Bill J. Ellis
December 21, 2001

NuSPADE - an opportunity to polish up SPADE?

1 Introduction

An important question where building a new system is which programming language to use. It would be unwise to write a nuclear power plant control system in Java[1] or an operating system in a functional language like SML[2]. Here the new system being developed is NuSPADE. This will exist as a wrapper around SPADE, adding new functionality.

In this particular situation many components of the system already exist. It is intended to use SPADE without a single amendment, and it is written in Prolog. It is also intended to borrow a substantial amount of code from CLAM, also written in Prolog.

Thus, it is accepted that NuSPADE will be written in Prolog. However, it is not quite clear which flavour of Prolog to use. Ideally, NuSPADE should be able to exist on all platforms that SPADE is currently available on[3]. Further, NuSPADE should be flexible enough to remain in sync with SPADE as it undergoes expected incremental changes.

1.1 Current Situation

SPADE is currently available on three different platforms:

- **Unix**
- **Vax**
- **Microsoft Windows**[4]

The Unix and Vax platforms are very similar. They share the same Unix-like operating system and both use the X Window System. Microsoft Windows is quite different to Unix and Vax.

A large amount of the SPADE code implements a console based proof checker, Console-SPADE. This includes all of SPADEs theorem proving functionality, plus a fully functional text interface. Further, Console-SPADE includes facilities to interrogate a licence server, Using the FLEXlm program.

The Unix and Vax versions do not add anything to Console-SPADE. The only difference is a single predicate used to identify the different target machines.

The Windows version of SPADE adds a Graphical User Interface (GUI) to Console-SPADE. However, the extent of this interface is very limited. All interactions with SPADE are still through the text interface of Console-SPADE. The GUI uses a window to display the current VC permanently on screen and another window for user interaction. Perhaps less obvious, the Windows version also has its own dialogue box for selecting the VCG input file. In Unix and Vax this is requested using the text interface or as a parameter.

These minor differences between the two systems focus on the look-and-feel of SPADE. The Unix and Vax versions follow the text based nature of their operating systems. The Windows version follows the graphical, user centric design, of its operating system.

Depending on the platform, SPADE is compiled by one of two different Prologs. The Unix and Vax versions are compiled using Poplog. The Windows version is compiled using LPA WIN-Prolog. Thus the current situation can be summarised:

- Poplog
  - **Unix and Vax -** Console SPADE
- LPA WIN-Prolog
  - **Windows -** Console SPADE + GUI

*Notes in this series are for ϵ-baked ideas, for $1 \geq \epsilon \geq 0$. Only exceptionally should they be cited or distributed outwith the CLAM-SPARK project (EPSRC GR/R24081).

[1] In fact, under the Sun licence, it would probably be illegal!

[2] Where strong type security and no support for imperative programming would probably slow things down a little too much.

[3] As noted in CS-Note 6, the early versions of NuSPADE will focus on Solaris. However, it is intended to keep the option open for other platforms, especially Microsoft Windows.

[4] SPADE is only supported on NT at the moment. However, it also operates on Windows 95/98.

2 Ideal Requirements

It is helpful to outline the ideal requirements of the NuSPADE system.

2.1 System Building

The choice of how to build a system is separate to the choice of the tools used to build the system. Some key features have been identified that are likely to lead to a system that is easier to maintain and upgrade.

- **Same GUI code across platforms -** As effort is placed on upgrading and extending the GUI for a program, the benefits should be passed on to every platform. It should not be the case that a separate GUI is developed for each platform. As noted earlier, there is an alternative argument. A different GUI should be created for each platform, allowing an interface that suits the look-and-feel of its operating system. Although there is some merit to this argument, it does require duplicated programming effort. Further, in practise all operating systems are tending to adopt a similar window-like environment. It is quite possible to develop a single GUI that is not only technically portable across platforms but also portable in look-and-feel.
- **Same base code across platforms -** While there is a potential argument for developing different GUIs for different platforms, the same is not true of the base code[5]. To ensure changes to the base code are passed on to every platform it is most straight forward to have a portable single version of the source code.
- **One language solution -** Modern systems often encompass many different languages. This allows components of the system to be written in a language that is most applicable. However, the introduction of more languages brings more complexity to the system. It will be necessary to find some method to allow the different languages to communicate. It will also be necessary to learn the other languages. Thus, while using additional languages is always a possible option, it is generally better if a one language solution can be found.

