

# Naval Research Laboratory

Stennis Space Center, MS 39529-5004



NRL/FR/7240--94-9449

## Conversion of an Oceanographic Expert System to a C-Based Language

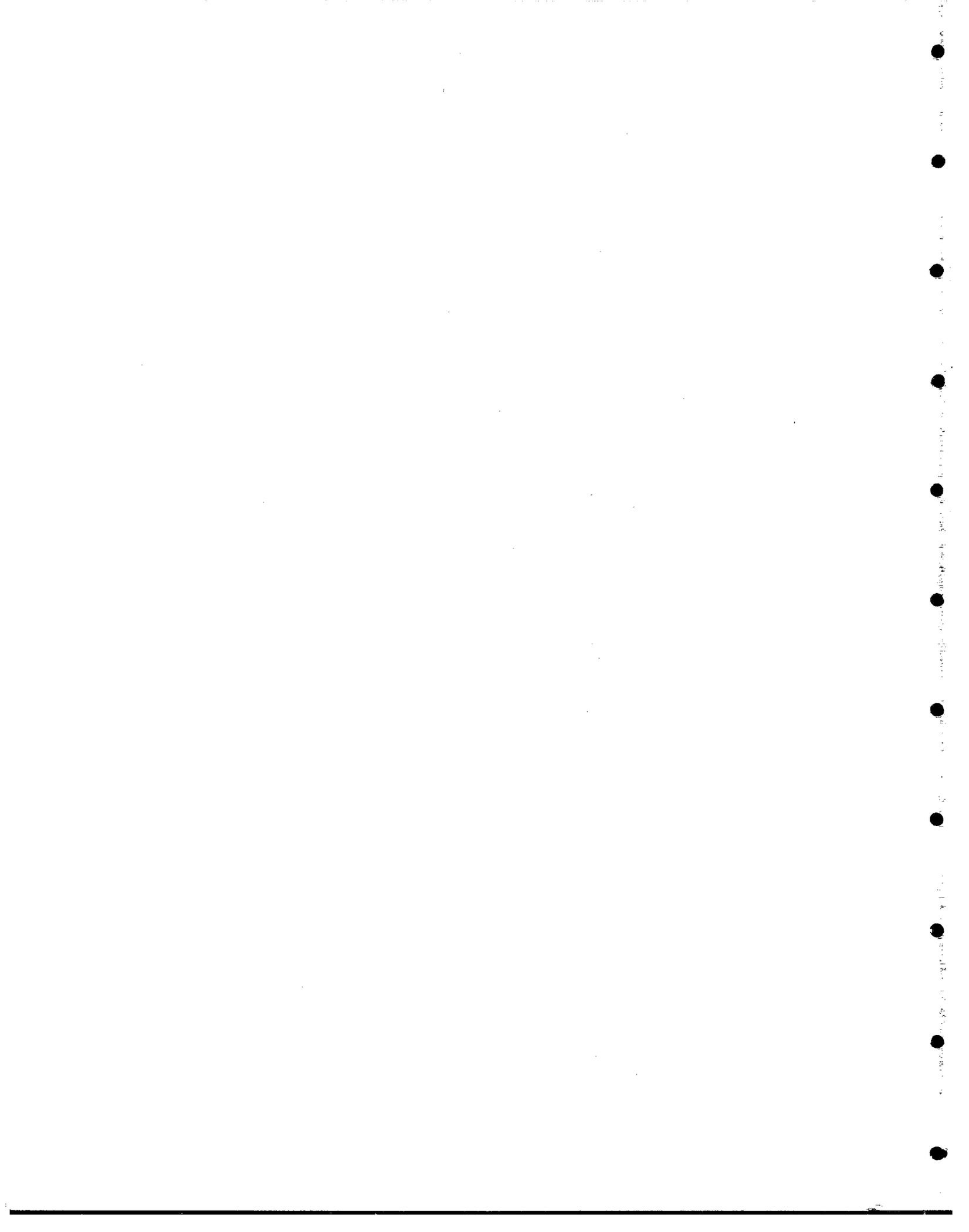
SUSAN BRIDGES  
LIANG-CHUN CHEN

*Computer Science Department  
Mississippi State University*

Prepared for Remote Sensing Division

February 1, 1995

Approved for public release; distribution unlimited.



# REPORT DOCUMENTATION PAGE

*Form Approved*  
**OBM No. 0704-0188**

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and reviewing the data, completing and reviewing the collection of information, sending comments regarding this burden or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

<b>1. AGENCY USE ONLY (Leave blank)</b>		<b>2. REPORT DATE</b> February 1, 1995	<b>3. REPORT TYPE AND DATES COVERED</b> Final	
<b>4. TITLE AND SUBTITLE</b> Conversion of an Oceanographic Expert System to a C-Based Language			<b>5. FUNDING NUMBERS</b> Job Order No. 572-5195-03 Program Element No. 0603704N Project No. X1598 Task No. 100 Accession No. DN255042 Contract No. NAS 13-330	
<b>6. AUTHOR(S)</b>  *Susan Bridges and Liang-Chun Chen			<b>8. PERFORMING ORGANIZATION REPORT NUMBER</b>  NRL/FR/7240--94-9449	
<b>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b> Computer Science Department Mississippi State University			<b>10. SPONSORING/MONITORING AGENCY REPORT NUMBER</b>	
<b>9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b> Space and Naval Warfare Systems Command 2451 Crystal Drive Arlington, VA 22202			<b>11. SUPPLEMENTARY NOTES</b>  *Prepared for Remote Sensing Applications Branch, Remote Sensing Division, Stennis Space Center, MS 39529-5004	
<b>12a. DISTRIBUTION/AVAILABILITY STATEMENT</b>  Approved for public release; distribution unlimited.			<b>12b. DISTRIBUTION CODE</b>	
<b>13. ABSTRACT (Maximum 200 words)</b>  The objectives of this project were to convert the NRL oceanographic expert system from the computer language OPS83 into the C language and CLIPS (C Language Integrated Production System), to redesign the control structure and the user interface and to improve the readability, understandability, and maintainability of the code. The eddy prediction component of an oceanographic expert system that was originally implemented in OPS83 has been translated to the CLIPS expert system shell. Portions of the system that were originally implemented in OPS83 procedural code have been translated to C. These changes will allow the system to be incorporated into the Semi-Automated Mesoscale Analysis System 1.2 (SAMAS 1.2). SAMAS 1.2 will eventually be incorporated into the third generation of the Navy's Tactical Environmental Support System, TESS(3), which does not support OPS83 code.  In addition to the translation tasks, the main control structure of the expert system was redesigned to achieve increased modularity and thus to improve the understandability of the code. An explanation component that was recently added to the system was also revised to improve maintainability. The revised and translated code was tested using several data sets that had previously been used to test the original system. The functionality of the revised system was exactly the same as that of the original system using all of the test data.				
<b>14. SUBJECT TERMS</b> data analysis, fronts and eddies, digital image analysis, remote sensing, satellite oceanography			<b>15. NUMBER OF PAGES</b> 35	
			<b>16. PRICE CODE</b>	
<b>17. SECURITY CLASSIFICATION OF REPORT</b> Unclassified	<b>18. SECURITY CLASSIFICATION OF THIS PAGE</b> Unclassified	<b>19. SECURITY CLASSIFICATION OF ABSTRACT</b> Unclassified	<b>20. LIMITATION OF ABSTRACT</b> Same as report	

•

•

•

•

•

•

•

•

•

•

•



## CONTENTS

1.0	INTRODUCTION .....	1
2.0	SYSTEM REQUIREMENTS .....	2
2.1	Functional Requirements .....	2
2.2	Interface Requirements .....	3
2.3	Software Requirements .....	4
2.4	Hardware Platform .....	4
2.5	Performance Requirements .....	4
3.0	SYSTEM DESIGN .....	5
3.1	Interfaces with Other Components in SAMAS 1.2 .....	5
3.2	Structured Design of WATE .....	5
3.3	Design Considerations .....	7
4.0	TRANSLATION OF OPS83 TO CLIPS AND C .....	8
4.1	Working Memory Elements in OPS83 and Templates in CLIPS .....	9
4.2	Rules in OPS83 and CLIPS .....	10
4.3	Procedural Code in OPS83 and in C .....	12
4.4	Invoking the Rule System in OPS83 and in Embedded CLIPS .....	13
5.0	IMPLEMENTATION .....	13
5.1	Coding and Naming Conventions .....	13
5.2	Embedding CLIPS in C .....	14
5.3	Running WATE .....	16
5.4	Initialization Component .....	16
5.5	Get User Options Module .....	17
5.6	Eddy Component .....	18
5.7	Explanation Component .....	20
5.8	Final Output Component .....	21
6.0	SYSTEM TESTING .....	22
6.1	Testing Criteria .....	22
6.2	Test Cases .....	22
6.3	Test Results .....	22

7.0	EVALUATION AND CONCLUSIONS .....	22
8.0	ACKNOWLEDGMENTS .....	23
9.0	REFERENCES .....	23
	APPENDIX A — Format of Input Files .....	25
	APPENDIX B — Test Cases .....	27
	APPENDIX C — Test Results .....	31
	APPENDIX D — Description of Files Containing Source Code .....	33

# CONVERSION OF AN OCEANOGRAPHIC EXPERT SYSTEM TO A C-BASED LANGUAGE

## 1.0 INTRODUCTION

The Naval Research Laboratory (NRL) has developed an oceanographic expert system that describes the evolution of mesoscale features in the Gulf Stream region of the northwest Atlantic Ocean (Thomason and Blake 1986). These features include the Gulf Stream current and the warm- and cold-core eddies associated with the Gulf Stream. An explanation capability, which was recently added to the eddy prediction component of the expert system, allows the system to justify the reasoning process it uses to provide predictions (Bridges and Lybanon 1993). The oceanographic expert system main program was implemented in the C language; the expert system rules were written in OPS83, and some graphics routines were implemented in Fortran.

