CARMA: Platform Freedom for a Graphical Lisp Application through Armed Bear Common Lisp

John D. Hastings
Dept. of Computer Science
University of Nebraska-Kearney
Kearney, NE, USA
hastingsjd@unk.edu

Alexandre V. Latchininsky
Dept. of Renewable Resources
University of Wyoming
Laramie, WY, USA
latchini@uwyo.edu

Abstract

CARMA is an advisory system that uses artificially-intelligent techniques including case-based reasoning to provide advice about the most environmentally and economically effective responses to grasshopper infestations. CARMA's core AI reasoner was initially written in Common Lisp and integrated with an Allegro Common Lisp for Windows graphical user interface (GUI). CARMA went public in 1996 and has been used successfully since. Recently, CARMA's architecture was reworked in order to avoid periodic development and deployment fees, and to produce a platform-independent system by following a philosophy called *platform freedom* which emphasizes freedom from both platform dependence and software costs. The implementation also demonstrates an approach to creating a Lisp application with an appealing GUI which is web capable. This paper details CARMA's new architecture including the two-way communication between the two distinct main parts: 1) a Lisp AI reasoner which runs inside the Armed Bear Common Lisp interpreter which in turn runs inside the Java interpreter (JVM), and 2) a Java GUI which runs inside the JVM.

Keywords Lisp, Java, architecture, decision support, grasshopper management, artificial intelligence, case-based reasoning, approximate-model-based adaptation

International Lisp Conference March 22–25, 2009, Cambridge, MA, USA

1. Introduction

CARMA is an advisory system for grasshopper infestations that has been successfully used since 1996 (Hastings, Branting, & Lockwood 2002). CARMA, short for CAse-based Rangeland Management Advisor, employs a variety of artificially-intelligent (AI) techniques to provide advice about the most environmentally and economically effective responses to grasshopper infestations. In the process, CARMA demonstrates an approach to providing advice concerning the behavior of a complex biological system by exploiting multiple, individually incomplete, knowledge sources (Hastings, Branting, & Lockwood 1996) including utilization of a technique known as *approximate-model-based adaptation* which integrates case-based reasoning with model-based reasoning for the purposes of prediction within complex physical systems.

CARMA's core AI reasoner was written in Common Lisp and was integrated with an Allegro Common Lisp for Windows interface. CARMA went public in 1996 and has been used successfully since. Recently, in order to avoid periodic development and deployment fees, and to produce a platform-independent system, CARMA's architecture was reworked. We call this general philosophy *platform freedom* because it advocates the freedom to execute the application on a wide variety of platforms, as well as the freedom to develop and execute the application for and on those platforms using freely available technologies. In addition to platform independence, CARMA's implementation demonstrates an approach to creating a web-capable Lisp application with an appealing GUI. This paper details the new architecture with a particular focus on the two-way communication between the two

distinct main parts: 1) the Lisp AI reasoner which runs inside the Armed Bear Common Lisp (ABCL) interpreter and 2) a Java GUI which runs inside the Java interpreter or Java "virtual machine" (JVM). The authors provided a cursory introduction of this approach in a paper (Latchininsky, Hastings, & Schell 2007) which focused mainly on CARMA's value as an advisor and provided only general details of the integration.

Section 2 provides a brief overview of CARMA and its problem domain. Section 3 touches on the main contribution of CARMA to the field of AI. Section 4 describes CARMA's initial implementation and is followed in Section 5 by challenges to CARMA's long term success resulting from early design decisions. The redesigns considered for CARMA are set forth in Section 6, with the chosen solution, ABCL/Java, introduced in section 7. Section 8 provides technical details of the ABCL/Java integration within the new CARMA, followed in Section 9 with some of the challenges encountered during the update.

2. CARMA: Grasshopper Infestation Advisor

Grasshoppers annually consume about 25% of rangeland forages in the 17 western U.S. states at an estimated loss of US$950 million. Figure 1 shows a hopper band of the Clearwing grasshopper in Wyoming. It illustrates the potential severity of grasshopper infestations where densities may reach as high as several hundred grasshoppers per square yard over extensive rangeland areas. During outbreaks, grasshoppers inflict severe damage to rangeland and crops and require large-scale insecticide applications to control them. For example, during the 1986-88 outbreaks, 20 million acres of rangeland were treated with 1.3 million gallons of insecticides at a cost of US$75 million. Since the 1980s, the federal funding for grasshopper pest management has dwindled dramatically. Nowadays the responsibility for grasshopper control in the U.S. is borne almost entirely by the producer. Therefore, there is a compelling need to develop efficient, economically and environmentally viable strategies of grasshopper management. Such strategies would sustain agricultural profits, reduce the insecticide expense and preserve non-target fauna.