2.1.1 System Building Table

The following table will be used to provide a quick check list of these points.

	System
Same GUI code across versions	yes *or* no
Same base code across versions	yes *or* no
One language solution	yes *or* no

2.2 Prolog Compiler

- **Compiler exists -** A straight forward way to produce a system on different platforms is to use one code base, and find an appropriate compiler for each platform. Unfortunately, probably without exception, every compiler has its own little quirks. It is unlikely that different compilers would interpret the same code base in the same way.

  However, it is often the case that compilers are ported onto different platforms. Using the same compiler on different platforms avoids the need to accommodate compiler-quirks. Thus it is a is a viable way to use one code base on multiple platforms.
- **GUI component -** Different platforms, although from the user perspective looking fairly familiar, can be quite different internally. This is often particularly true for using the platforms graphical interface. Thus it is not uncommon for the same compiler on different platforms to require different GUI code.

  It is certainly beneficial if this can be avoided, by using a single GUI component that is portable across the compilers.
- **Prolog $\rightarrow$ Other language -** Communicating with other languages from within the Prolog compiler is a useful facility. For the purposes of SPADE it is also believed that some interface to another language is required to use FLEXlm[6].

[5] In this case Console-SPADE + NuSPADE wrapper.

[6] It is know that C is used at the moment. However, it is not know that this is the only the language that can be used to call FLEXlm. Largely this is not important, as nearly every Prolog has a C interface.

- **Other language → Prolog -** Communicating with Prolog from another language is perhaps a less useful facility. However, this does provide alternative ways to access the Prolog core.

- **Big-integer -** Big-integers are a type supported on some Prologs. They allow for the correct behaviour of integer arithmetic operations, even where the numbers involved are very big. Typically, where big-integers are not supported, integers are restricted to a machine register size (32/64 bit) and automatically converted to floating point where this size is exceeded. Such behaviour is not ideal in a formal reasoning tool.

- **Robust and supported -** In the case of a formal reasoning tool, a robust Prolog is essential. It is also desired that the Prolog is well supported, with good documentation, and access to developers for any tricky questions.

- **Good future prospects -** Although a system may be robust and supported at the moment, it is by no means certain this will remain the same in the future. To cater for new hardware and software, systems need to continue to adapt.

- **Cost -** SPADE is owned by a commercial company. If the Prolog compiler is too expensive it may not be commercially viable.

  The analysis of cost is outside the scope of this note. Emphasis has been placed on the technical aspects. It is conjectured that the cost of any Prolog will be a relatively small proportion to the cost of developing SPADE.

2.2.1 Prolog Compiler Table

The following table will be used to provide a quick check list of these points.

	Vax and Unix	**Windows**
Compiler exists	yes *or* no	yes *or* no
GUI component	GUI components *or* None *or* N/A	GUI components *or* None *or* N/A
Prolog → Other language	Other languages *or* None *or* N/A	Other languages *or* None *or* N/A
Other language → Prolog	Other languages *or* None *or* N/A	Other languages *or* None *or* N/A
Big-integer	yes *or* no *or* N/A	yes *or* no *or* N/A
Robust and supported	yes *or* no *or* N/A	yes *or* no *or* N/A
Good future prospects	yes *or* no *or* N/A	yes *or* no *or* N/A
Cost	Guideline cost	

3 Evaluating Various Prologs

Based on the criteria discussed above, a number of popular Prologs are evaluated.

3.1 Poplog

	Vax and Unix	Windows
Compiler exists	yes	yes
GUI component	X[a]	None[b]
Prolog → Other Language	C, Fortran	C, Fortran
Other Language → Prolog	None	None
Big-integer	yes	yes
Robust and Supported	yes	no[c]
Good future prospects	no[d]	no[d]
Cost	Free	

a With support for widget sets: athena, motif, openlook.

b Console interface only.

c Windows version exists, but not supported. Not even sure if it is available for commercial users.

d Since becoming free for commercial users, future development of Poplog has been left in the hands of few expert users.