The oceanographic expert system contains two major components: one predicts movement of the Gulf Stream, the other predicts movement of the eddies. Both were originally implemented in C and OPS83. The Gulf Stream component has been rewritten by NRL, and predictions are currently computed by a neural network system written in C code (Peckinpaugh 1993). The project described in this report is primarily concerned with the component that predicts eddy movement (referred to hereafter as the Expert System). The main purpose of this project was to convert the Expert System from OPS83 to C and CLIPS (C Language Integrated Production System). This conversion will allow the system to be incorporated into the Semi-Automated Mesoscale Analysis System 1.2, SAMAS 1.2 (Peckinpaugh 1993). SAMAS 1.2 will be eventually embedded into the third generation of the Navy's Tactical Environmental Support System TESS(3) (Lybanon 1992), which does not support OPS83 code. CLIPS is an expert system tool developed by the National Aeronautics and Space Administration that has been approved for use in TESS(3). In addition to the conversion task, the Expert System's capability to be read, understood, and maintained also needed to be improved. It was also necessary to modify interfaces with other components of SAMAS 1.2 during the conversion process.

This project had three major objectives. First, the rule system of the Expert System needed to be converted into CLIPS to facilitate the incorporation of SAMAS 1.2 into TESS(3). SAMAS 1.2 has been developed by NRL to automate the analysis and interpretation of satellite infrared images in the Gulf Stream region of the northwest Atlantic Ocean. The Expert System is used to predict eddy movement when the target area is obscured by clouds. TESS(3) a system currently under development, intended for use on ships and in some land-based facilities, will provide information about the Gulf Stream and the associated eddies.

Second, much of the interface and control structure of the Expert System was implemented in OPS83 procedural code and needed to be rewritten in a language suitable for TESS(3). In addition, the structure and quality of the code had deteriorated with many years of maintenance by several

different groups. The code included many obsolete variables and statements that had originally been used for debugging. The lack of comment statements and coding conventions made the source code very difficult to read. The control structure of the Expert System needed to be redesigned to modularize the functions provided by the Expert System. One module often contained more than one function and the flow of control among modules was convoluted. The control structure was redesigned so that each module performs one function with the goal of improving the understandability and the maintenance of the code. Also, the user interface that provides the user with different options for running the Expert System was improved by implementing a window-based interface.

Third, the explanation module of the Expert System needed to be implemented in a more flexible manner. The module that generated the explanations was hard-coded in C. Each prediction rule of the Expert System had an associated procedure that saved designated values to an array of linked lists each time the rule fired. When explanations were requested, the content of the explanation was constructed based on pre-stored text with certain slots that could be filled with corresponding values stored in the linked lists. This construction meant that every time a rule was changed, added, or deleted the corresponding C procedure also must be changed, all of which made the explanation system difficult to maintain. The goal was to implement the explanation system using the rule-based paradigm. In other words, to make the explanation system easier to understand and update, meta-rules (rules about rules) were used to reason about the rules that fire when a prediction is made.

The overall objectives of the project were to convert the Expert System from OPS83 into C and CLIPS, to redesign the control structure and the user interface, and to improve the readability, understandability, and maintainability of the code. The converted system developed in this project has been named WATE 1.0 (an acronym for Where Are Those Eddies). The deliverables of this project include the WATE system and the written documentation of the system design, implementation, and test results.

## 2.0 SYSTEM REQUIREMENTS

WATE is a revision of an existing system. The general requirements of WATE were (1) that it produce the same prediction output as the OPS83 version of the Expert System when given the same input data, (2) that it be able to run as a stand-alone system or integrated with other components of SAMAS 1.2, and (3) that the readability, understandability, and maintainability of the system be improved by removing obsolete variables and statements, modularizing the system, and making the explanation system more adaptable to future enhancements. In addition, WATE had the following functional requirements, interface requirements, and software and hardware requirements.

### 2.1 Functional Requirements

#### 2.1.1 Input

The system is required to read the following four input files (file format specifications are given in Appendix A.)

- Upper Gulf Stream (UGS) data file. The system must be able to read the file that describes a Gulf Stream north wall. If the north wall data is not provided, then default values for the north wall will be used (called the nominal north wall).

- Eddy data file. The system must be able to read the file that describes the position and size of both warm-core rings (WCR) and cold-core rings (CCR) in the northwest Atlantic Ocean.

- Parameter file. The system must be able to read the file that describes parameters for both WCRs and CCRs. In each region (nine regions are defined in the study area), parameters are specified for each of 16 compass directions. The WCR parameters are given as latitude and longitude adjustment factors, and the CCR parameters are given as heading and final direction values.

- Region file. The system must be able to read the file that describes default values, such as speed, direction, heading, minimum radius, and so on, for each region.

### 2.1.2 Output

WATE must be able to correctly generate the following outputs:

- General output. WATE must be able to store the general output into a set of external files specified by the user. The general output must contain the final predicted positions of the north and south walls, as well as the eddies. Gulf Stream positions will be followed by a source code of "O." Eddy positions will be followed by a source code of "ES." For example, a line of the north wall output file is as follows:

```
34.9707    -75.0016    O.
```

A line of the eddy output file is as follows:

```
34.9039    -72.9087    55.3934    C    ES,
```

where "C" indicates CCR. A more detailed description of the general output can be found in Peckinpugh (1993).

- Rule trace. WATE must be able to generate a natural language translation of the sequence of rules that have been activated during every prediction cycle (Bridges and Lybanon 1993).

- Explanation of results. WATE must be able to generate natural language summaries that describe how the system made a prediction and that are of at least the same level of sophistication as those produced by the Expert System (Bridges and Lybanon 1993).

- Graphical display. WATE must be able to interface with the current graphics routines that display the status of the Gulf Stream and eddies at each time step (Peckinpugh 1993).

## 2.2 Interface Requirements

### 2.2.1 User Interface

When WATE is invoked either from SAMAS (integrated mode) or by entering the executable name at the UNIX prompt (stand-alone mode), the user should be given the following options:

- Options for the PV-Wave graphic display. If the PV-Wave option is selected, the user should be provided the following additional options: (1) displaying eddies with or without trails, (2) displaying the Gulf Stream with or without keeping previous Gulf Stream drawings, and (3) displaying grids and/or regions.

- Options for selecting UGS data file and eddy data file.

- Options for the number of days in each time step.

During the session, the user should be able to control the prediction cycle and should be given the option of quitting after each time step. When quit is selected, the user should have the option of seeing explanations for each input eddy or for all eddies. Explanations for all eddies during the prediction cycles should be stored in a file named *explan.dat*. The type of explanation should be either rule trace or summary. The user should be able to quit the explanation session.

After the user leaves the explanation session, an option for saving the predicted final status of the Gulf Stream north wall, Gulf Stream south wall, and eddies with active status to external files should be allowed.

### 2.2.2 External Interface

WATE should interface with the following components in SAMAS 1.2 (Fig. 1):

- User Interface Module — including the Expert System driver and the PV-Wave Display Module.
- Geometry Routines.
- Gulf Stream Prediction Module.

## 2.3 Software Requirements

WATE should be written in CLIPS 5.1 and the K&R (after Kernighan and Ritchie 1978) C language.

## 2.4 Hardware Platform

WATE must run on a Sun Sparc Station.

## 2.5 Performance Requirements

Performance is not a critical requirement of WATE, but the system should not be noticeably slower than the current system.

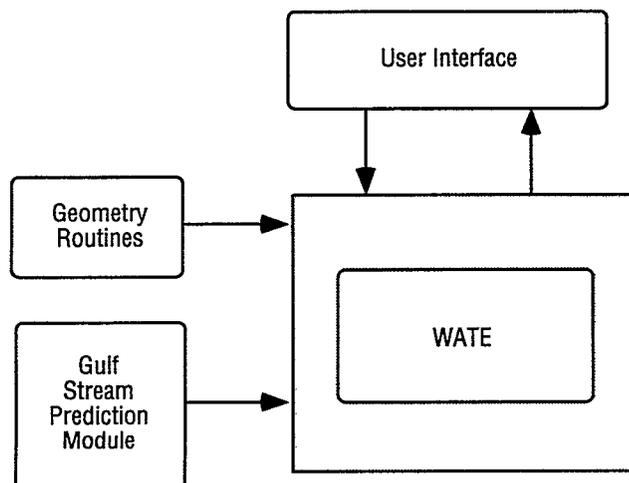


Fig. 1 — Interfaces of WATE and external components

### 3.0 SYSTEM DESIGN

The main goals of the design phase were that the resulting system should be reimplemented in CLIPS and be more readable, understandable, and maintainable. Therefore, the general approaches were to modularize the Expert System, to simplify the control structure, and to define the module interfaces.

#### 3.1 Interfaces with Other Components in SAMAS 1.2

The control structure of the Expert System was written in OPS83. The interface of the Expert System with other components in SAMAS 1.2 was implemented by calling C routines associated with each external module. Although the WATE design required that the control structure written in OPS83 code be converted to C code, the interfaces of WATE with other SAMAS components could be implemented by retaining the original C routine calls.

#### 3.2 Structured Design of WATE

WATE contains four main components as shown in the structure chart in Fig. 2:

- Initialization Component, containing five modules: Init Globals, Get User Options, Setup CLIPS, Setup PV Wave, and Setup GS (Gulf Stream).
- Eddy Component.
- Explanation Component, containing two modules: Explanation Driver and Presentation Module.
- Final Output Component.

##### 3.2.1 Module Specifications

Each module of WATE is specified in terms of its interface, its abstract behavior, and its submodules. WATE first invokes the Initialization Component, which contains the following modules. The modules are invoked in the order in which they are listed.

