Adding New Agents and Models to the NED-2 Forest Management System

Donald Nute*, Walter D. Potter, Zhiyuan Cheng, Mayukh Dass, Astrid Glende, Frederick Maier, Cy Routh, Hajime Uchiyama, Jin Wang, Sarah Witzig

Artificial Intelligence Center
The University of Georgia
Athens, GA 30605, U.S.A.

dnute@uga.edu
potter@cs.uga.edu

Mark Twery, Peter Knopp, Scott Thomasma
Northeastern Research Station
USDA Forest Service
P.O. Box 968, Burlington, VT 05402-0968, USA
mtwery@fs.fed.us

H. Michael Rauscher
Creek Experimental Forest, Southern Research Station
USDA Forest Service
Asheville, NC, USA
mrauscher@fs.fed.us
Keywords: agents, blackboard architecture, ecosystem management, decision support system, knowledge based system, regeneration, fire risk analysis.

Summary: NED-2 is a robust, intelligent, goal-driven decision support system that integrates databases, growth and yield models, wildlife models, silvicultural expert systems, and geographical information systems into a single, unified decision support system for forest ecosystem management. NED-2 uses a blackboard architecture and a set of semi-autonomous agents to manage these tools for the user. Each agent has the procedural knowledge needed to operate a class of decision support tools used in forest ecosystem management. This paper focuses on how this multi-agent approach with its blackboard architecture supports the expansion of NED-2 to incorporate new models and decision support tools as they become available. We first describe the NED-2 architecture. Then we explain how new agents and models were added to an early version of NED-2 to expand its capabilities to include a new regeneration model, a fire risk analysis model, GIS capability, and the capability to create and use personalized specifications for silvicultural treatments.

1. Introduction

The goal of the NED project is to develop a decision support system for managing forest ecosystems that integrates a broad range of decision tools and models. The decision process modeled in NED-2 is goal-driven and the goals that drive the process
include timber, wildlife, forest health, fire risk management, water, and visual goals.

NED-1 (Twery et al. 2000) grafted knowledge-based goal evaluation modules onto a C++ program for inventory management and report generation. The next step was to add components for creating and simulating silvicultural treatment plans. The NED-1 architecture did not easily support these enhancements.

The NED-2 architecture was developed with a few key design principles in mind. First, NED-2 users should be able to enter data and view outcomes using a single, simple interface regardless of how those outcomes are produced. Second, NED-2 should perform all conversions between the data formats used by different tools and models without user intervention. Third, NED-2 should have an open architecture that allows additional third-party components to be added to the system without extensive revision of existing NED-2 code. Fourth, developers of new NED-2 components should not need to know very much about how other components work. These design criteria resulted in three essential features of the NED-2 architecture. The first is a robust ontology that supports communication between NED-2 and a wide variety of forest management decision support tools. This ontology is implemented using both database and logic programming technologies. Second, NED-2 is an agent-based system in which semi-autonomous agents perform different tasks as parts of the overall decision process. Third, NED-2 agents coordinate their behaviors and communicate information using a blackboard architecture (Ni 1989).
We described the NED-2 architecture and NED-2 version 0.2 in (Nute et al., 2002). A companion to this paper (Twery et al., 2003) describes NED-2 functionally from the point of view of the forest manager and includes several figures showing the interface. In this paper, we will discuss NED-2 version 0.6. The first part of the paper provides a more detailed description of the NED-2 architecture. An important claim for the NED-2 blackboard architecture is that it facilitates incorporation of new components. The evolution of NED-2 version 0.6 from NED-2 version 0.2 put this claim to the test. Without any changes in the underlying architecture, we have provided the forest manager with the ability to create and share knowledge bases of parameter settings for silvicultural treatments, we have integrated new models for regeneration and fire risk analysis, and we have integrated GIS displays of the results of NED-2 analyses using the commercial product ArcMap. The second part of the paper will explain how these enhancements were accomplished.