CARMA is a decision-support system which targets the end-users (ranchers and farmers) and addresses the need for proper grasshopper infestation response (Latchininsky, Hastings, & Schell 2007). CARMA gives advice by comparing the current infestation to known previous infestations (i.e., cases) and adapting management recommendations accordingly. Infestation probabilities and treatment efficacies are used to predict reinfestations (Branting, Hastings, & Lockwood 1997; Zimmerman, Lockwood, & Latchininsky 2004); statistical methods are used to predict the economic benefits; and rules are used to select the treatments. In addition to conventional, blanket applications of insecticides, CARMA includes an option called Reduced Agent and Area Treatments (RAATs) (Lockwood & Schell 1997) in which the insecticide dose rates are reduced from conventional levels by alternating treated and untreated swaths. This results in a lower environmental impact and cost of the treatment program. In 2003, the RAATs strategy was applied to 400,000 acres in Wyoming which saved half a million US dollars for local agriculturists. The contribution that CARMA has played and continues to play in supporting the development and implementation of sustainable pest management strategies such as RAATs is detailed in Hastings, Latchininsky & Schell(2009).

Figure 1. Hopper band of early instar nymphs of the Clearwing grasshopper *Camnula pellucida*, one of the most important economic pest grasshopper species in the western U.S. Photo: A. Latchininsky.

Initially, the geographic extent of CARMA was limited to Wyoming rangelands. In 2003, a special module dealing with grasshopper infestations in cropland was developed. It handles situations when grasshopper populations build up at the rangeland-cropland interface and spread into cropland, such as small grains. In recent years, several western states expressed interest in

CARMA, which resulted in the expansion of its capabilities. In addition to Wyoming, the geographic extent of CARMA currently covers grasshopper infestations in Colorado, Montana, Nebraska, New Mexico, North Dakota, Oregon, and South Dakota.

3. Contribution to AI

CARMA's particular problem domain, rangeland pest management advising, requires in part predicting the forage consumption by grasshoppers in an infestation. As is typical of biological systems, the interactions affecting grasshopper population dynamics are too poorly understood and too complex to permit precise prediction through numerical simulation (Lockwood & Lockwood 1991).

CARMA counters the complexity of the task by introducing a novel AI approach termed *approximate-model-based adaptation*[1] (Hastings, Branting, & Lockwood 1995; Branting & Hastings 1994) which utilizes case-based reasoning to provide an approximate solution and model-based reasoning to adapt this approximation into a more precise solution. Cases compensate for model incompleteness and the model compensates for insufficient coverage by the cases. Approximate-model-based adaptation is most applicable to complex physical systems with models and empirical data which are individually insufficient for the purposes of prediction.

CARMA employs approximate-model-based adaptation to make forage consumption predictions as follows. Each case represents a prototypical infestation scenario with an associated forage loss. New infestations are compared to prototypical cases to identify the most similar previous infestations, and the model is employed to account for any differences between the prototypical cases and the new infestation by adjusting the prototypical case estimates in order to provide a forage loss prediction for the new infestation.

4. CARMA: The Early Years

The Lisp programming language was invented in 1958 (Steele & Gabriel 1993). Lisp was known as an artificial intelligence (AI) language for years (Graham 1993), and still holds a special place amongst AI enthusiasts due in part to its ability to evolve through the implementation of new operators and abstractions using the Lisp language itself (Graham 1996), and the ease through which it can support elegant solutions to problems too complex to solve conveniently with conventional programming languages.

Figure 2. CARMA – a grasshopper infestation advisor. CARMA logo adapted from artwork in Capinera & Sechrist(1982).

A variety of not entirely compatible Lisp flavors were combined to produce Common Lisp, with the first specification appearing in Steele(1984), followed by standardized ANSI Common Lisp which was previewed in Steele(1990). It was around this time (early 90's) that work on CARMA began. Given that CARMA was an AI system, the choice of Lisp was completely natural. The AI core of CARMA was written in Common Lisp and was able to run on any platform (e.g, Linux, Macintosh, Windows) for which a Lisp interpreter was available. Initially, CARMA was a research tool with development and experimentation conducted on Sun Sparc and Silicon Graphics machines.

In order to fully develop CARMA into a user-friendly advising system, the Common Lisp core was augmented with a GUI implemented in Allegro Common Lisp (ACL) for Windows. CARMA was released to the public as CARMA 2.0 in 1996 and was subsequently featured as a customer success story on the Franz, Inc. website (Franz, Inc. 2008).