Additional notes:

- Poplog is actually an integrated environment, including a number of different programming languages. These are POP-11, Prolog, Common LISP and Standard ML. For the purposes of NuSPADE only the Prolog part of Poplog is required. That the Prolog is actually a subset of Poplog poses no problems. The Prolog part of Poplog can be accessed independent to its other languages.

3.2 Sicstus

	Vax and Unix	Windows
Compiler exists	yes	yes
GUI component	TCL/TK[a]	TCL/TK[a]
Prolog → Other Language	C, Java	C, Java
Other Language → Prolog	C, Java	C, Java, Visual Basic
Big-integer	yes	yes
Robust and Supported	yes	yes
Good future prospects	yes	yes
Cost	$1200[b]	

a TCL/TK is a scripting language plus a collection of graphics widgets. In Sicstus Prolog predicates are available to interface with TCL/TK. However as arguments to these predicates are sections of TCL/TK code. Thus it will still be necessary to learn TCL/TK. TCL/TK is available on many platforms including Vax, Unix and Windows, free for commercial use. The TCL/TK program binaries can be installed along with the program that requires it. Due to the external nature of TCL/TK it is generally slower than directly built-in graphics.

b A distinction is made between academic, sponsor and commercial licences. Only the commercial licence allows the development of software that is not required to be given free of charge to Sicstus[7]. The cost of a commercial licence is dependent on the number of users. The price shown here is for starting a commercial licence for one user, with support for one year. Maintenance instalments are required for continued support after each year, costing a third of the original starting cost.

[7] Under such a scheme Sicstus has accumulated some powerful libraries developed by other parties.

3.3 LPA WIN-Prolog

	Vax and Unix	Windows
Compiler exists	no	yes
GUI component	N/A	PDM[a]
Prolog → Other Language	N/A	C, C++, Pascal, etc.[b]
Other Language → Prolog	N/A	Visual Basic, etc.[c]
Big-integer	N/A	no
Robust and Supported	N/A	yes
Good future prospects	N/A	yes
Cost	£995[d]	

a The Portable Dialog Manager (PDM) is produced by the same people that make LPA WIN-Prolog. It provides a graphics module that is portable across LPA WIN-Prolog, Quintus and LPA MacProlog. This is bundled with LPA WIN-Prolog, and covered with the same licence.

b A Dynamic Link Library (DLL) is available for communicating with external programs.

c A Dynamic Data Exchange (DDE) is available. This makes communication from many Windows programs possible.

d This is for starting a single user commercial licence with maintenance for one year. Optional annual maintenance is £495 for technical support and a further £495 for updates and upgrades. Note that £995 is about $1440.

Additional notes:

- LPA WIN-Prolog is designed to be closely compatible with Quintus Prolog. This effort, along with the PDM, means that graphical LPA WIN-Prolog programs can be ported to Quintus.

3.4 SWI

	Vax and Unix	Windows
Compiler exists	yes	yes
GUI component	XPCE[a]	XPCE[a]
Prolog → Other Language	N/A	C, C++
Other Language → Prolog	N/A	C, C++
Big-integer	no	no
Robust and Supported	no[b]	no[b]
Good future prospects	yes	yes
Cost	2000 Euro[c]	

a XPCE allows GUIs to be developed in Prolog and other interactive/dynamically typed languages. It is licenced similar to SWI. It advocates closer integration than, say, external TCL/TK. XPCE is a separate language to Prolog and will require learning.

b SWI is intended for research and education. It is developed by a small, but active, community. Thus, it is uncertain as to whether SWI is robust enough for a machine reasoning tool.

c SWI is distributed under the GNU General Public Licence (GPL). Thus SWI is free for public use, so long as any software that includes SWI is also free for public use. For commercial software this is not suitable. Thus a special licence is available for commercial users of SWI, with an extra clause ensuring that ownership remains with the commercial developer. Note that 2000 Euro is about $1800.