2. Overview of the NED-2 Architecture

Wherever possible, NED-2 integrates already existing tools. So far, these include a Microsoft Access database, the Forest Vegetation Simulator (FVS) (Crookston 1997), and ArcMap. For some tasks, including goal analysis, no appropriate tools were available and the NED developers had to construct the needed component. Whenever we integrate a third-party component, a NED-2 agent is developed that knows how to use that component. Where there is no available third-party component, the necessary functionality is built directly into the NED-2 agent.
A complex software system that integrates many components can have any number of architectures. One fundamental design question is whether control of the system will be centralized or distributed. The NED architecture is distributed. This was seen as the best choice for a system that allowed additional components of unanticipated kinds to be added later. NED-2 is a community of agents, each performing some specialized task. Another question is how the components will communicate with each other. Hierarchical models and other architectures where one agent sends requests or information directly to other agents require that agents have considerable knowledge of what other agents do and of what information they need to perform their tasks. NED-2 uses a blackboard architecture. With this approach, all information and requests for tasks to be performed are placed on a blackboard where all agents can view them. This approach minimizes the amount of knowledge each agent must have about other agents in the system.

The NED-2 blackboard system and all of the NED-2 agents are implemented in WIP- Prolog 4.300, a commercial implementation of the logic programming language Prolog (LPA 2003). ProData, an extension of Prolog to support access to an external database using SQL queries, is used to integrate a Microsoft Access database into the NED-2 blackboard. Most of the NED-2 user interface and the codes for computing basal area, volume, and other values are developed as Dynamic Link Libraries (DLLs) in Microsoft Visual C++. Different Prolog agents call functions in these DLLs as appropriate, but a
single Prolog interface agent handles most of the communication between NED-2 and the user through the C++ interface.

**Figure 1 goes about here.**

### 3. The NED-2 Ontology and Internal Data Model

There are specialized tools such as DAML (Hendler and McGuinness 2000) for representing ontologies. However, the ultimate purpose of an ontology is to provide a system with the ability to represent domain knowledge and to utilize information and knowledge about the domain. We have adopted a less abstract and more practically motivated method for representing the NED-2 ontology. The ontology for NED-2 is incorporated into the design of the NED-2 database and the design of a set of Prolog clauses that store temporary information during a NED-2 session. To communicate with a variety of external components, to represent a forest over time, to evaluate a potentially large set of management goals, and to generate all the reports that a manager might require, NED-2 requires a robust ontology and internal data model.

A management unit is divided into stands. Database tables were designed to hold permanent information about the management unit and the stands such as location, size, and physical characteristics. Management goals must be coordinated with knowledge bases for evaluating whether a particular goal has been satisfied. So the set of available goals is limited to those included in the database in a goals table. New goals can be added to the system by entering the goal in the goals table and adding rules for evaluating the goal to the knowledge base. There is also a table that includes necessary
information about all the different reports NED-2 can generate. There will be a goal report for each goal in the system. Currently the goals table includes timber, wildlife, water, ecology, and visual goals. Fire management goals are under development.

Neither the management unit nor the stand is static. So most information about stands is stored as snapshots that represent the stand at a point in time under a particular treatment regime. The initial snapshot for each stand represents a year when inventory was taken for at least one plot in the stand. Each snapshot includes a set of observations for the overstory, understory, and groundcover component of the stand. These include individual tree data for the overstory and understory, and additional information for the understory and the ground layers concerning the presence of ponds, rocky areas, coarse woody debris, etc.

All plans are developed from a common baseline year. If inventory has not been taken on every stand during the baseline year, then simulated data are generated for that stand using a growth simulator. To generate the baseline and to simulate treatment plans later, a simulation model must be chosen for each stand. The user selects treatments from a pre-established list and either accepts default parameters or enters his own parameters for each treatment. The user’s treatment definitions are stored in a knowledge base and in a treatment table in the database.

The user develops one or more treatment scenarios for the management unit, specifying both silvicultural and non-silvicultural treatments and the years they are to be applied.
This information is stored in a [Scenarios] and a [Scenarios_design] table in the database. When a plan is simulated, NED-2 will generate pre- and post-treatment snapshots for each year that a treatment has been specified. Each snapshot is represented in the database by a snapshot number. This coordinates the snapshot observations with the treatment entry in the plan or scenario.