- *Init Globals* is invoked by the main module. Its function is to initialize global variables that are related to the prediction process. After it finishes its work, control is returned to the main module.

- *Get User Options* is invoked by the main module. This module provides the user with options specified in the Interface Requirements section of the System Requirements. Control is returned to the main module.

- *Setup CLIPS* is invoked by the main module. It sets up an appropriate environment for the Eddy Component based on the user options obtained by the Get User Options module. This module contains four submodules:

- *Init CLIPS* initializes the CLIPS system.

- *Load Constructs* loads the knowledge structures (discussed in the next section), eddy movement rules, and explanation rules.

- *Reset CLIPS* resets the CLIPS system, making loaded facts and rules active to the CLIPS inference engine.

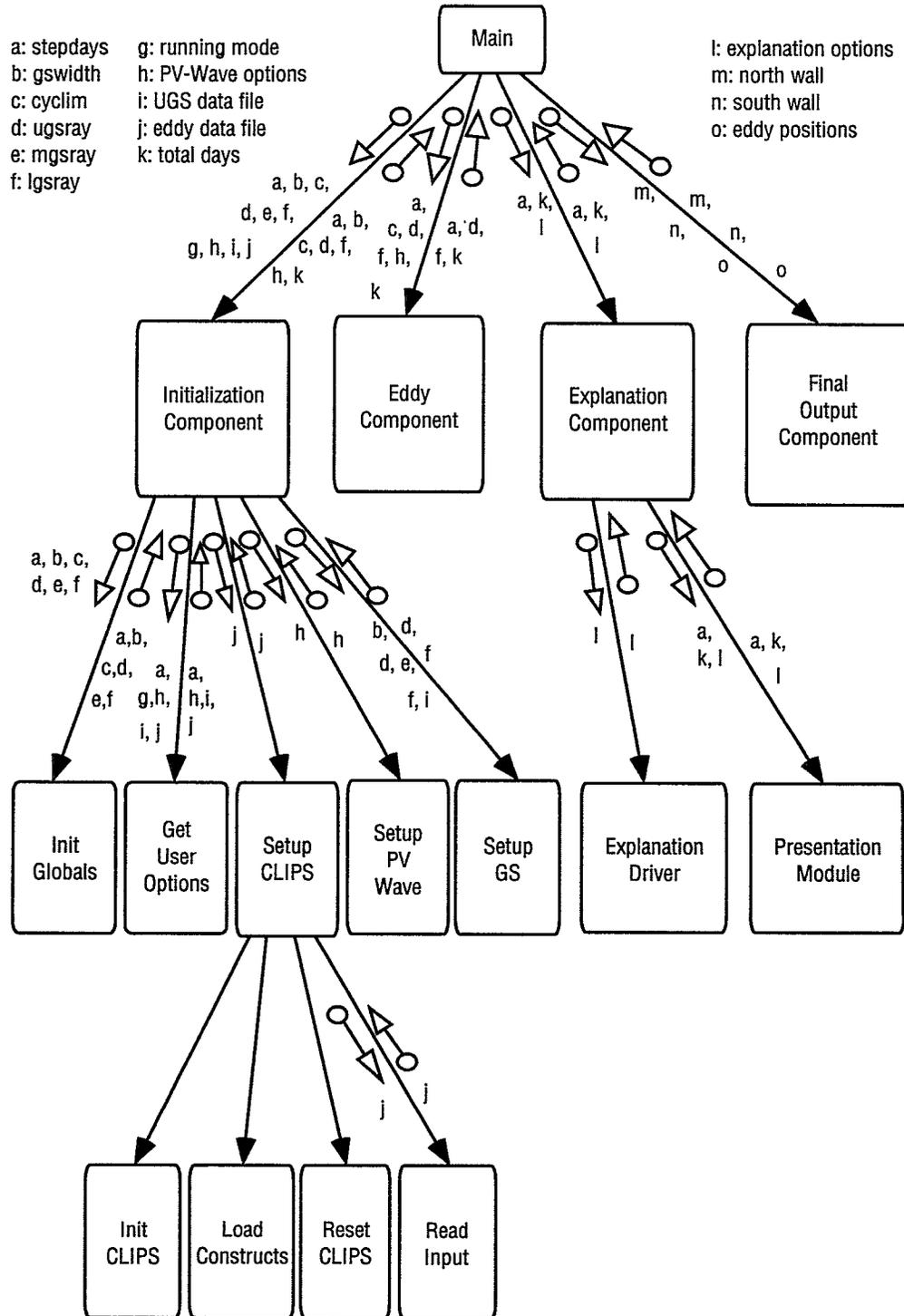


Fig. 2 — Structure chart of WATE

— *Read Input* reads in parameters, region data, and eddy data. Modules, including *Init CLIPS*, *Load Constructs*, and *Reset CLIPS*, will be discussed in detail in the next section.

- *Setup PV Wave* is invoked by the main module. Based on the selected PV-Wave options, this module interfaces with external PV-Wave display routines in SAMAS 1.2.

- *Setup GS* is invoked by the main module. This module uses a default value for the width of the Gulf Stream and the user-specified UGS data file to build up a Gulf Stream south wall. It then returns the control over to the main module.

After initialization is complete, control passes to the Eddy Component, which controls the prediction of eddy movement. The Eddy Component interfaces with CLIPS, and the prediction is done by a rule system written in CLIPS. As the rules fire, the reasoning process of the rule system is recorded as a set of CLIPS facts. During the prediction session, the user can ask for a prediction for the next time step or quit the prediction session. After the user quits the prediction session, control is returned to the main module.

The main module can invoke the Explanation Component if the user requests an explanation. The Explanation Component of WATE contains two modules:

- Explanation Driver displays an explanation menu and asserts appropriate facts to CLIPS based on the user's selection of explanation preference.

- Presentation Module contains explanation rules and uses them to construct an explanation of the reasoning process of the Eddy Component.

The Explanation Component will be discussed in more detail in the next section. After the user quits the explanation session, control returns to the main module.

The main module then invokes the Final Output Component. The Final Output Component prompts the user for options for saving the north wall, the south wall, and eddy positions to external files. Then control returns to the main module, which either terminates WATE (in integrated mode) or again invokes the *Get User Options* module (in stand-alone mode).

### 3.3 Design Considerations

Several issues were considered in the redesign of the system. First, the rule system of the Eddy Component needed to be reimplemented in CLIPS. This process required that the interfaces between the rule system of the Eddy Component and its external modules be modified to incorporate the external C routines and the CLIPS rules.

Second, the control structure of the Expert System needed to be redesigned to modularize the system and make it more understandable. The original Expert System was not well modularized in the sense of one module performing one function.

Finally, the Explanation Component of WATE was reimplemented in a declarative form to make maintenance of the explanation system easier. The tasks performed by the Explanation Component of the original Expert System were implicitly (or procedurally) implemented in C routines. These included routines for recording activation of the agenda, storing variable names and their values, and displaying the pre-stored templates with slots filled by appropriate values. Most of these routines required extensive revision to interact with the CLIPS inference engine instead of the OPS inference engine. All of these made the explanation system hard to maintain.

The Eddy Component of WATE records the reasoning process of the rule system as a set of CLIPS facts asserted in working memory rather than using external C routines to store information in complex lists. Figure 3 illustrates the interface of the Explanation Component of WATE with CLIPS. The Explanation Driver prompts the user for the type of explanation to be generated (rule trace or summary and the decision to be explained). This information is used by the Explanation Driver to assert facts into CLIPS that describe the type of explanation to be generated. The explanation rules use both the CLIPS facts that recorded during the reasoning process of the Eddy Component and the CLIPS facts that were asserted by the Explanation Driver to construct the appropriate explanation. The Explanation Component in the new system is more independent of the Eddy Component because the reasoning process does not need to be recorded by separate procedures as it was in the original Expert System. This simplification enhances the understandability and maintainability of the explanation system.

#### 4.0 TRANSLATION OF OPS83 TO CLIPS AND C

Both OPS83 and CLIPS support the rule-based paradigm. A rule-based system has three basic components:

- Working memory — contains a set of facts
- Knowledge base — contains a set of rules
- Inference engine — is responsible for the reasoning based on the facts and rules.

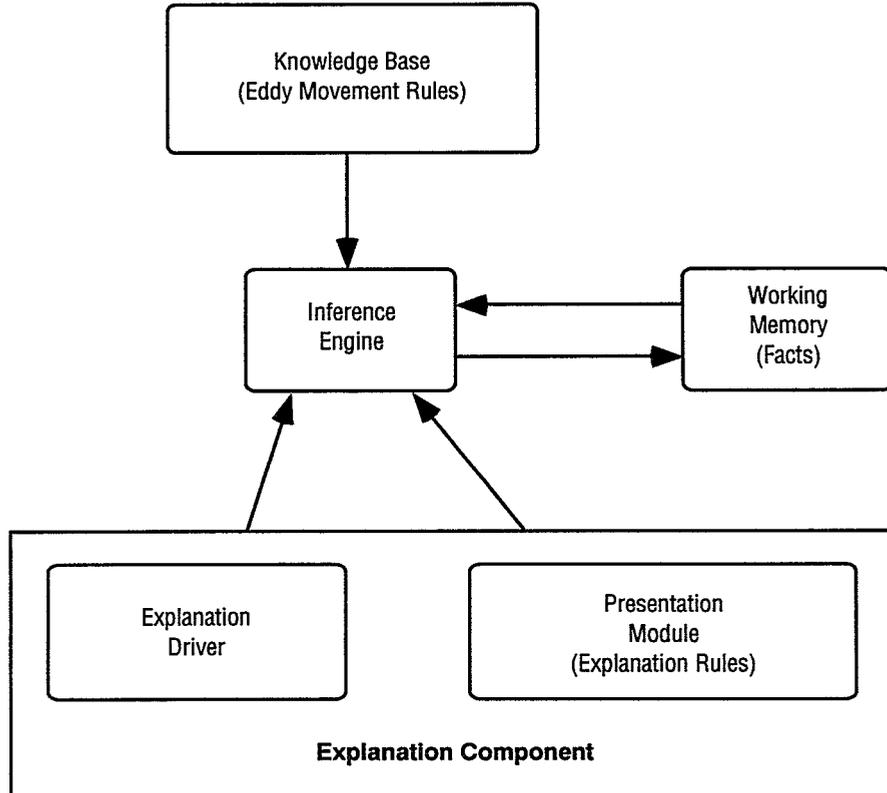


Fig. 3 — Interface of WATE's Explanation Component and CLIPS

An overview of the process for translating OPS83 code to CLIPS and C is discussed in this section. The OPS83 users manual (Forgy 1986) gives a detailed description of the language's functionality. Because both OPS83 and CLIPS are descendants of the OPS production language and both are forward-chaining expert system shells, the mapping from OPS83 to CLIPS was fairly straightforward.

