficient than hand-digitizing [10, 12], but how much faster is it to use Feature Modeler than to treat each image as a separate extraction? This and related questions deserve an answer.

c) A study which examines the usefulness of Feature Modeler as a teaching tool (via recipe mode with embedded tutorials) would also be useful. Given the novelty of automated feature extraction in general, such a study could provide useful information for entities seeking to embrace AFE technology.

BIBLIOGRAPHY

[1] Albaredes, G., A new approach: user oriented GIS. In *Proceedings of EGIS '92*, 1992, pp. 830-837.

[2] Budic, Z., Effectiveness of geographic information systems in local planning. *Journal of the American Planning Association* 60.2 (1994): 244.

[3] Elroi, D. Geek-speak: Model Builder vs. Python vs. AML? *GIS Software Techniques and Implementation* 3.3 (2003): 1-2.

[4] Esch T., Roth A., Dech S. Robust approach towards an automated detection of built-up areas from high resolution radar imagery. In *Proceedings of the ISPRS Joint Symposium URBAN05 and URS05*, 2005.

[5] Henry, J. *Software Project Management.* Boston, MA: Pearson, 2004.

[6] Keukelaar, J. Topics in soft computing. PhD thesis. Royal Institute of Technology, Stockholm, 2002.

[7] Koelma D., Smeulders A. A visual programming interface for an image processing environment. *Pattern Recognition Letters* 15.11 (1994): 1099-1109.

[8] Konstantinides, K., Rasure, J.R. The Khoros software development environment for image and signal processing. *IEEE Trans. Image Proc.* 3.3 (1994): 243–252.

[9] Malczewski, J. *GIS and Multicriteria Decision Analysis*. New York: Wiley, 1999.

[10] O'Brien, M. Feature extraction with the VLS Feature Analyst system. In *Proceedings of the American Society for Photogrammetry and Remote Sensing*, 2003.

[11] O'Brien M., Irvine J., Information fusion for feature extraction and the development of geospatial information. In *Proceedings of the Seventh International Conference on Information Fusion*, 2004, pp. 976-982.

[12] Opitz, D. Hierarchical feature extraction: removing the clutter. In *Proceedings of the International ESRI User Conference*, 2002, pp. 924-930.

[13] Royce, W. Managing the development of large software systems, In *Proceedings of the 9th International Conference on Software Engineering*, 1987, pp. 328-338.

[14] Stankiewicz, A. A view from above. *Geospatial Solutions*, 15.7 (2005): 30-35.

[15] Stojanovic' D., Djorjevi-Kajan S., Petkovic' M., Stoimenov L. Development and quality control of the spatial database for telecom network management in GIS. In *Proceedings of the 3rd Joint European Conference and Exhibition on Geographical Information*, 1997, pp. 1221-1230.