5. Impediments to Long-term Success

While CARMA operated perfectly for several years, aggravation set in when changes to Windows in turn

[1] Approximate-model-based adaptation is defined and contrasted with perfect-model-based adaptation in Branting(1998).

led to required upgrades (and expenses) to ACL for Windows, which in turn necessitated time consuming changes to CARMA. While the expense of ACL may be compatible with commercial problem domains, CARMA is a non-commercial, public-service product which has little to no revenue stream: CARMA is provided free of charge and thus brings in no distribution income, and it targets a problem domain for which development funding is quite limited. As such, CARMA does not monetarily support periodic upgrades of the development software, particularly a variety for which deploying developed applications (ACL Enterprise) is more expensive than the standard developmental version (ACL Professional). Furthermore, CARMA's revenue stream rarely supports the staffing necessary to make periodic coding changes, thus, long-term stability of the software is a key concern.

Avoiding "maintenance" expenses that had little to do with the application itself, and were instead tied to external changes in the operating system or Lisp environment was combined with a desire to provide a multi-platform product. Although CARMA's Lisp AI core was capable of running on multiple platforms, the GUI was restricted specifically to Windows. Our long term goal was to open up CARMA to a wider user base. At the time that our concerns came to a head (around the early 00's), a separate license was required for ACL for each platform, and furthermore, ACL did not seem to offer the support for conveniently developing standalone applications with a consistent GUI with a shared code base which would be capable of running on various platforms.

Common sense, then, dictated a solution not tied to a single platform and not dependent on a fee-based development environment for which CARMA would continue to face periodic, upgrade fees and time consuming updates. The challenge was then to formulate an approach which would make CARMA "platform free" by making it accessible on a wide variety of computing platforms using freely available technologies, with a particular need to rework the GUI through a free graphical language.

6. Architectural Possibilities

A variety of "free" approaches to reworking CARMA's architecture were considered. Each of the possibilities include some use of Java (Gosling *et al.* 2000), an interpreted programming language with acceptable graphical capabilities and a nice degree of platform independence tied to the availability of a Java interpreter for a relatively wide variety of operating systems. The considered approaches are summarized in the following subsections.

6.1 Complete rewrite in Java

A complete rewrite of both CARMA's Lisp AI core and GUI in Java would provide platform independence of the entire application. It is in fact not uncommon to see AI systems, e.g., Weka (Witten & Frank 2005), written in Java even if such coding is not entirely natural or elegant. However, given that CARMA's Lisp core contains thousands of lines of code, and Lisp and Java are entirely different languages, a complete rewrite would be time consuming, if not overwhelming. Particular items that would make a Java rewrite unpleasant include:

- Lisp built-in list structures which are used extensively in CARMA, and
- the ability in Lisp to use identifiers without specifically declaring their associated data type, a feature used for each of CARMA's variables.

It appears that the amount of code would explode if completely converting to Java. The lack of elegance offered by a total Java solution also represents a psychological hurdle to a long time Lisp aficionado.

6.2 Server-side Lisp

An Apache server module named mod_lisp (Battyani 2008) would make the Lisp AI reasoner available through Apache and would require developing an outer web interface (e.g., using Java). An advantage would be that the Lisp reasoner would likely require few changes and/or additions in order to integrate with the web interface. A second advantage would be that CARMA could be updated or revised as needed (e.g., to remain consistent with the latest grasshopper management practices or to include new data) and the new version would be immediately available by running it from the CARMA website rather than notifying users that they should download and install the most recently released version. The disadvantages include a dependence on uptime of the server, and a discontinuation of the option to install and run CARMA locally (a rather large dis-

advantage for users who access CARMA in the field or other remote locations without Internet access).[2]

6.3 Java / Lisp communication framework

Jacol (Lowdermilk 2002) is a framework for supporting communication between Java and Common Lisp using sockets. It allows Common Lisp to act as an extension language for Java although it is also said to expose Java APIs to Lisp. This framework could potentially allow the CARMA Lisp AI reasoner to be integrated with a Java GUI. The slight disadvantage to this approach is that Jacol is said to require CLISP (CLISP 2008), and would thus result in logistical complications in order to make CARMA conveniently available on multiple platforms (e.g., Could or should CARMA be bundled with CLISP for each platform, or would the user be required to complete an additional step by obtaining CLISP separately?). This choice would also restrict CARMA to the fewer platforms for which CLISP is available. As another option, Jacol might also support a client/server configuration (such as that described in the previous section). In limited testing, Jacol did not seem to represent a sufficiently mature solution and was unlikely to progress further given that development appeared to cease in 2002.