#### 4.1 Working Memory Elements in OPS83 and Templates in CLIPS

Both OPS83 and CLIPS support frame-like data structures called working memory elements (WME) and templates, respectively. A fact in OPS83's working memory is also called a WME. Fields in a WME and fields in a template correspond to slots in a frame. CLIPS permits a template to have only one field with multiple values. An example of an OPS83 WME named *goal* from the Expert System is given below:

```
type goal = element
                (ringtype:  symbol;
                 time:      integer;
                 refno:     integer;
                 reg:       integer);
```

A corresponding template in WATE is defined as follows:

```
(deftemplate goal
  (field ringtype
    (allowed-symbols no wcr ccr))
  (field time
    (type INTEGER))
  (field refno
    (type INTEGER))
  (field reg
    (type INTEGER)))
```

where the type definition of *ringtype* indicates that the type of its value is symbol, and the allowed values that can be assigned to this field are *wcr* and *ccr*. Although the format of the structure's definitions are different, the translation of WMEs in OPS83 to templates in CLIPS is straightforward.

In addition, WMEs (facts) are uniquely numbered by OPS83 in its working memory, and each WME can be accessed by its index number. Thus, a count of the total number of facts in working memory is accessible in OPS83. In CLIPS, such facts as template facts are associated with unique generic pointers. The accessing mechanism for CLIPS facts in the fact list will be discussed in the section that describes implementation.

## 4.2 Rules in OPS83 and CLIPS

Rules in CLIPS and OPS83 have similar formats. Rules in both languages have a left-hand side (LHS) and a right-hand side (RHS). The LHS and RHS are separated by "-->" in OPS83 or by "=>" in CLIPS. The format of an OPS83 rule is as follows:

```
rule <rule-name>
{
    <LHS>
-->
    <RHS>
}.
```

In CLIPS, the rule format is as follows:

```
(defrule <rule-name>
    <LHS>
=>
    <RHS>
).
```

In both OPS83 and CLIPS, the LHS is a sequence of patterns and the RHS is a sequence of actions. A rule in OPS83 or in CLIPS can be read as follows: if all patterns on the LHS of the rule match facts in working memory, then execute the actions on the RHS of the rule.

The main difference in rules in the two languages is in the use of variables. In OPS83, a WME is bound to a pointer variable on the LHS. If the value of any field of the WME is used thereafter, it can be referenced in a manner similar to the way a field in a record is referenced in C or Pascal. For example, a portion of a rule in OPS83 might be as follows:

```
rule CheckDissipation
{
    &goal (goal);
    &reg (region
        ringtype = &goal.ringtype;
        reg = &goal.reg);
    .....
-->
    .....
};
```

where *region* is a WME, and *ringtype* and *reg* are two of its fields. Symbols preceded by "&" indicate pointer variables. In the above example, the fields *ringtype* and *reg* of the *goal* template do not need to be specified in the *goal* template pattern in order to be bound to appropriate values.

In CLIPS, however, the variable corresponding to a CLIPS fact must be bound to a value during the pattern-matching phase before it is used as a bound variable. The rule below is a CLIPS version of the previous rule where *region* is where region is a template structure. Symbols preceded by “?” indicate variables.

```
(defrule CheckDissipation
  (goal
    (ringtype ?g-rtype)
    (reg ?g-reg))
  (region
    (ringtype ?g-rtype)
    (reg ?g-reg))
  .....
=>
  .....
).
```

Another difference is the use of variables on the RHS of a rule. In OPS83, new variables used on the RHS must be defined before they are used. In CLIPS, new variables used on the RHS are implicitly declared in some types of statements, such as the bind statement. For example, a portion of a rule in OPS83 might be as follows:

```
rule CheckCoalescence
{
  .....
  &mring (moved-ring);
-->
  local  &LAT: real,
         &LONG: real;

  &LAT = &mring.modlat;
  &LONG = &mring.modlong;
  .....
};
```

In CLIPS, the corresponding rule would be written as follows:

```
(defrule CheckCoalescence
  .....
  (moved-ring
    (modlat ?m-modlat)
```

```

                (modlong ?m-modlong)
            .....
=>
                (bind ?LAT ?m-modlat)
                (bind ?LONG ?m-modlong)
            .....
        ).

```

where *moved-ring* is a WME in OPS83 and a template in CLIPS. The translation of rules in OPS83 to rules in CLIPS is straightforward.

### 4.3 Procedural Code in OPS83 and in C

OPS83 also supports a procedural language in addition to the rule-based constructs. In fact, the format of its procedural language is similar to a combination of Pascal and C. For example, a portion of a function written in OPS83 has the following format:

```

function foo (&x: out integer, &y: symbol): integer
{
    local  &a:  integer;
    .....
    &a = &a + 1;
    &x = &a + &y;
    .....
    return(&i);
};

```

where the parameter *&x* is passed by address and *&y* is passed by value. A procedure written in OPS83 has a similar format except for the heading. For example, a procedure heading might have the following format:

```

procedure foo (&x: out integer, &y: symbol)
{
    .....
};

```

Prototype declarations are similar to those in C. For example, a prototype definition in OPS83 has the following format:

```

external procedure foo(&x: out integer, &y: symbol);

```

One final comment about OPS83 and C is that the indices of array elements start with 1 in OPS83, but start with 0 in C. Again, the translation of procedural code in OPS83 to that in C can be done in a straightforward manner.

#### 4.4 Invoking the Rule System in OPS83 and in Embedded CLIPS

In OPS83, a recognize-act (RA) cycle is responsible for processing the rules. The user must write OPS83 code to implement the RA cycle. A rule is said to be "instantiated" when the patterns on its LHS are matched against a set of WMEs. This match is the recognition part of the RA cycle. OPS83 maintains a conflict set that is a set of all the instantiations that exist at a given time during the execution of a rule system. A conflict resolution routine is used to select one rule instantiation from the conflict set. This recognized instantiation can then be executed by employing the *fire* statement. The conflict set is updated after each rule firing. The user-written RA cycle should continue to loop until all instantiations have fired.

In CLIPS, the RA cycle is part of the predefined inference engine. An embedded CLIPS program is usually run by using the following sequence of procedure calls from C:

```
InitializeCLIPS();  
LoadConstructs(<fileName>);  
ResetCLIPS();  
RunCLIPS(n);
```

The *InitializeCLIPS()* function must be called prior to any other CLIPS function call and should be called only one time. The *LoadConstructs(<fileName>)* function loads facts and rules from a file. Its argument is a string that represents the name of a file. The *ResetCLIPS()* statement causes the facts to be asserted into working memory and initiates pattern matching. After CLIPS has been initialized and reset, each rule instantiation will be on the CLIPS agenda, and the *RunCLIPS(n)* statement can then be used to execute *n* instantiations on the agenda. If *n* is negative, *RunCLIPS(n)* executes all instantiations on the agenda until the agenda is empty. Unlike OPS83, CLIPS has its own predefined conflict resolution routines, which can be selected by the user. The sequence of instantiation execution is based on the selected conflict resolution strategy.

## 5.0 IMPLEMENTATION

This section discusses issues that were addressed during the implementation phase. The general approach was first to modularize the system and then to convert the system modules from OPS83 to C and CLIPS. Finally, the Explanation Component was reimplemented in a more declarative form. The basic logic of the Expert System was not changed. During discussion with Matthew Lybanon and Sarah Peckinpaugh of NRL, obsolete variables and statements were identified. Variables were declared in the new code on an as-needed basis. The source code of WATE was delivered in electronic form.

### 5.1 Coding and Naming Conventions

#### 5.1.1 C Code

The following coding and naming conventions apply to C code and to CLIPS:

- Constant Declaration — constant names are declared with all capitalized letters.
- Type Declaration — type names are declared with only the first letter capitalized.

- Variable Declaration — variable names are declared with lower case letters except for the file-name variables and the rule-name variables, which are declared with all capitalized letters. If a composite word is desired, then the variable name is named as a combination of two or more words without hyphens, e.g., *stepdays*.

- Function Declaration — functions defined in WATE are named with the first letter of each word capitalized.

- Global Variable — global variables are declared in the *globals.h* and *explan.h* files.

- Prototype Declaration — prototypes of user-defined functions called from the CLIPS environment are declared in the *globals.h* file.

- Comment Statements — each file contains introductory comment statements that describe the content and the purpose of that file. The functions provided by each routine are described as well. The location of each external routine is also described for each prototype declaration.

### 5.1.2 CLIPS Code

- Rule Naming — rules should be named as meaningfully as possible. For example, names like *rule1* should be avoided. Use *CCREstimateMotion* instead.

- User-Defined Function Naming — user-defined functions should be named in a meaningful manner with lower case letters, e.g., *get-input*.