Facts are stored temporarily on the blackboard as Prolog clauses. Persistent information is stored in the database or in Prolog knowledge bases. Facts on the blackboard are represented as Attribute-Object-Value triples. The Attribute in an AOV triple is always represented as a simple “atom”, but the Object and the Value can be complex. For example, the triple

\[ tpa([\text{snapshot}(17), \text{species}(\text{oak}), \text{size}(6)], 24) \]

indicates that there are 24 six-inch oaks per acre in snapshot 17, while the triple

\[ \text{goal}([\text{balance_size_classes}], [[21, 0.0], [22, 0.4], [23, 1.0]]) \]

indicates that snapshot 21 satisfies the balanced size class goal with confidence 0, snapshot 22 with confidence 0.4, and snapshot 23 with confidence 1. Objects are always indicated by a list of identifiers. In the first example, the attribute is trees per acre; here the object is complex (six-inch oaks in snapshot 17) but the Value is simple (24). In the second example, the attribute is balanced size classes; now the object is simple (a
timber goal) but the value is complex (a list of snapshots together with the confidence that each snapshot satisfies the goal.)

The NED-2 ontology and internal data model must be robust enough to support exchange of information with all external components integrated into the NED-2 system. The structure of the database and of the Prolog facts on the blackboard provides the concepts of the ontology. We also need to establish critical relations between these concepts. These are provided partly by knowledge bases of rules relating the concepts and partly by utilities that calculate the values of certain attributes from the values of other attributes. These knowledge bases and utilities may be accessed by any NED-2 agent that needs them.

Finally, part of the ontology of the NED-2 system is stored in meta-knowledge bases that describe the structure of the NED-2 database including relations between information in different tables. The function of these meta-knowledge bases is described in the next section.

4. The NED-2 Blackboard/Database Integration

When a user opens a NED-2 working file containing his inventory, goals, treatment definitions, and plans, this database becomes an integral part of the NED-2 blackboard, not physically but conceptually. Any NED-2 agent may access a fact from the blackboard using the Prolog query
known(\text{Attribute}(\text{Object}, \text{Value}))

whether that fact is stored as a Prolog clause or as a record or set of records in the
database. To achieve this transparent integration, the blackboard must be an active set
of procedures rather than just a static set of facts. These procedures analyze the request
for information to determine how the request may be satisfied. First, the system looks to
see if there is a corresponding Prolog fact. If not, then the system determines whether or
not the information is stored in the database. To do this, the system must consult a meta-
knowledge base regarding the structure of the NED-2 database. This may require
performing joins across several tables. For example, the query

?- known( ba([ stand(17), plan(`Maximize Timber`),
year(2023 ) ], BA).

asks what the basal area of stand 17 will be in 2023 if we implement the plan called
`Maximize Timber`. The blackboard will look in the [Scenarios] table to find the
scenario number for the `Maximize Timber` plan. Then it will look in the
[Scenarios_design] table to find the snapshot number corresponding to that plan for
stand 17 and year 2023. Then it will look in the [Treatment_measurements] table to find
the basal area for that snapshot, which will have been calculated from values in the
[Overstory_obs] table when the plan was simulated. All of this will occur automatically
without user involvement. The developer building a new NED-2 agent does not need to
know the underlying structure of the database that supports the NED-2 ontology in
order to formulate his query because the blackboard knows this. The knowledge is provided by the meta-knowledge base.

Updating knowledge on the blackboard is more complex. Any agent can put temporary data on the blackboard as a Prolog clause, but only certain agents may update persistent information in the database. These include the interface agent, the treatment development agent, and the simulation agent all described below. These agents construct specific SQL queries for database update. So these agents must have specific knowledge of the database and its relationship to the NED-2 ontology. For more details on integrating the database with the Prolog blackboard, see (Maier 2002; Maier et al. 2002).

Requests are also stored on the blackboard as Prolog clauses. Requests can be as simple as

\[
\text{request(interface)}. \\
\]

or as complex as

\[
\text{request([analysis('Wildlife',[\text{american\_goldfinch, cedar\_waxwing, northern\_flying\_squirrel}], [inventory, baseline]), analysis('Timber', [cubic\_foot], [inventory, baseline]) arcview([\text{american\_goldfinch, cedar\_waxwing,}}\\
\]

Donald Nute
The first example simply requests that the interface dialogs be enabled. This is the request that initiates interaction with the user when NED-2 starts. The second request is actually a plan of action including several component requests: analyze the inventory and baseline data to see whether habitat is available for the American goldfinch, the cedar waxwing, and the northern flying squirrel; analyze the inventory and baseline data to see whether the goal of focusing timber production on cubic foot production is satisfied; and display the results of this analysis on a map defined in the shape file `mystandmap.shp`. Different agents will satisfy these requests as explained below.