3.5 Quintus

	Vax and Unix	Windows
Compiler exists	yes	yes
GUI component	ProXT, ProXL[a]	None[b]
Prolog → Other Language	C, C++	C, C++
Other Language → Prolog	C, C++	C, C++, Visual Basic
Big-integer	no	no
Robust and Supported	yes	yes
Good future prospects	yes	yes
Cost	$5800[c]	

a ProXT provides access to the Motif widget set and X Toolkit via Prolog predicates. ProXL offers a comprehensive interface to X, supporting the functionality of Xlib.

b Console interface only.

c Quintus makes a distinction between academic and commercial licences. The commercial licence is nearly 6 times the cost of the academic licence. However, this licence is required to develop and distribute commercial programs. Similar to Sicstus, the price of this licence depends on the number of users. The price shown here is for starting a commercial licence for one user, with support for one year. Annual maintenance instalments for this single user licence is $1600.

4 Evaluating Various Systems

Making use of the discussed Prologs, a number of possible system building ideas are proposed.

4.1 System: Poplog and LPA WIN-Prolog

Reason:
As noted in the introduction, this is the current situation. Little effort is required to adopt this system as it is the current state of affairs.

Problems:

- There are quirks between Poplog and LPA WIN-Prolog. In particular different quoting styles are used. This requires duplicate sections of code to cater for the two different Prolog compilers.
- There is no common GUI component that is applicable on both Poplog and LPA WIN-Prolog. Different GUIs for each platform will need to be developed.
- Adding a GUI to Poplog will be done through its interface to X widget sets. This may not be particularly straight forward[8].
- Though Poplog is quite viable at the moment, its future is not so promising.
- LPA WIN-Prolog does not support Big-integer.

Changes to implement:

- Write code to deal with the lack of Big-integer in LPA WIN-Prolog.
- Add a GUI to the Poplog platforms.

	System
Same GUI code across versions	no
Same base code across versions	no
One language solution	yes

---

[8] The interface Poplog provides to widget sets closely resemble their C origins. X windows programming in this manner can be quite difficult.

4.2 System: Sicstus

Reason:
Sicstus is a powerful Prolog, providing all of the required features for a Prolog compiler. It is available on all target platforms so should ensure portable base-code. Its GUI using TCL/TK will also allow for a portable and flexible interface.

Problems:

- Although the GUI is thought of as being part of Sicstus, it is really supported by an external program TCL/TK. Thus it is necessary to learn TCL/TK, and distribute it with the finished product.
- TCL/TK is a scripting language. This allows for flexible and quickly created interfaces. However, for larger interfaces or programs that must be maintained over time, the scripting language may not be strong enough.

Changes to implement:

- Remove the GUI from the LPA WIN-Prolog version.
- Re-converge the separate Poplog and LPA WIN-Prolog code streams.
- Port the singe code stream to Sicstus. It is noted that, by design, LPA WIN-Prolog programs can be compiled by Quintus. The following extract was taken from the Sicstus web site:

  > SICS has acquired Quintus Prolog (QP) from Andata Ltd. ... The two Prolog dialects are virtually identical, with differences being concerned mostly with library modules and low-level programming interfaces.

  Although a Sicstus version of SPADE has not been implemented, initial experiments suggest there are no serious stumbling blocks to achieving a Sicstus port.
- Adapt interface to the FLEXlm licence manager.
- Add a TCL/TK GUI.

	System
Same GUI code across versions	yes
Same base code across versions	yes
One language solution	yes

4.3 System: Sicstus and Java

Reason:
For the reasons above, Sicstus is a very able Prolog meeting all requirements. However, the use of TCL/TK may be not be considered ideal. Sicstus also provides a mechanism to be called from within Java[9]. Unlike TCL/TK, Java provides a strong maintainable *programming* language. Further, the Java Foundation Classes (In particular Swing) make producing interfaces quite straight forward.

Problems:

- Using Java for the GUI really is using another programming language. This is different to the embedded nature of TCL/TK. Learning Java as a full programming language will be required. Java will also need to be available on the target platform. However, given that portability is key to the Java philosophy[10] this is unlikely to be an issue.

Changes to implement:

- Remove the GUI from the LPA WIN-Prolog version.
- Re-converge the separate Poplog and LPA WIN-Prolog code streams.
- Port the singe code stream to Sicstus.
- Add hooks to the Sicstus program. The complexity of these will depend a lot on the complexity of the GUI. As the GUI steps away from Console-SPADE more hooks will be required.
- Adapt interface to the FLEXlm licence manager.
- Write the Java GUI, making use of the hooks into the Sicstus program.