- Comment Statements — each construct in CLIPS, including *deffacts*, *deftemplate*, and *defrule*, should have a comment, giving its purpose and properties.

## 5.2 Embedding CLIPS in C

The rule systems of both the Eddy Component and the Explanation Component are implemented in CLIPS. This section discusses several general issues regarding embedding CLIPS in C, but the CLIPS manual (Giarrantano 1990) should be consulted for more details.

### 5.2.1 CLIPS Library

When external functions defined by CLIPS are used (to be called from either C or CLIPS), the header file *clips.h* must be included in the declaration section.

### 5.2.2 User-Defined Functions

An external function (written in C) can be defined and employed in CLIPS as an ordinary CLIPS function. A CLIPS function *DefineFunction* statement must be used to define the function name used in the CLIPS environment, the data type returned by the function, and the link to the external function. For example, a C function named *GetLcntgs* that returns the value of a global integer C variable *lcntgs* is given below. The function will be accessed in CLIPS using the name *get-lcntgs*.

```
DefineFunction("get-lcntgs", 'i', GetLcntgs, "GetLcntgs");
```

```
int GetLcntgs()
```

```
{
```

```

        return(lcntgs);
    }.

```

The first argument, *get-lcntgs*, is the name of the function that can be used in CLIPS as a generic function. The second argument, *i*, indicates that this function returns an integer. The third argument defines the link between CLIPS and this function. The last argument specifies the actual function name defined in the external routine. The third and fourth arguments are not necessarily the same. The declarations of all user-defined functions employed in WATE are defined in the *wate-stubs.c* file. All of the user-defined functions are defined in the *returns.c* file except the geometry routines, which are defined in *mathe.c*, and the routines used to access region information, which are defined in *region.c*.

### 5.2.2.1 Parameters Passed by Value from CLIPS to C

Parameters can be passed by value from CLIPS to user-defined functions. However, the passed values must be transformed to the appropriate data type by employing functions, such as *RtnLong(n)*, *RtnDouble(n)*, and *RtnString(n)*, where *n* indicates the position of the parameter in the parameter list. For example, a user-defined function that returns the value of the *n*th element of an array could be declared in a *DefineFunction* statement and defined as follows:

```

DefineFunction("get-gsdist-value", 'd', GetGSdistValue, "GetGSdistValue");
double GetGSdistValue()
{
    int    index;
    index = (int) RtnLong(1);
    return(gsdist[index]);
}.

```

where *gsdist* is a global variable of array of double type. Note that this function will be invoked from CLIPS with one argument (the index of the element to be retrieved) even though the C function has no arguments. CLIPS provides error-checking routines that can be called from the C function to make sure that the function has been invoked with the correct number and type of arguments.

### 5.2.2.2 Accessing CLIPS Facts from External Functions

CLIPS facts can be accessed by external functions via pointers. For example, function *GetNextFact(ptr)* returns a generic pointer (*ptr*) pointing to one of the following: (1) the first fact of the fact list if the value of *ptr* is NULL, (2) the fact following the fact pointed to by *ptr* if *ptr* is pointing to a fact of the fact list, or (3) NULL if *ptr* is pointing to the last fact of the fact list.

The function *GetMFValue(ptr, n)* returns a generic pointer pointing to the value located at the *n*th position of a multifield fact pointed to by *ptr*. Please note that if a template fact is accessed, the sequence of the fields defined in the *deftemplate* construct (*deftemplate* will be discussed in the next section) determines the position indicated by *n*. Therefore, if a modification is made in the template definitions in terms of the sequence of fields, all the values returned by the *GetMFValue* function will be influenced. Currently, the *GetMFValue* function has been employed in the *eddies.c* and *final-output.c* files.

The functions *ValueToLong(ptr)*, *ValueToDouble(ptr)*, and *ValueToString(ptr)*, return values of type long, double, and string pointer, respectively. These functions were used extensively to access CLIPS facts from external functions. The following example uses functions mentioned above to search through the CLIPS fact list and locate a *goal* template fact.

```
factptr = GetNextFact(NULL);
fact = ValueToString(GetMFValue(factptr, 1))
while (strcmp(fact, "goal") != 0)
{
    factptr = GetNextFact(factptr);
    fact = ValueToString(GetMFValue(factptr, 1));
}
```

After the above statements have been executed, the value of each field defined in the *goal* template can also be obtained. For example, the following statement can be used to return the value of the field named *refno* as an integer type:

```
(int) ValueToLong(GetMFValue(factptr, 4)).
```

### 5.3 Running WATE

WATE supports two running modes: integrated mode and stand-alone mode. The default running mode is integrated mode unless the user explicitly types in *wate -w* at the command line. The integrated mode of WATE is invoked from the SAMAS menu.

### 5.4 Initialization Component

The Initialization Component sets up the environment for CLIPS, PV Wave, and the Gulf Stream prediction system. This component invokes the following modules: *Init Globals*, *Get User Options*, *Setup CLIPS*, *Setup PV Wave*, and *Setup GS*. The *Init Globals* module initializes the following global variables:

- *stepdays* — the number of days in a time step (currently initialized to seven).
- *cyclim* — the limit of the number of the prediction cycles (currently set to 100).
- *gswidth* — the default distance between the GS north and south walls (currently set to 100.0).
- *ugsray*, *lgsray*, and *mgsray* — arrays of upper (north), lower (south), and middle GS points (currently all of the array elements are initialized to 0.0).

The *Get User Options* module will be discussed in the next section. The *Setup CLIPS* module invokes the following submodules:

- *Init CLIPS* — performs initialization of CLIPS.
- *Load Constructs* — loads prediction rules stored in the *wcrrules.clp* and *crrules.clp* files, and explanation rules stored in the *explain.clp* file.
- *Reset CLIPS* — resets the CLIPS environment.

- *Read Input* — reads data from the parameter file, region data file, and eddy data file.

The *Setup PV Wave* module interfaces with an external PV-Wave display module by calling the following routines:

*InitWave()* — defined in *mapping.c* file.

*ExpertMap()* — defined in *mapping.c* file.

*DrawGS()* — defined in *mapping.c* file.

*wavewait()* — defined in *mapping.c* file.

The functionality of the above routines is described in Peckinpaugh (1993).

The *Setup GS* module interfaces with the external Gulf Stream prediction module, in which the Gulf Stream north and south walls are formed by calling the following external routines:

*ModesInit()* — defined in *modes.c* file.

*CreateSW()* — defined in *modes.c* file.

*ReformGS()* — defined in *modes.c* file.

*ReadGS()* — defined in *modes.c* file.

*GSfillEnds()* — defined in *modes.c* file.

*fillGS2()* — defined in *mathe.c* file.

*nomGS2()* — defined in *mathe.c* file.

*normalize()* — defined in *nrmlz.c* file.

The functionality of each routine in the Gulf Stream Prediction Module is described in Peckinpaugh (1993).

## 5.5 Get User Options Module

The *Get User Options* Module has two versions. If WATE is running in stand-alone mode, the User Options Window (see Fig. 4) will be displayed, including the choice of using PV Wave to interpolate the predictions, the UGS data file, the eddy data file, and the number of days for each prediction cycle. If PV Wave is chosen, the user can further decide how to draw the images of rings, the Gulf Stream, and the map area (with or without grids and/or region limits). Although the user can still select drawing preferences, if the *NO* for PV Wave option is selected by the user (or by default), the selections for the drawing preferences will be ignored by WATE.

The User Options Window has an error detection facility if the UGS data file and/or eddy data file either is not given or cannot be opened. Users will be notified by an error message when they try to start the prediction by clicking the *OK* button if one of these errors exists.

The default value of the UGS data file is *nominal*. The default number of days for each step of prediction is seven. Clicking on the *Help* button invokes a pop-up window with help text for the User Options Window. The *Quit* button provides a way to quit the WATE system.

Fig. 4 — The User Options Window

If WATE is running in integrated mode from SAMAS 1.2, all the options are presented through a command line interface.

## 5.6 Eddy Component

The main module of the Eddy Component is implemented in C, and its rule system is implemented in CLIPS. An outer loop in the main module controls the reasoning process of the rule system based on a time-step. The outer loop contains two inner loops for processing WCRs and CCRs. The following pseudocode illustrates the control of the prediction process:

```

quit <- false
time <- 0
repeat (outer loop)
  for region <- 1 to 9 do (WCR inner loop)
    assert template fact goal for wcr
    RunCLIPS(-1)
  for region <- 1 to 9 do (CCR inner loop)
    assert template fact goal for ccr
    RunCLIPS(-1)
  time <- time + time step
until the number of prediction cycles reaches limit or user enters quit.

```

At the first iteration of the WCR inner loop, the following *goal* template is asserted into CLIPS to cause instantiation of appropriate rules at time frame of 0 in region 1:

```
(goal (ringtype wcr)
      (time 0)
      (refno 0)
      (reg 1)).
```

The rule system begins to reason about WCRs that have the time stamp of 0 and are located in region 1. A *refno* value of 0 is used to indicate that no ring has been selected for processing. In the CCR loop, the same logic is applied with the *ringtype* field set to *ccr*. Within each inner loop, the statement *RunCLIPS(-1)* causes all instantiations to execute.