5. Adding a Treatments Definition Editor to NED-2

Treatment plans are developed in NED-2 using a spreadsheet dialog box. Rows in the spreadsheet represent stands in the management unit and columns represent years. The plan begins with a single column for the baseline year and the user adds additional years to the plan as needed. By double-clicking on a cell in the spreadsheet, the user can call up a treatment selection dialog. Multiple treatments can be entered in a plan for a single stand and year. The planning module, part of the NED-2 C++ user interface, includes plan integrity checks. If treatments are entered that would invalidate later treatments already entered for that stand, the later treatments are removed from the plan. Plans can be stored, edited, and copied for alteration. Every treatment in a plan is represented in
the NED-2 internal data model as a row of values in the [Scenario_designs] table. The keys for this table are the plan or SCENARIO number, the STAND number, and a SEQUENCE number representing the position of that treatment in the sequence of treatments defining the plan for that stand. For every cell in the plan spreadsheet except the baseline cells, there is an implicit growth treatment that occurs before any other treatments. So a stand is grown for the appropriate number of years before any other treatments are applied to it.

In NED-2 version 0.2, only five treatments are available: clearcutting and light and heavy thinning from above and below using basal area. Residual basal areas and other parameters for these treatments are ‘hard-wired’ in the simulation agent. Basic information about these five treatments is stored in a [Treatments] table in every NED-2 working file. When the user requests plan simulation, the NED-2 C++ interface passes a message to the Prolog interface agent. The interface agent puts a request for simulation on the blackboard and suspends the currently active interface module. The simulation agent then responds to the request by simulating every treatment (including growth treatments) in every plan the user has saved for every stand in the management unit. This arrangement was adequate for developing and testing the original simulation agent, but we need more flexibility in the treatments that can be used in plans and in the way plans are simulated.

The first requirement was a method for including more treatments in NED-2. We also wanted to allow users to modify the parameters for any treatments that were already
The process for adding treatment definitions to NED-2 involved four steps. First, we designed the format for a treatments knowledge base to hold a user’s treatment specifications. Second, we developed the user dialogs to allow the user to review and edit the specifications for treatments. Third, we developed a new NED-2 agent that operates the treatment definition editor and creates the user’s treatments knowledge base. Fourth, we made necessary changes in the simulation agent and the C++ planning module.

We began by creating a default treatments knowledge base that includes a library of standard treatments together with default parameter settings for these treatments. This knowledge base contains a set of Prolog clauses with the following form:

```prolog
def_treatment(fvs(_), 'Clearcut', '5_clearcut_2.bmp', '',
               clearcut, [[keyword(thindbh),
                           field(2,'Minimum DBH','1.0'),
                           field(3,'Maximum DBH','999.0'),
                           field(4,'Cutting Efficiency','1.0'),
                           field(5,'Species','ALL'),
                           field(6,'Residual TPA','0'),
                           field(7,'Residual BA','0')]].
```

The first argument in each of these clauses gives the simulation model for which the treatment is defined. In this case, the default treatment definition applies to both the
northeastern and southern variants of the Forest Vegetation Simulator (fvs(ne) and
fvs(sn)). The second argument is the name of the treatment, the third is an icon that will
be used to represent the treatment in the planning spreadsheet, and the fourth is a
description of the treatment. The user can modify these values using the treatments
editor. The fifth argument indicates the treatment type; this argument is internal to
NED-2 and cannot be altered by the user. Finally, we get a list of values including the
keywords used by FVS to describe the treatment together with a list of parameters and
values the simulator will use to implement the treatment. Here we have defined a
clearcut as a thinning based on the diameter (dbh) of the trees to be cut where we will
cut all trees of all species regardless of dbh with a cutting efficiency of 1.0 leaving no
residual basal area and no residual trees per acre.

After the knowledge base of default treatments had been constructed, the treatments
editor was designed. This is a series of dialogs written in WIN-Prolog 4.300. We used
Prolog rather than C++ for this part of the NED-2 interface because the treatments
editor would need to interpret Prolog clauses, build dialog boxes at run time based on
the information in these Prolog clauses, and then modify the Prolog clauses in response
to user actions.