6.4 A Lisp interpreter written in Java

A Lisp interpreter written in Java could run CARMA's AI core and could include hooks to provide access from the Lisp code to Java's graphical capabilities. A disadvantage of this approach would be the inefficiency of running an interpreter inside another interpreter, i.e., CARMA's AI core would run through a Lisp interpreter running inside a Java interpreter. In addition, writing a Lisp interpreter (in Java) would be a daunting task relative to making an existing application continue to run, especially given a lack of time and resources. As with the previous two approaches, an advantage would be that the Lisp AI core would need few changes. Additional advantages would include support for an appealing GUI, and platform independence (i.e., CARMA could run inside this new Lisp interpreter on any platform for which a Java interpreter is available). In addition, by being wrapped inside a Java-coded interpreter and not tied to a Lisp implementation on a particular platform, CARMA's code would be somewhat insulated from changes to operating systems (e.g., Microsoft) assuming that the Java language does not undergo periodic changes which would then require maintenance of the new interpreter. A further advantage of this approach is that CARMA would be capable of running either as a platform-independent standalone application (by downloading and installing) or through a web browser using Java Web Start, an option which would allow users to conveniently access and run the most current version of CARMA (assuming support by the browser).

[2] Recently, a option called ABCL-web (ABCL-web 2008) has become available, although it is currently in alpha, and was not available at the time of the decision.

7. Survey Says: ABCL

Small prototypes for the first three options were developed and subsequently deemed unacceptable due generally to the disadvantages mentioned above. In 2003, shortly after we initiated the effort to write a Lisp interpreter in Java (the fourth option), Peter Graves fortunately beat us to the punch with Armed Bear Common Lisp (ABCL) (Graves 2008). ABCL is a very capable Common Lisp implementation that runs within a Java interpreter. According to the ABCL website, the most recent version, ABCL 0.0.11, is said to fail "only" 47 out of 21702 tests in the GCL ANSI test suite.[3] In terms of its abilities for the subset of Lisp used in CARMA (which makes some use of CLOS), ABCL produced expected results for CARMA's test cases as did the available versions of CLISP and ACL, while GNU Common Lisp (GNU 2008) did not.

With the availability of ABCL, attention quickly shifted to developing the Java GUI and linking it to CARMA's AI core through ABCL. An overview of the new architecture of CARMA is given in the next section.

8. The New Architecture

CARMA appears to the user as a Java application, although the logic and primary program control resides within the Lisp AI core. CARMA's architecture, illustrated in Figure 3, is comprised of two distinct components:

1. A Java GUI which runs as a Java application through the JVM, and

[3] ABCL 0.0.8, the version used within CARMA, fails only slightly more test cases although I am no longer able to find those statistics.

2. The AI core which runs as a Lisp application through the ABCL interpreter (which in turn runs through the JVM).

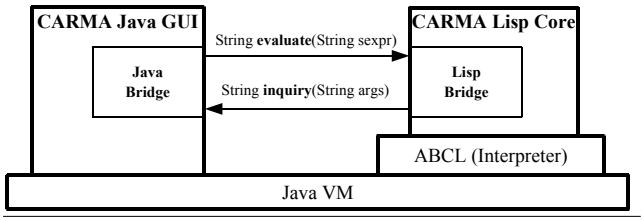

Figure 3. A general overview of the architecture of CARMA.

Two-way communications between the Lisp and Java components are handled by bridge modules. In general, the Lisp bridge creates the Java bridge object, and through this connection directs questions and messages to the Java bridge whenever such information should be presented to the user through the GUI. The Java bridge is responsible for answering such user input requests, and is in turn capable of making requests of the Lisp bridge when a user requests a clarification (e.g., hitting a "Help" button) which is in turned handled by the Lisp AI core (which retains the domain logic).

Notice in Figure 3 that the `evaluate` and `inquiry` methods are the primary means of communication between the two sides. `inquiry` is a method defined within the Java bridge and is called from the Lisp side. `evaluate` is a Java method within ABCL to which Lisp s-expressions can be passed for evaluation. Although `evaluate` is technically part of ABCL, it is the means through which Lisp bridge functions can be called from the Java side.