The knowledge structure of the rule system of the Eddy Component is defined by four templates: the *goal* template, the *ring* template, the *region* template, and the *moved-ring* template. The original rules of the Expert System were readily converted from OPS83 to CLIPS with a few modifications of the patterns on the LHS and of the external function calls on the RHS. In addition, actions were added at the end of the RHS of each prediction rule to record the rule firing and values of key variables as CLIPS facts. This part of each rule is purely for explanation. For example, one of the WCR prediction rules is as follows:

```
(defrule WCRStrongInteraction ;;Is the WCR-GS interaction strong?
  (goal
    (ringtype wcr)
    (reg ?g-reg)
    (refno ?g-refno)
    (time ?g-time))

  (region
    (ringtype wcr)
    (reg ?g-reg)
    (bkpt1 ?bk1))

  ?m <- (moved-ring
    (ringtype wcr)
    (reg ?g-reg)
    (inter nil)
    (ratio ?m-ratio&:(> ?m-ratio ?bk1)))

  ?rfr <- (rule-fire-record
    (ringtype wcr)
    (refno ?g-refno)
    (time ?g-time)
    (rules-fired $?rules))

  =>
```

```

(modify ?m (regime 1)
      (inter strong))
;; below is for explanation only
(modify ?rfr (rules-fired $?rules WCRStrongInteraction))
(assert (values-for-explanation
      (rule-name WCRStrongInteraction)
      (ringtype wcr)
      (refno ?g-refno)
      (time ?g-time)
      (var-val bkpt1 ?bk1))))

```

The templates *rule-fire-record* and *values-for-explanation* will be discussed in the next section.

The Eddy Component accesses Gulf Stream data structures by calling the following routines:

*ModeGS\_move()* — defined in *modes.c*  
*CreateSW()* — defined in *modes.c*  
*GSfillEnds()* — defined in *modes.c*

The PV-Wave display module is accessed by calling the following routines:

*Refresh()* — defined in *mapping.c*  
*DrawEddy()* — defined in *mapping.c*  
*EraseEddy()* — defined in *mapping.c*  
*DrawGS()* — defined in *mapping.c*

The rule system of the Eddy Component interfaces with external Geometry Routines by the following user-defined CLIPS functions:

- *PreRtoGS2()* — works as an interface between the rule system and the external routine *RtoGS2()* that is defined in *mathe.c*.
- *Prenringstep()* — works as an interface between the rule system and the external routine *ringstep()* that is defined in *mathe.c*.
- *Preadjustringstep()* — works as an interface between the rule system and the external routine *adjustringstep()* defined in *mathe.c*.

All of the user-defined CLIPS functions are defined in *returns.c*.

## 5.7 Explanation Component

The Explanation Component contains two main modules, the Explanation Driver and the Presentation Module. The Explanation Driver is implemented in C, and the Presentation Module in CLIPS. The user is provided with the following options: (1) explain the prediction of a single ring, (2) explain all eddy movement, or (3) quit the explanation menu. If the explanation of the prediction

of either a single ring or all eddies is selected, the user is given the further choice of a rule-trace explanation or a summary of the reasoning process. Depending on the type of explanation requested by the user, the Explanation Driver asserts appropriate facts in the CLIPS working memory to activate the appropriate rules of the Presentation Module. The templates for structuring the explanation that were previously represented in C were put on the RHS of each rule of the Presentation Module. The desired value slots in the template are filled by the variables whose values were bound at the LHS of that rule. The Presentation Module then prints out the requested explanation, either rule trace or summary, of the prediction process for the desired ring(s). The explanation rules are defined in *explain.clp*.

The knowledge structure of the Explanation Component contains two templates: the *rule-fire-record* template and the *values-for-explanation* template. Each ring at each time frame has one *rule-fire-record* template fact associated with it. For every single rule firing, one *values-for-explanation* template fact is asserted. These two template facts are linked by the indexing fields, including the ring type, reference number, and time stamp. For example, a *rule-fire-record* template fact with the following slot values

```
(rule-fire-record
  (ringtype wcr)
  (refno 1)
  (time 0)
  (rules-fired WCREstimateMotion WCRGetInteraction))
```

can be translated to English as follows: the rules that fired to predict the movement of WCR 1 at time stamp 0 are *WCREstimateMotion* and *WCRGetInteraction*. Then, the following *values-for-explanation* template fact can be accessed to obtain appropriate values for explaining the status of WCR 1 as the *WCREstimateMotion* rule fired:

```
(values-for-explanation
  (rule-name WCREstimateMotion)
  (ringtype wcr)
  (refno 1)
  (time 0)
  (var-val var1 val1 var2 val2 ...)).
```

## 5.8 Final Output Component

This component was converted from OPS83 to C. It provides the user with options for storing the final positions of the Gulf Stream north wall, the Gulf Stream south wall, and/or the eddies with active status into external files with a name specified by the user. The Final Output Component returns control to the main module if WATE is running in stand-alone mode. The User Options Window then becomes active. The user is allowed to run the system again until he or she quits the system from either the User Options Window or the command line.

## 6.0 SYSTEM TESTING

WATE was tested for the adequacy of the predictions of eddy movement, its explanation capability, and its reasoning process. The final results that were obtained from system testing are considered sufficient to pass the requirements specified for input, output, and the user interface.

### 6.1 Testing Criteria

The criterion for passing each test case is that WATE should generate the same prediction at each time step as the Expert System, given the same input. The explanation produced by WATE, both rule trace and summary, should have the same content as the Expert System, given the same input. All test cases were tested over a period of 28 days with a time step of 7 days except for the explanation testing, which was run over a 14-day period.

### 6.2 Test Cases

WATE was tested against the same test data sets that were used to test the previous versions of the oceanographic expert system. The test cases are described in Appendix B. The first 10 test cases use the NRL DART (Data Assimilation Research and Transition program) data sets (Lybanon 1990). Files are named *dartyydd.gs* or *dartyydd.eddy*, where *yy* indicates the last two digits of the year and *ddd* stands for the day number of that year. The DART GS files are used with the DART Eddy files for the same time period. The last three test cases use artificial data constructed to exercise all of the rules of the expert system (Bridges and Lybanon 1993). These test cases use the files *ugs.dat*, *wcr.dat*, *ccr.dat*, and *eddies.dat* to test the explanation capability and the reasoning process of WATE. These artificial data files were designed to test the performance of the system in each of the nine regions for each type of eddy. The locations and size of WCRs cause each of the constraint rules for warm-core eddies to fire. Eddies were also included that should dissipate during the prediction period, coalesce with the GS, and move from one region to another. The input files, the region data file, and the parameter data file for all test cases were delivered in electronic form.

### 6.3 Test Results

WATE passed all test cases successfully. The results, delivered in electronic form to NRL, show that the old and new versions of WATE give identical results.

## 7.0 EVALUATION AND CONCLUSIONS

WATE was successfully converted from OPS83 to C and CLIPS. This conversion will facilitate the incorporation of WATE into SAMAS 1.2, which will eventually be embedded in TESS(3). The source code of WATE is fully commented with a straightforward layout. By modularizing the system, the control structure of WATE is easier to understand and maintain. The user interface was also improved by employing a window interface. The most valuable reconstruction was in the Explanation Component in that it is implemented by a rule system. Since the information about the reasoning process of the Eddy Component is asserted into working memory as a set of facts, instead of recording activation of the agenda and the desired variable-value pairs by external C routines each time a rule fired, the Explanation Component can be viewed as more independent of the Eddy

Component. Furthermore, the explicit implementation of the Explanation Component makes it easier to maintain and enhance (Bridges and Lybanon 1993). Overall, this project has made the Expert System more readable, understandable, and easier to maintain, and has made the system portable to the TESS environment.

## **8.0 ACKNOWLEDGMENTS**

This work was sponsored by the Space and Naval Warfare System Command, CDR D. Markham, Program Manager, under Program Element Number 0603207N.

## **9.0 REFERENCES**

- Bridges, S. M. and M. Lybanon, "Adding Explanation Capability to a Knowledge-Based System: A Case Study," Applications of Artificial Intelligence 1993 Conference on Knowledge-Based System in Aerospace and Industry, April 13-15, 1993, Orlando, FL, SPIE Vol. 1963, pp. 40-49.
- Forgy, C. L., "The OPS83 User's Manual System Version 2.2," Production Systems Technologies, Inc., Pittsburgh, PA, 1986.
- Giarrantano, J. C., "CLIPS Manual and References Guide," NASA Lyndon B. Johnson Space Center, Information Systems Directorate, Software Technology Branch, 1990.
- Lybanon, M., "Oceanographic Expert System: Potential for TESS(3) Applications," Naval Research Laboratory, Stennis Space Center, MS, NOARL Tech. Note 286, 1992.
- Lybanon, M., "Oceanographic Expert System Validation Using GOAP Mesoscale Products and Gulfcast/Dart Validation Test Data," Naval Research Laboratory, Stennis Space Center, MS, NOARL Report 5, 1990.
- Peckinpaugh, S. H., "Documentation for the Semi-Automated Mesoscale Analysis System 1.2," Naval Research Laboratory, Stennis Space Center, MS, in publication.
- Thomason, M. G. and R. E. Blake, "Development of An Expert System for Interpretation of Oceanographic Images," Naval Research Laboratory, Stennis Space Center, MS, NORDA Report 148, 1986.



## Appendix A

### FORMAT OF INPUT FILES

#### FORMAT FOR INPUT FILE OF WCRs AND CCRs

The file contains one line for the total number of eddies, followed by one line for each eddy, containing the following values:

latitude longitude radius wc-code src-code,

where wc-code is either "W" or "C," and src-code is either "EE" or "ES."

#### FORMAT FOR PARAMETER FILE

The parameter file is named *parms.dat*. The file contains a set of parameters for each region for each type of eddy. The parameters for WCRs for all regions are given first, followed by parameters for the CCRs for each region. For each region the WCR parameters have one line containing the region number, followed by one line for each direction (1-16) containing the following values:

direction lat-fac-1 long-fac-1 lat-fac-2 long-fac-2,

where lat-fac-*n* is the latitude adjust factor for regime *n*, and long-fac-*n* is the longitude adjust factor for regime *n*.