When the user is building a treatment plan, he cannot directly access the treatments
definitions in the default treatments knowledge base. No treatments are available for the
user to use in plan development until he builds his own treatments knowledge base or
loads a treatments knowledge base that was created and saved previously. Minimally,
the user must pick the default treatments he wants to use in his plans and move them into his own treatments knowledge base together with the default parameter settings for the selected treatments.

The first dialog box for the treatments editor allows the user to select a simulation model and starts a new treatments knowledge base or selects an existing treatments knowledge base to edit. The user then selects one of the default treatments to add to his knowledge base. The editor creates a new dialog box in which the user can review fixed parameters and edit modifiable parameters for the treatment he has selected. The user can always change the name and icon for the treatment and can enter a brief description for the treatment. Which parameters can be modified will depend on the knowledge of the different treatment types built into the editor. For example, in a clearcut none of the parameters can be altered. However, for an actual thinning based on tree diameter (dbh), the user would be able to determine the minimum and maximum dbh for removal, the species to remove, and other parameters for the treatment.

When the user has finished building his own treatments knowledge base, he can save it under whatever name he chooses. The primary difference between the default treatments knowledge base and users’ treatments knowledge bases is that the predicate \texttt{def_treatments} is used in the default knowledge base while the predicate \texttt{treatments} is used in user treatments knowledge bases. Once saved, a user’s treatments knowledge base is available to use in as many NED-2 projects as he wants, and he can share his treatments knowledge base with other users.
Next we built a treatments definitions agent that includes all the code for the treatments editor plus additional Prolog code needed to integrate this agent with the rest of NED-2. Since the user must now create or select a treatments knowledge base before developing treatment plans, we added a function to the treatments definitions agent to do this. In NED-2 version 0.2, there was a static [Treatments] table in every NED-2 working file that told the C++ planning module which treatments were available to use in plans. This table is now dynamic. When the user selects a treatments knowledge base, the treatments agent builds the [Treatments] table in the current NED-2 working file. This table includes treatment names, icons, user descriptions, and types. It also includes the simulation model for each defined treatment. It does not contain any information about keywords used to control the simulator or treatment parameters since the planning module will never use this information.

5. Adding a Regeneration Model to NED-2

REGEN (Boucugnani et al. 2003) is an implementation of a regeneration model based on Loftis’ work (Loftis 1989, 1990). The model uses data for both the understory and overstory of a stand preceding the regeneration-triggering event to predict regeneration. A REGEN knowledge base contains functions used to compute the competitive ranking of both stump sprouts and seedlings for different species, and these rankings are used stochastically to select the winners during the regeneration process. Different knowledge bases can be developed by regeneration experts to represent different
geographical or ecological regions. The regeneration model was implemented as a
Prolog inference engine with an Excel user interface. Our goal was to integrate the
REGEN inference engine into NED-2.

We expanded the functionality of the NED-2 simulation agent to incorporate REGEN.
The simulation agent knows how to use two variants of the Forest Vegetation Simulator
(FVS). These are independent programs that read two input files, one containing stand
inventory data and the other containing a script that controls the simulation. As the
simulation programs run, they output files containing simulated tree data and cut lists
for treatments. Regeneration models are included in the FVS variants. If the user
decides to use the Loftis regeneration model instead of the regeneration model built into
one of the FVS variants, then the simulation agent must *interleave* the growth model it
is using (an FVS variant) with the Loftis regeneration model.

The NED-2 C++ planning module was modified so that the user can specify for each
plan which growth model and which regeneration model should be used for each stand.
To use REGEN, the user must select one of the REGEN species competition knowledge
bases. When the simulation agent identifies a treatment that will trigger regeneration for
a stand where the user has selected REGEN as the regeneration model, it passes to the
REGEN inference engine the snapshot number for the real or simulated tree data
preceding the treatment, the REGEN knowledge base to be used in the simulation, and a
snapshot number where the results of the simulation are to be written. After REGEN
completes the simulated regeneration, the simulation agent continues with any further
simulation tasks.

[Figure 2 goes about here.]