	System
Same GUI code across versions	yes
Same base code across versions	yes
One language solution	no

[9]This is a fully supported Sicstus feature, called Jasper. It includes a Java package to provide the Sicstus language hooks.
[10]Best illustrated by the trademarked slogan: “write once, run anywhere”.

4.4 System: LPA WIN-Prolog and Quintus

Reason:
A version for LPA WIN-Prolog already exists. As a design goal, LPA WIN-Prolog is compatible with Quintus. There is also a common GUI facility PDM that can be used across the two Prologs.

Problems:

- It is accepted, without any experiments, that programs written in LPA WIN-Prolog truly are portable to Quintus. However, LPA WIN-Prolog and Quintus are still separate Prologs. They will have different manuals, different interfaces, etc. Though portability at a technical level is expected, there may still be other less tangible barriers to overcome.
- Neither LPA WIN-Prolog or Quintus support Big-integer.

Changes to implement:

- Remove the GUI from the LPA WIN-Prolog version[11].
- Re-converge the separate Poplog and LPA WIN-Prolog code streams.
- Port the singe code stream to LPA WIN-Prolog.
- Write code to deal with the lack of Big-integer in LPA WIN-Prolog and Quintus.
- Adapt interface to the FLEXlm licence manager.
- Add the PDM GUI to this code stream.
- Test that the code stream behaves as expected on LPA WIN-Prolog and Quintus.

	System
Same GUI code across versions	yes
Same base code across versions	yes
One language solution	yes

[11]The current interface does not appear to use PDM. However, given that PDM is produced by the same people behind LPA WIN-Prolog, it would not be surprising if the existing GUI could be kept with very minor changes.

5 Discussion

A number of different system building ideas are proposed, each with different benefits. Significantly, the current situation of using Poplog and LPA WIN-Prolog (§4.1) does not appear to be ideal. As Poplog begins to age, no longer as a commercially supported product, this situation may not even be workable in the future.

Two favoured alternative system proposals both make use of Sicstus at its core (§4.2 and §4.3). While the interface is lightweight, as it is now, using TCL/TK for the GUI is quite practical. However, if the interface was to be significantly expanded, consideration should be given to the Java proposal.

The system making use of PDM and a design intention by the makers of LPA WIN-Prolog to stick close to Quintus (§4.4) is an interesting option. It is likely that this could be implemented very quickly, given that an LPA WIN-Prolog version already exists. However, the two compiler nature of this system is not attractive. A study of the practicability of using Quintus and LPA WIN-Prolog might be conducted to improve confidence in this proposal.

5.1 NuSPADE development

As NuSPADE does not yet exist, this note is mostly relevant to SPADE. However, it is important that NuSPADE remains in sync with SPADE. Thus accommodating any potential changes to SPADE should be seen as a design goal.

5.1.1 NuSPADE Short term

In the short term NuSPADE should be developed on Poplog alone. This will restrict NuSPADE to Unix and Vax users of SPADE. However, it will provide a sound platform for early development that already has a working version of SPADE.

Although working in Poplog, no Poplog specific routines should be used. It may be beneficial to occasionally port the system to Sicstus to verify this claim.

5.1.2 NuSPADE Long term

In the long term it would be advantageous to port NuSPADE to Windows. The method used is very much dependent on the method that SPADE is ported to Windows.

Assuming the current situation still prevails, it will be necessary to deal with the quirks between Poplog and LPA WIN-Prolog. Fortunately, NuSPADE will only have a text interface, so there is no GUI that needs porting[12].

6 Conclusion

The development of NuSPADE presents an ideal opportunity to reevaluate SPADE. The current combination of Poplog and LPA WIN-Prolog, while not presenting any problems at present, may not necessarily be the ideal system choice.

A few alternative system choices have been presented here. The systems with Sicstus at the core seem to be particularly straight forward. While it will require development time to adopt a new system choice, it may be worth the effort.

[12]As NuSPADE is currently envisaged. The possibility of outsourcing the development of a GUI to a Heriot-Watt student may be a worthwhile exercise.