The CCR parameters have one line containing the region number followed by one line for each direction (1-16) containing the following values:

direction h-1 h-2 f-d-2 h-3 f-d-3 h-4 f-d-4,

where h-*n* is the heading value for regime *n*, and f-d-*n* is the final direction for regime *n*.

#### FORMAT FOR REGION FILE

The input file for regions, *regions.dat*, contains the following values for each line:

region-name speed direction heading decay<sup>1</sup> min-rad decay<sup>2</sup>,

where speed is in centimeters per second, heading is a compass heading, decay<sup>1</sup> is the decay value without GS interaction, mn-rad is the minimum radius for no coalesce with GS, and decay<sup>2</sup> is the decay value with GS interaction. For WCR regions the values also contain break-point values, bk1 to bk3, that are used to compute the degree of GS interaction. For CCR regions, the values also contain break-point values, bk1 to bk4, that are used to determine the path of rings that have encountered the GS and are in a looping motion.

## Appendix B

### TEST CASES

This appendix describes 13 test cases. The test data named *dartxxxx.gs* and *dartxxxx.eddy* were used as pairs of UGS data files and eddy data files for each test case. There are 10 test cases with a *dart* heading. The test data named *ugs.dat* was used as the UGS data file tested with the test data named *wcr.dat*, *ccr.dat*, and *eddies.dat*, all of which form the other three test cases. The *dart* test cases were adopted from Lybanon (1990). The rest of test cases were adopted from Bridges and Lybanon (1993). The files described and the input files of parameters (*parms.dat*) and region data (*regions.dat*) are available in electronic form in the directory *testing* in the file *test.data*.

#### dart85223.gs

337  
36.7647 -74.0235 O  
36.8529 -74.0235 O  
36.8529 -73.9137 O  
36.8529 -73.8039 O  
36.9412 -73.8039 O  
36.9412 -73.6941 O  
37.0294 -73.6941 O  
37.0294 -73.5843 O  
37.1176 -73.5843 O  
37.1176 -73.4745 O  
37.2059 -73.4745 O  
37.2059 -73.3647 O  
37.2941 -73.3647 O  
37.2941 -73.2549 O  
37.3824 -73.2549 O  
37.3824 -73.1451 O  
37.4706 -73.1451 O  
37.4706 -73.0353 O  
37.5588 -73.0353 O  
37.6471 -73.0353 O  
37.6471 -72.9255 O  
37.7353 -72.9255 O  
37.8235 -72.9255 O  
37.9118 -72.9255 O  
37.9118 -72.8157 O  
38.0000 -72.8157 O  
38.0882 -72.8157 O  
38.0882 -72.7059 O

#### dart85223.gs

38.1765 -72.7059 O  
38.2647 -72.7059 O  
38.2647 -72.5961 O  
38.3529 -72.5961 O  
38.3529 -72.4863 O  
38.4412 -72.4863 O  
38.4412 -72.3765 O  
38.5294 -72.3765 O  
38.5294 -72.2667 O  
38.6176 -72.1569 O  
38.6176 -72.0471 O  
38.6176 -71.9373 O  
38.6176 -71.8275 O  
38.5294 -71.8275 O  
38.5294 -71.7177 O  
38.4412 -71.7177 O  
38.3529 -71.6078 O  
38.2647 -71.4980 O  
38.1765 -71.4980 O  
38.1765 -71.3882 O  
38.0882 -71.2784 O  
38.0882 -71.1686 O  
38.0000 -71.1686 O  
38.0000 -71.0588 O  
37.9118 -71.0588 O  
37.9118 -70.9490 O  
37.9118 -70.8392 O  
37.9118 -70.7294 O  
37.8235 -70.7294 O

#### dart85223.gs

37.8235 -70.6196 O  
37.8235 -70.5098 O  
37.9118 -70.5098 O  
37.9118 -70.4000 O  
37.9118 -70.2902 O  
38.0000 -70.1804 O  
38.0000 -70.0706 O  
38.0882 -69.9608 O  
38.0882 -69.8510 O  
38.1765 -69.8510 O  
38.1765 -69.7412 O  
38.2647 -69.6314 O  
38.2647 -69.5216 O  
38.3529 -69.5216 O  
38.3529 -69.4118 O  
38.4412 -69.4118 O  
38.4412 -69.3020 O  
38.4412 -69.1922 O  
38.5294 -69.1922 O  
38.5294 -69.0824 O  
38.6176 -68.9725 O  
38.6176 -68.8627 O  
38.7059 -68.8627 O  
38.7059 -68.7529 O  
38.7059 -68.6431 O  
38.7941 -68.5333 O  
38.7941 -68.4235 O  
38.7941 -68.3137 O  
38.7941 -68.2039 O

dart85223.gs

38.7941 -68.0941 O  
 38.7941 -67.9843 O  
 38.7059 -67.9843 O  
 38.7059 -67.8745 O  
 38.6176 -67.8745 O  
 38.5294 -67.7647 O  
 38.4412 -67.7647 O  
 38.4412 -67.6549 O  
 38.3529 -67.6549 O  
 38.2647 -67.6549 O  
 38.2647 -67.5451 O  
 38.1765 -67.5451 O  
 38.0882 -67.5451 O  
 38.0882 -67.4353 O  
 38.0000 -67.4353 O  
 37.9118 -67.3255 O  
 37.8235 -67.2157 O  
 37.8235 -67.1059 O  
 37.7353 -67.1059 O  
 37.7353 -66.9961 O  
 37.7353 -66.8863 O  
 37.7353 -66.7765 O  
 37.7353 -66.6667 O  
 37.7353 -66.5569 O  
 37.8235 -66.4471 O  
 37.8235 -66.3373 O  
 37.9118 -66.2274 O  
 37.9118 -66.1176 O  
 38.0000 -66.0078 O  
 38.0000 -65.8980 O  
 38.0882 -65.7882 O  
 38.0882 -65.6784 O  
 38.1765 -65.6784 O  
 38.1765 -65.5686 O  
 38.2647 -65.4588 O  
 38.2647 -65.3490 O  
 38.3529 -65.3490 O  
 38.3529 -65.2392 O  
 38.3529 -65.1294 O  
 38.4412 -65.1294 O  
 38.4412 -65.0196 O  
 38.4412 -64.9098 O  
 38.5294 -64.8000 O  
 38.5294 -64.6902 O  
 38.6176 -64.5804 O  
 38.6176 -64.4706 O  
 38.6176 -64.3608 O  
 38.6176 -64.2510 O  
 38.6176 -64.1412 O  
 38.5294 -64.1412 O

dart85223.gs

38.5294 -64.0314 O  
 38.5294 -63.9216 O  
 38.4412 -63.9216 O  
 38.4412 -63.8118 O  
 38.3529 -63.8118 O  
 38.3529 -63.7020 O  
 38.2647 -63.7020 O  
 38.2647 -63.5922 O  
 38.1765 -63.5922 O  
 38.1765 -63.4824 O  
 38.1765 -63.3726 O  
 38.0882 -63.2627 O  
 38.0882 -63.1529 O  
 38.0882 -63.0431 O  
 38.0882 -62.9333 O  
 38.0882 -62.8235 O  
 38.0882 -62.7137 O  
 38.0882 -62.6039 O  
 38.0882 -62.4941 O  
 38.0882 -62.3843 O  
 38.0882 -62.2745 O  
 38.0882 -62.1647 O  
 38.0882 -62.0549 O  
 38.0882 -61.9451 O  
 38.0882 -61.8353 O  
 38.0882 -61.7255 O  
 38.0882 -61.6157 O  
 38.0882 -61.5059 O  
 38.0882 -61.3961 O  
 38.0882 -61.2863 O  
 38.0882 -61.1765 O  
 38.0882 -61.0667 O  
 38.1765 -61.0667 O  
 38.2647 -61.0667 O  
 38.3529 -61.0667 O  
 38.4412 -61.0667 O  
 38.5294 -61.0667 O  
 38.6176 -61.1765 O  
 38.7059 -61.1765 O  
 38.7941 -61.1765 O  
 38.7941 -61.2863 O  
 38.8824 -61.2863 O  
 38.9706 -61.3961 O  
 39.0588 -61.3961 O  
 39.0588 -61.5059 O  
 39.1471 -61.5059 O  
 39.2353 -61.6157 O  
 39.3235 -61.6157 O  
 39.4118 -61.7255 O  
 39.5000 -61.7255 O

dart85223.gs

39.5882 -61.7255 O  
 39.6765 -61.7255 O  
 39.7647 -61.7255 O  
 39.7647 -61.6157 O  
 39.8529 -61.6157 O  
 39.9412 -61.5059 O  
 39.9412 -61.3961 O  
 40.0294 -61.3961 O  
 40.0294 -61.2863 O  
 40.0294 -61.1765 O  
 40.1176 -61.1765 O  
 40.1176 -61.0667 O  
 40.2059 -60.9569 O  
 40.2059 -60.8471 O  
 40.2059 -60.7373 O  
 40.2059 -60.6274 O  
 40.2059 -60.5176 O  
 40.2059 -60.4078 O  
 40.2059 -60.2980 O  
 40.2059 -60.1882 O  
 40.1176 -60.1882 O  
 40.1176 -60.0784 O  
 40.1176 -59.9686 O  
 40.0294 -59.9686 O  
 40.0294 -59.8588 O  
 39.9412 -59.8588 O  
 39.9412 -59.7490 O  
 39.8529 -59.7490 O  
 39.8529 -59.6392 O  
 39.7647 -59.6392 O  
 39.7647 -59.5294 O  
 39.6765 -59.5294 O  
 39.5882 -59.4196 O  
 39.5882 -59.3098 O  
 39.5000 -59.3098 O  
 39.5000 -59.2000 O  
 39.4118 -59.2000 O  
 39.4118 -59.0902 O  
 39.4118 -58.9804 O  
 39.4118 -58.8706 O  
 39.4118 -58.7608 O  
 39.3235 -58.7608 O  
 39.3235 -58.6510 O  
 39.3235 -58.5412 O  
 39.3235 -58.4314 O  
 39.3235 -58.3216 O  
 39.3235 -58.2118 O  
 39.2353 -58.1020 O  
 39.2353 -57.9922 O  
 39.2353 -57.8824 O