An overview of the two-way communication between the two components within the context of an advising session is as follows:

1. CARMA is executed as an ordinary Java application with a standard Java GUI, in this case a Swing `JFrame` class called `CarmaFrame`. During initialization of `CarmaFrame` (in its constructor), the user is presented with a Java splash screen (shown in Fig. 2) during which time `CarmaFrame`:
   (a) forces the ABCL interpreter to load into the JVM by evaluating a dummy Lisp expression using the statement:

```
org.armedbear.lisp.Interpreter
  .evaluate("(+ 3 5 )" );
```

   (b) instructs ABCL to load the CARMA Lisp AI core source files into the Lisp interpreter one by one by calling:

```
LispLoad.loadCarmaFile(lisp-filename,
  false, false, false);
```

   All of CARMA, including ABCL, is bundled within a single Java jar (Java archive) file. The `loadCarmaFile` method within `LispLoad` is a slightly modified version of the `load` method within the `Load` class provided by ABCL such that the Java `URL` class and `openStream` method are used to allow loading of the CARMA Lisp source files into the ABCL interpreter from the jar file.

   The previous two initialization steps are conceptually part of the Java bridge in the sense that the Java side is contacting the Lisp side, although they are performed prior to the creation of a specific Java bridge object through which most communication on the Java side is channeled.

2. Once initialization is complete and the CARMA GUI has appeared, the user is presented with a selection of menu options, the most important of which is the option to run a new advising consultation. When the user chooses "New consultation", the selection triggers the following Lisp command from the Java side which requests the Lisp bridge to run a consultation and in the process transfers control of code execution to the CARMA Lisp AI core:

```
org.armedbear.lisp.Interpreter
  .evaluate("(consultation nil)" );
```

3. With control transferred to the Lisp side, the Lisp bridge initializes the Lisp to Java communication path as follows:
   (a) If a Java bridge object has not yet been created, it is instantiated using the ABCL functions `jnew` and `jconstructor`:

```
(setq *java-bridge-object* (jnew
  (jconstructor "Bridge")))
```

   "Bridge" is the name of the Java class from which the Java bridge object is instantiated. ABCL has the option of linking to a Java constructor with parameters but that feature was not necessary in our implementation.

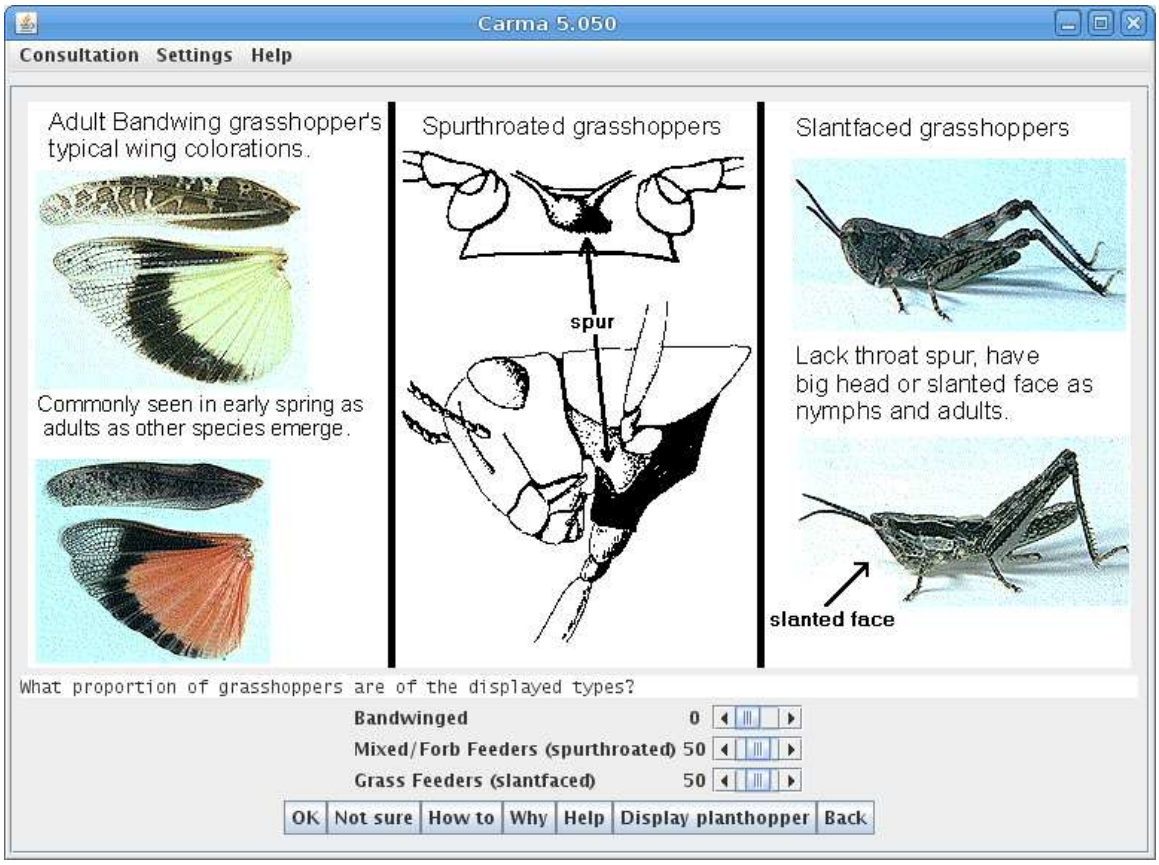

Figure 4. Elicitation of Grasshopper Type Information in CARMA.