6. Adding a Fire Risk Analysis Model to NED-2

The fire risk analysis model for NED-2 is based upon separate wildlands (Hemel 2003)
and urban interface (Long 2003) models. The first step was to determine what variables
were needed to support these models and then add these to the NED-2 internal data
model. We added a new dialog to the C++ inventory module to collect and store the
information we needed about structures on the management unit, and a [Buildings] table
to the NED-2 working file the store the new information. We also added a variable for
litter depth to the understory observations section of the NED-2 working file. We made
corresponding changes in the NED Variables database and ran the program we use to
generate the meta-knowledge bases the NED-2 blackboard handler uses to integrate the
Prolog and Access parts of the blackboard. Next, we developed rules to determine the
fire risk analysis for a stand based upon our models. (There are interesting issues in
knowledge acquisition we could discuss here, but they exceed the scope of this paper.)

A NED-2 fire risk agent was developed. This agent responds to requests for fire risk
analysis by calling the appropriate NED-2 inference engine with a set of goals that
includes information about which snapshots (representing actual inventory, the baseline
year, or a specific year in a treatment plan the user has created) are to be analyzed. The
results of this analysis are placed on the blackboard by the inference engine. If the user
has requested a set of reports that includes a fire risk analysis, the fire risk agent reads
the results of the analysis from the blackboard, writes the report as an HTML file,
removes the results of the analysis from the blackboard, and puts the name of the fire
risk analysis report on the blackboard where it can later be found by another agent that
creates a top-level HTML file for the set of reports and sends it to the default Web
browser. If the user has requested that the results of the analysis be displayed on a map
of the management unit, the fire risk agent leaves the results on the blackboard where
the GIS agent described below can fine them.

7. Adding GIS to NED-2

The NED-2 simulation agent prepares input files for an external simulation program,
and then executes that program. The NED-2 GIS agent works much the same way
except that the external simulation program is replaced by an ArcMap project.

We wanted to be able to paint a map of the management unit to show the values of
different variables for each stand. For example, we might display forest type, basal area,
or percentage of overstory coverage. We also wanted to be able to color a map to show
which stands satisfy some goal. Furthermore, we wanted our map to have layers
representing different years for the same plan or different plans for the same year.

The first step was to design a database that would contain the information an ArcMap
project would need to accomplish our goal. Since NED-2 already uses an Access
database as the NED-2 working file, we decided to store this information in a separate
Access database with a name determined by the user. (This would allow the user to
access the data in ArcMap later without first starting up NED-2.) Then we created code
in the Visual Basic Application language (VBA) that told our ArcMap project what to
do when it was executed. The VBA code looks to see if a temporary file with a special
name exists. If it does, then the code reads this file to get a name of an Access database
and an ArcMap shape file. It merges the data in these two files to create the data file
that will be used to drive the ArcMap displays. (If the ArcMap project does not find a
file containing the names of an Access database and an shape file, it prompts the user
for the database and the shape file.) Finally, we included VBA code that allows the user
to select the variable or goal to be used to color the maps. While ArcMap is running, he
can switch between any of the variables or goals that are included in the database. He
can also switch between layers in the usual way to change the year or the treatment plan
the map represents.

Once we had designed the ArcMap project and the database that would drive it, we
developed the NED-2 agent that would create the database for the ArcMap display and
then execute the ArcMap project. This GIS agent performs the final step in a series of
steps that are planned by a special NED-2 planning agent. When the user requests a GIS
display, this planning agent responds to the request and asks the agent whether he wants
to display information for real inventory, for the baseline year, for all the years included
in a treatment plan, or for all treatment plans for a specific year. The user is also asked
which variables and goals he wants to display. The planning agent prepares a list of
requests corresponding to the user’s answers and puts them on the blackboard. The
different goal analysis agents perform the goal analysis and put the results on the
blackboard. Then the GIS agent takes the results of the goal analysis off the blackboard
and puts these, together with the values for the variables the user selected, into the
special Access database. It then writes the command file containing the name of the
Access database and the name of the shape file the user has entered previously as part of
the management unit information. Finally, the GIS agent executes the NED-2/ArcMap
project. The NED-2 interface agent re-activates the NED-2 C++ interface, and the user
can switch between NED-2 and the ArcMap display.