<u>dart85223.gs</u>	<u>dart85223.gs</u>	<u>dart85223.eddy</u>
39.2353 -57.7725 O	41.9706 -57.8824 O	8
39.2353 -57.6627 O	41.9706 -57.7725 O	39.8 -66.9 80.0 W D1
39.2353 -57.5529 O	41.9706 -57.6627 O	39.4 -69.5 75.0 W D2
39.3235 -57.5529 O	41.9706 -57.5529 O	40.4 -64.2 90.0 W D3
39.3235 -57.4431 O	41.9706 -57.4431 O	41.0 -54.2 70.0 W D4
39.4118 -57.4431 O	41.8824 -57.4431 O	40.6 -62.4 80.0 W D8
39.4118 -57.3333 O	41.8824 -57.3333 O	35.3 -68.2 80.0 C D5
39.5000 -57.3333 O	41.7941 -57.2235 O	36.2 -60.7 85.0 C D6
39.5882 -57.3333 O	41.7059 -57.1137 O	37.0 -58.5 70.0 C D7
39.6765 -57.3333 O	41.6176 -57.0039 O	
39.6765 -57.4431 O	41.6176 -56.8941 O	
39.7647 -57.4431 O	41.5294 -56.8941 O	
39.8529 -57.5529 O	41.4412 -56.7843 O	
39.9412 -57.6627 O	41.3529 -56.7843 O	
40.0294 -57.6627 O	41.3529 -56.6745 O	
40.1176 -57.7725 O	41.2647 -56.6745 O	
40.2059 -57.8824 O	41.1765 -56.5647 O	
40.2059 -57.9922 O	41.0882 -56.4549 O	
40.2941 -57.9922 O	41.0000 -56.4549 O	
40.2941 -58.1020 O	40.9118 -56.3451 O	
40.3824 -58.1020 O	40.8235 -56.2353 O	
40.3824 -58.2118 O	40.7353 -56.2353 O	
40.4706 -58.2118 O	40.7353 -56.1255 O	
40.5588 -58.3216 O	40.6471 -56.1255 O	
40.5588 -58.4314 O	40.5588 -56.0157 O	
40.6471 -58.4314 O	40.4706 -56.0157 O	
40.7353 -58.4314 O	40.4706 -55.9059 O	
40.7353 -58.5412 O	40.3824 -55.9059 O	
40.8235 -58.5412 O	40.2941 -55.7961 O	
40.8235 -58.6510 O	40.2059 -55.7961 O	
40.9118 -58.6510 O	40.2059 -55.6863 O	
41.0000 -58.6510 O	40.1176 -55.6863 O	
41.0882 -58.6510 O	40.0294 -55.6863 O	
41.0882 -58.7608 O	39.9412 -55.5765 O	
41.1765 -58.7608 O	39.8529 -55.5765 O	
41.2647 -58.7608 O	39.8529 -55.4667 O	
41.3529 -58.7608 O	39.7647 -55.4667 O	
41.3529 -58.8706 O	39.6765 -55.3569 O	
41.4412 -58.8706 O	39.6765 -55.2471 O	
41.5294 -58.8706 O	39.5882 -55.2471 O	
41.6176 -58.7608 O	39.5882 -55.1373 O	
41.7059 -58.7608 O	39.5000 -55.0275 O	
41.7941 -58.6510 O	39.5000 -54.9176 O	
41.7941 -58.5412 O	39.4118 -54.8078 O	
41.8824 -58.5412 O	39.4118 -54.6980 O	
41.8824 -58.4314 O	39.4118 -54.5882 O	
41.9706 -58.3216 O	39.4118 -54.3686 O	
41.9706 -58.2118 O	39.4118 -54.2588 O	
41.9706 -58.1020 O	39.4118 -54.1490 O	
41.9706 -57.9922 O	39.4118 -54.0392 O	
	39.4118 -53.9294 O	



## Appendix C

### TEST RESULTS

This appendix describes the test results of the 13 test cases from Lybanon (1990). The test cases *dartxxxx.gs* and *dartxxxx.eddy* contributed to test results:

- (1) ES — *dartxxxx* generated by the original Expert System.
- (2) *wate* — *dartxxxx* generated by WATE 1.0.

Bridges (1993). The GS data file tested with test cases *wcr.dat*, *ccr.dat*, and *eddies.dat* was *ugs.dat*, which together contributed test results:

- (1) ES-*wcr*, ES-*ccr*, and ES—*eddies* generated by the original Expert System.
- (2) *wate-wcr*, *wate-ccr*, and *wate*—*eddies* generated by WATE 1.0.

Please note that the difference between the sequence of eddies appeared in each result generated by the original Expert System and that generated by WATE was due to the different conflict resolution selected by CLIPS and OPS83, i.e., the decision of which rule on the conflict set should fire next. However, this will not affect the correctness of test results. For convenience, test results generated by two systems are listed correspondingly. All of these test cases were run for a 28-day period.

In addition, the validation of explanations generated by WATE was done by employing the artificial test data set found in the files *wcr.dat* and *ccr.dat*, with *ugs.dat* as the UGS data. This artificial eddy data forces all of the constraint rules to fire. Again, if two rings existed in the same region, OPS83 may reason about one of them first, and CLIPS may reason about the other first. Sample test results are delivered in electronic form in the directory *testing* in the file *test.results*. The explanations generated by two systems were over a 14-day period. The test results of the explanation component are delivered in electronic form in the directory *testing* in the files *result.ES* (explanation by the original Expert System) and *result.wate* (explanation by WATE).



## Appendix D

### DESCRIPTION OF FILES CONTAINING SOURCE CODE

The source code is available in electronic form in the directory *source*.

#### A. C FILES

globals.h	Header file for the global variables used in WATE 1.0.
explan.h	Defines constants and structures for menu choices for explanation.
pvdummy.h	Contains prototype declarations for the PV-Wave routines that will be used for graphical displays. This file will be eliminated when the code is integrated with the actual PV-Wave routines.
wate-interface.h	User interface object and function declarations.
wate.c	Contains the main function of WATE and event callback functions for the User Options Window if it is running in stand-alone mode.
initialize.c	Contains the following functions: <i>InitGlobals</i> , <i>GetUserOpt</i> , <i>SetupCLIPS</i> , <i>SetupPVWave</i> , and <i>SetupGS</i> . The <i>SetupCLIPS</i> function logically contains five functions, including <i>InitializeCLIPS</i> , <i>LoadConstructs</i> , <i>ResetCLIPS</i> , <i>ReadParms</i> , and <i>ReadEddies</i> .
eddiess.c	Contains the module of the Eddy Component, <i>start()</i> . Please note that the sequence of the fields in the template definitions (defined in <i>k-stru.clp</i> ) DOES matter, since it influences the value obtained by the function <i>GetMFValue</i> . Therefore, if the field ordering in the template definition is changed, this file may also need to be changed.
explan.c	Contains functions definitions for the Explanation Driver Module of the Explanation Component.
final-output.c	Prints the final status of north wall of the Gulf Stream, south wall of the Gulf Stream, and all eddies to three external files. This is a translation of the file <i>FinalOutput.ops</i> in OPS.

---

<code>pvdummy.c</code>	Contains function definition stubs for the PV-Wave routines that will be used for graphical displays. This file will be eliminated when the code is integrated with the actual PV-Wave routines.
<code>readinput.c</code>	User-defined functions for reading values for templates, parms, and eddy info.
<code>regions.c</code>	Contains the following functions: <i>TestReg</i> . This function was in <i>regions.ops</i> . The other function, <i>In_GS</i> , which was also in <i>regions.ops</i> , is now in <i>returns.c</i> , since it has been changed to return some value back to CLIPS.
<code>returns.c</code>	User-defined functions for returning values to CLIPS from external functions.
<code>wate-interface-stubs.c</code>	User interface object initialization functions.
<code>wate-stubs.c</code>	Gives the CLIPS definition of user-defined C functions that can be called from CLIPS.

## B. CLIPS FILES

<code>ccrrules.clp</code>	Contains the rules for predicting the movement of cold-core eddies. Templates used in these rules are defined in <i>k-stru.clp</i> . Please note that this rule file must physically reside in the directory you specify during the <i>get-user-option</i> session. The start procedure in module <i>eddies</i> cycles through all regions asserting <i>goal</i> with <i>refno</i> = 0 and <i>ringtype</i> = <i>ccr</i> . <i>CCREstimateMotion</i> selects an eddy in the region to update if there is one, does initial estimates of radius, and gets GS interaction info. User-defined functions used in these rules are in <i>returns.c</i> .
<code>explain.clp</code>	These rules handle the explanation requests from the user.
<code>k-stru.clp</code>	Contains a <i>deffacts</i> structure and six <i>deftemplates</i> . Please note that the sequence of the fields defined in templates will affect the values returned by function call <i>GetMFValue</i> , which is called many times in the <i>start()</i> routine found in <i>eddies.c</i> file.
<code>wcrrules.clp</code>	Contains the rules used to predict the movement of warm-core eddies. The templates used by these rules are in <i>k-stru.clp</i> . Please note that this rule file must physically reside in the directory specified during the <i>get-user-option</i> session. The start procedure in module <i>eddies</i> cycles through all regions asserting <i>goal</i> with <i>refno</i> = 0 and <i>ringtype</i> = <i>wcr</i> . <i>WCREstimateMotion</i> selects an eddy in the region to update if there is one, does initial estimates of radius, and gets GS interaction info.

---