(b) Links to Java bridge methods are made using the ABCL function `jmethod`, *e.g.*,

```
(setq *java-bridge-inquiry-method*
  (jmethod "Bridge" "inquiry"
    "java.lang.String"))
```

In this case, `inquiry` is specified to be a Java method tied to a `Bridge` object which accepts a single `String` argument. Although ABCL allows a connection to Java methods with multiple parameters and with various types, the decision was made to keep the connection simple and pass parameters across in a single `String`. When a specific inquiry requires multiple parameters, they are concatenated into a single string, passed across and parsed on the Java side. The reason for this design choice is explained in the next section.

4. Once the Lisp to Java connection is initialized, the consultation begins. During its reasoning process, the Lisp AI core directs questions or messages to the Java bridge:

(a) When the Lisp AI core has a question for the user, it passes the question along with any additional arguments as a string (e.g., "location-state" for determining the state of the infestation) to the Java bridge through the Lisp bridge which then prompts the appearance of an input window. The user's response is passed back to the Lisp bridge in the form of a string. The call to the Java `inquiry` method using the ABCL `jcall` function, with the parameters `*java-bridge-object*` and `*java-bridge-inquiry-method*` initialized above, is coded as follows:

```
(setq return-string (jcall
  *java-bridge-inquiry-method*
  *java-bridge-object*
  question-string))
```

(b) On the Java bridge side, the `inquiry` method signature is:

```
public String inquiry (String args)
```

`inquiry` parses the `String` parameter to determine the question type along with any associated trailing parameters, triggers the appearance of the necessary input panel (which uses the trailing parameters), and returns the result as a `String`. A typical Java input panel appears in Figure 4.

(c) The Lisp bridge parses the returned string into a list on the way to the Lisp AI core for use in the analysis of the consultation.

9. Development Issues

By and large, the biggest challenges encountered during the development of the new architecture were related to determining how best to utilize ABCL to make the bridge modules communicate properly. A summary of other development issues or choices are provided in the following subsections.

9.1 Decoupling the View

Although ABCL would support direct calls from the Lisp side to any variety of methods on the Java side, the decision was made to simplify the connection so that the Lisp side was not filled with and dependent on specific Java method names (other than the main Java bridge methods). When the Lisp side has a question of the user, it is passed to the Java side through the bridge, then mapped to a specific input screen on the Java side. In converting to ABCL/Java, very few lines of code within the CARMA Lisp AI core were modified other than the former calls to ACL for Windows code which were replaced by questions passed through the Lisp bridge. If necessary at some later date the Java GUI could be swapped out for an alternate GUI environment without requiring changes to the Lisp code other than the Lisp bridge. This loose coupling puts CARMA more in line with approaches such as the Model-View-Controller (MVC) architecture (Reenskaug 1979) which advocate separating the view from the logic.

9.2 Stack Overflow

Once the new architecture was in place, CARMA encountered Java stack overflow errors tied to some of its recursive non-GUI Lisp functions which had worked perfectly prior to the conversion. CARMA, like many Lisp programs, makes extensive use of recursion as its looping mechanism. This failure was initially quite troubling. It seemed apparent that Java (or perhaps the ABCL interpreter) was not as adept at handling recursion. We ultimately discovered that only a handful of CARMA's Lisp functions triggered the overflow, and although they did not recurse very deeply in our opinion, they recursed just enough to cause problems. Rather than trying to "eliminate" the stack-overflow issue by increasing the Java stack size or further digging into the ABCL code, the few "erroneous" recursive Lisp functions were rewritten iteratively using `loop while` until a CARMA advising consultation could run to completion without errors on an old personal computer with limited memory. It is possible that this issue could reappear on less powerful computing devices or environments such as PDA's.