8. Activating the New NED-2 Functions

The final step in adding any new function to NED-2 is to make a few changes to the
NED-2 interface agent. First, we add the names of any new files that contain the code
for any new agents or utilities to the list of files that NED-2 loads at start-up. Next, we
modify the user interface so the user can access the new function. The primary NED-2
interface is a full-screen window that is divided into several window ‘panes’. The A-
pane is a region in the upper left corner of the screen that always displays an outline of
the NED-2 decision process. At start-up, Prolog loads all of the Dynamic Link Libraries
(DLLs) that make up the C++ user interface. It actives one of these, the home module
for the NED-2 user interface, and gives it the information it needs to build the contents
of the A-pane. Whenever a user clicks on an item in the A-pane, the currently active
C++ interface module sends the Prolog interface agent a message that tells it which item
was selected. The interface agent then consults a knowledge base to determine what action to take. To activate a new function to NED-2, we must also modify this knowledge base. As an example, we added ‘Define Treatment Sets’ and ‘Select Treatment Set’ under ‘Planning’ in the outline displayed in the A-pane. Then we added two rules to the knowledge base for the interface agent: one telling the interface agent to add a \texttt{treatments\_edit} request to the blackboard when the user clicks ‘Define Treatment Sets’ and another telling the interface agent to add a \texttt{treatments\_select} request to the blackboard when the user clicks ‘Select Treatment Set’.

9. NED-2 Expandability

Integrating several decision support tools into a single DSS like NED-2 has been challenging. The two foundations for this integration are the rich NED-2 ontology and internal data model, and the distributed control using autonomous agents and a blackboard architecture. The ontology and data model make it possible to develop semantics for the data formats and control structures of external models. Programmers developing different NED-2 agents can also depend on a stable model for the information those agents will use in performing their tasks.

The expandability of NED-2 was not really tested in developing NED-2 version 0.2 (Nute et al. 2002). The initial functions included in NED-2 were integrated into this version all at the same time. Subsequent versions leading up to and including the
current NED-2 version 0.6 have required us to add new agents and new models to an already existing, stable system. The capability to build knowledge bases representing different sets of treatment parameters, the fire risk analysis model, and the GIS capability were all added by building new NED-2 agents. A new regeneration model was added by expanding the scope of the already existing NED-2 simulation agent. These additions also required us to enhance the NED-2 data model and the modular C++ interface. In each case, the new components were developed and the changes were made in a few existing components without destabilizing the overall system. The blackboard system continued to provide communication between the new and old components. When it was necessary to enhance the NED-2 data model, already existing auxiliary programs were used to regenerate the meta-knowledge about the structure of the NED-2 working file that the blackboard system needs to integrate the Prolog clauses and the Access database. These expansions of the NED-2 system demonstrate the power of the multi-agent blackboard architecture used in NED-2.

All versions of NED-2 numbered 0.X represent test versions. These are being distributed to a limited number of users for testing and for use in case studies. NED-2 version 0.6 is very close to the distribution version of NED-2 that we plan to release in the first half of 2004. This version includes simulation models for the eastern United States and wildlife habitat models for the northeastern United States. While we are preparing the first version of NED-2 for general distribution, we continue to work on models to simulate change in the understory, a wildlife habitat model for the southeastern United States, agents to integrate visualization tools such as the landscape
visualization system ENVISION (McGaughey 2000), and a financial model to evaluate financial aspects of treatment plans. Our experience so far gives us confidence that we will be able to integrate these new functions seamlessly into NED-2. If resources permit, we should be able to expand NED-2 to cover the entire United States by incorporating addition simulation, regeneration, wildlife, and other models as they become available. As new kinds of decision support tools become available for this domain, we should be able to incorporate them into NED-2 as well. Developing a European or Asian or other version of NED-2 using our basic architecture and interface modules should be straightforward. The flexibility of this multi-agent approach and the blackboard architecture should allow us to replace many of the agents or models specific to the eastern United States with corresponding components developed for other regions of the world. As science produces better models of the different components of a forest ecosystem and as new decision support tools are developed for this domain, there will always be the possibility of incorporating them into the NED-2 architecture. Paradoxically, one of the greatest strengths of the NED-2 system is that it may never be completed.

10. References


Figure 1: NED-2 Blackboard Architecture with Multiple Agents.
Figure 2: Simulation with Interleaved Growth and Regeneration Models.