9.3 ABCL Version

Another slight issue was selecting the version of ABCL to embed within CARMA. At the time of the initial integration, ABCL 0.0.8 was used. ABCL 0.0.8 handles Lisp files (including its own Lisp library) as plain text source files. A more recent version, ABCL 0.0.9, includes a feature to compile Lisp source to Java bytecodes. Furthermore, ABCL 0.0.9 itself represents its internal Lisp library using compiled Lisp files (i.e., Java bytecodes). A quick check of the two versions shows:

- ABCL 0.0.8: 781 Java class files (1.1MB) and 154 Lisp source files (1.1MB)
- ABCL 0.0.9: 1010 Java class files (2.2MB) and 4495 compiled Lisp files (7.5MB)

Although 0.0.9 includes additional features (which CARMA doesn't require) that partly contribute to the larger size, the bigger issue is that Lisp source code is generally smaller in size than the equivalent compiled Lisp→Java files, for example:

- ABCL 0.0.8: *typep.lisp* is 5.3KB
- ABCL 0.0.9: the five compiled files for *typep.lisp* total 11.8KB

Because CARMA is available from the web, distribution size and associated download times are a key concern. For this reason, CARMA continues to use ABCL 0.0.8 with its Lisp modules in plain text, and represents its own Lisp core using non-compiled Lisp source in order to minimize the distribution size of CARMA (both in terms of the embedded ABCL as well its own Lisp files). Although this choice might require a slightly longer time for loading the ABCL and CARMA Lisp

source into the Lisp interpreter (if Lisp source modules do indeed take longer to load than compiled Lisp modules), the time needed for downloading CARMA is reduced.

The most recent version of ABCL, 0.0.11 has not been reviewed.

9.4 Speed of ABCL

Although some have found ABCL to be substantially slower than other Lisp implementations (some have pointed specifically to interpreted code), its speed has not presented a noticeable lag for our purposes. Because CARMA performs its calculations as a consultation proceeds, rather than all at once after a set of user inputs, any delay caused by ABCL is mostly imperceptible (this feature was particularly advantageous on the slower Windows environments of the mid-90's). The only noticeable delay is the initial load of CARMA's Lisp core into ABCL. However, for running a large set of scripted tests, we might prefer a faster Lisp interpreter.

9.5 Differences Between ABCL and other Lisp Flavors

For the subset of Lisp used within CARMA, syntax for ABCL was generally consistent with other popular Lisp implementations unless we dug too deeply. For example, one portion of CARMA requires a list of slot names for an object. CLISP provides a CLOS method, while several other implementations (including ABCL) do not. A snippet of CARMA's `list-of-slots` function which illustrates this difference is as follows:

```
(defun list-of-slots (object)
  #+clisp (clos::slot-names object)
  ...
  #+cmu (list-of-slots-aux
     (clos-mop::class-direct-slots
        (class-of object)))
  #+armedbear (list-of-slots-aux
     (system::class-direct-slots
        (class-of object))))
```

`list-of-slots-aux` is a helper function (not shown) that takes a list of slot definitions, and picks off and returns a list of the slot names.

10. Availability

The most recent version of CARMA, 5.050, with grasshopper advising capabilities for Colorado, Montana, Nebraska, New Mexico, North Dakota, Oregon, South Dakota and Wyoming is available free of charge for noncommercial purposes and can be downloaded and installed from http://carma.unk.edu, or run as a Java Web Start application.

11. Conclusion

CARMA is a long running advising system for western U.S. grasshopper management whose architecture was recently upgraded in order that it continue to provide a valuable public service. The architectural changes to CARMA demonstrate an approach to using ABCL to integrate a Lisp application with a Java GUI in a manner which leverages the strengths of the two languages (i.e., Lisp for elegant reasoning and Java for an appealing interface). The solution supports two-way communication between the two components, frees the developer from licensing costs, and produces an application which is platform independent and web capable. The approach should be particularly relevant to those wishing to create highly portable, real-world Lisp applications which require an interesting user experience.

12. Acknowledgments

CARMA's development since 2003 was supported through funds from Cooperative Agreements between USDA-APHIS-PPQ (Western Region) and the University of Wyoming (grants USDAAPHIS5112, USDAAPH44906 and USDAAPH44909GHS).

References

ABCL-web. 2008. ABCL-web (web page). http://abcl-web.sourceforge.net (Accessed 11 Nov 2008).

Battyani, M. 2008. Fractal Concept: mod_lisp home page (web page). http://www.fractal concept.com/asp/mod_lisp (Accessed 11 Nov 2008).

Branting, L. K., and Hastings, J. D. 1994. An empirical evaluation of model-based case matching and adaptation. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94) Workshop on Case-Based Reasoning (WS–94–01)*, 72–78. Menlo Park, CA, USA: AAAI Press.

Branting, L. K.; Hastings, J. D.; and Lockwood, J. A. 1997. Integrating cases and models for prediction in biological systems. *AI Applications* 11(1):29–48.

Branting, L. K. 1998. Integrating cases and models through approximate-model-based adaptation. In *Proceedings of the AAAI 1998 Spring Symposium on Multimodal Reasoning (SS-98-04)*, 1–5. Menlo Park, CA, USA: AAAI Press.

Capinera, J. L., and Sechrist, T. S. 1982. Grasshoppers (Acrididae) of Colorado: Identification, biology, and management. Technical Report 584S, 1M, Colorado State University Experiment Station, Fort Collins, CO, USA.

CLISP. 2008. CLISP - an ANSI Common Lisp Implementation (web page). http://clisp.cons.org (Accessed 11 Nov 2008).

Franz, Inc. 2008. Franz Inc Customer Applications: University of Wyoming Applied AI Lab (web page). http://www.franz.com/success/customer_apps/research /uwyoming.lhtml (Accessed 11 Nov 2008).

GNU. 2008. GCL - GNU Common Lisp (web page). http://www.gnu.org/software/gcl (Accessed 11 Nov 2008).

Gosling, J.; Joy, B.; Steele, G.; and Bracha, G. 2000. *The Java Language Specification*. Boston, MA, USA: Addison-Wesley, 2nd edition.

Graham, P. 1993. *On Lisp: Advanced Techniques for Common Lisp*. Upper Saddle River, NJ, USA: Prentice Hall.

Graham, P. 1996. *ANSI Common Lisp*. Upper Saddle River, NJ, USA: Prentice Hall.

Graves, P. 2008. Armed Bear Common Lisp (web page). http://common-lisp.net/project/armedbear (Accessed 11 Nov 2008).

Hastings, J. D.; Branting, L. K.; and Lockwood, J. A. 1995. Case adaptation using an incomplete causal model. In *Proceedings of the First International Conference on Case-Based Reasoning (ICCBR-95) Lecture Notes in Artificial Intelligence 1010*, 181–192. New York, NY, USA: Springer.

Hastings, J.; Branting, K.; and Lockwood, J. 1996. A multi-paradigm reasoning system for rangeland management. *Computers and Electronics in Agriculture* 16(1):47–67.

Hastings, J.; Branting, K.; and Lockwood, J. 2002. CARMA: A case-based rangeland management adviser. *AI Magazine* 23(2):49–62.

Hastings, J. D.; Latchininsky, A. V.; and Schell, S. P. 2009. Sustainability of grasshopper management and support through CARMA. In *Proceedings of the 42nd Hawaii International Conference on System Sciences (HICSS-42)*, 10 pages, CDROM. Los Alamitos, CA, USA: IEEE Computer Society.

Latchininsky, A. V.; Hastings, J. D.; and Schell, S. S. 2007. Good CARMA for the high plains. In *Proceedings of the 2007 Americas' Conference on Information Systems (AMCIS 2007)*, 9 pages, CDROM. Association for Information Systems.

Lockwood, J. A., and Lockwood, D. 1991. Rangeland grasshopper (Orthoptera: Acrididae) population dynamics: insights from catastrophe theory. *Entomological Society of America* 20(4):970–980.

Lockwood, J. A., and Schell, S. P. 1997. Decreasing economic and environmental costs through reduced area and agent insecticide treatments (RAATs) for the control of rangeland grasshoppers: Empirical results and their implications for pest management. *Journal of Orthoptera Research* 6:19–32.

Lowdermilk, J. 2002. JACOL - Java and Common Lisp (web page). http://jacol.sourceforge.net (Accessed 11 Nov 2008).

Reenskaug, T. M. H. 1979. Trygve/MVC (web page). http://heim.ifi.uio.no/ trygver/themes/mvc/mvc-index. html (Accessed 11 Nov 2008).

Steele, G. L., and Gabriel, R. P. 1993. The evolution of Lisp. In *Second ACM SIGPLAN History of Programming Languages Conference (HOPL-II)*, 231–270. New York, NY, USA: ACM.

Steele, G. L. 1984. *Common Lisp: The Language*. Newton, MA, USA: Digital Press, 1st edition.

Steele, G. L. 1990. *Common Lisp: The Language*. Newton, MA, USA: Digital Press, 2nd edition.

Witten, I. H., and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. San Francisco, CA, USA: Morgan Kaufmann, 2nd edition.

Zimmerman, K. M.; Lockwood, J. A.; and Latchininsky, A. V. 2004. A spatial, markovian model of rangeland grasshopper (Orthoptera: Acrididae) population dynamics: Do long-term benefits justify suppression of infestations? *Environmental Entomology* 33(2):257–266.