4. Requirements

In this section, the general strategy adopted during development of extended-SPASS is described, and then the informal requirements for the software developed (extended-SPASS and the user interface) are presented, oriented to the needs of the user.

4.1 Design strategy.

From the software engineering perspective, the code base for SPASS is undocumented. Available documentation for SPASS was examined. It was determined that all of the requirements, specification, and design/implementation documentation was unavailable, although a good user manual, with a description of much of the theory behind the resolution prover is available [17, 18, 25]. The authors have attempted to make the code self-documenting, with functions organized into coherent modules, that are named for the task performed by each module (.c and .h files). Functions and their formal parameters are likewise often named to indicate their intended purpose, and code contains comments. Unfortunately, there was still a very large amount of information that had to be gleaned by reading the code and running test experiments. Importantly for the design strategy in this project, there is also no test suite provided with the software. As a consequence, the impact of any changes made to the existing code cannot be judged in terms of disruption of pre-existing functionality.

As a result of these findings a decision was made as follows.

- To produce the prototype axiomatic translation system as a 'bolt-on' module, making the functions of this module unavailable to other SPASS code.
- To make absolutely no changes to the functionality of pre-existing SPASS code.
- To make the minimum additions to pre-existing SPASS code, and confine them as far as possible to additional input to control the translation of modal problems.
- To confine the interaction of new code with pre-existing SPASS code to the minimum subset of functions required support the translational functionality. Two classes of pre-existing code that could be used were identified. First, a number of SPASS functions were determined (by reading SPASS code for the modules *term*, *list*, *symbol*, *flag*, etc) to be well defined and have no side effects in the normal functioning of the SPASS code. Second, all the SPASS functions used in the pre-existing eml-module (for translation of modal logic into first-order logic using a semantic translation) were identified.
- Having identified these two sets of pre-existing SPASS functions, to reverse engineer the minimum documentation to support understanding of their usage. This was done mainly by adding a comprehensive set of 'print' statements to the pre-existing eml-

module (for translation of modal logic into first-order logic using semantic translation), and then running and watching multiple test cases.

- To use a comprehensive set of debugging 'print' statements to confirm the correct functioning of pre-existing SPASS functions in their interactions with the new code, and to unit-test each such interaction by manual inspection of the program output. (These print statements were removed in the final released version of the code).
- To solve problems occurring in the interactions of pre-existing SPASS functions with the new axiomatic translation code, by providing additional functions and data-structures isolated *within* the new code (see for example, the local symbols cache used as a solution for the lack of stable support for dynamic symbol allocation). That is, *not* by modifying pre-existing SPASS code (since the knock-on effects cannot be judged).
- To provide a set of test problems capable of exercising each of the newly added routines, and automate the checking of the output of these problems against the expected output. This test system can be used in the current project, and also by subsequent developers, to allow extensions of the axiomatic translation module to be made while being confident that existing functionality is not disturbed. In non-object oriented systems (where coupling of code is often high) this is a very important requirement.

This strategy is simple, straightforward, and resulted in code which is itself robust, and has not yet been found to have any impact of pre-existing SPASS functionality. A similar strategy had already been partially adopted in the pre-existing eml-module for semantic translation of modal formulae implemented by R. Schmidt (in MSPASS, and later in SPASS). The alternative considered was to reverse engineer the missing documentation and test-suite. This was assessed to be too lengthy a process and not to be contributing directly towards the production of a prototype for axiomatic translation of modal logic.

4.2 Informal Requirements for extended-SPASS.

It was required that a software component be produced that is integrated into the first-order resolution prover SPASS (written in C), that would implement the ideas of axiomatic translation described in [1]. This work [1] is restricted to the axiomatic translation of modal formula with a relatively small number of additional theorems (or axioms) added to the basic axiom K. This paper suggests that more general results also apply, specifically that:

- Other combinations of modal-axioms than those covered in [1] can be subject to axiomatic translation, provided that composition of the instantiation sets arising from the modal-axioms takes place.

- The axiomatic translation can be extended to cover multi-modal examples, and other modal axioms not covered in [1].
- That the local satisfiability calculations described in [1] can be extended to global satisfiability.

The tasks referred to in the following discussion are also seen in figure 5.1. In each of the tasks, it was felt that the user should be given sufficient information for her to follow the translation in detail. This allows manual error checking during the testing phase, but should remain as part of the final released software since it helps educate the user in the intricacies of the axiomatic translation.

- *Task 3: Initialize EML module.*

  Should be transparent to the user.

- *Task 4: Read and analyze user defined .dfg file and command-line switch options.*

The .dfg file and command-line switches are the means by which the user interacts with extended-SPASS and must provide a means to :

- o Define the target modal formula to be used as input for axiomatic translation, including multi-modal formulae, with an arbitrary number of modalities.
- o Define the modal-axiom or axiom combinations to be applied during translation, including the definition of the order in which axioms are to be applied, and the mode (without or with compositional terms, etc) to be used, and including application of modal-axioms to distinct modality indices within multi-modal formulae.
- o Define a translation mode in which only modal-axiom translations or axiom-combinations that have been proven complete are possible (intended for novice users).
- o Define an experimental translation mode in which potentially incomplete modal-axiom translations or axiom-combinations are allowed.
- o Define a means by which axiomatic and classical translations of modal-axioms can be combined.
- o Define a means by which both uni-modal and bi-modal axioms can be parameterized.
- o Define a means by which certain features of the translation can be defined (translation options).

The Prolog program *ml2dfg* [referred to in 1, and provided by R. Schmidt] implemented a syntax in which most of the above requirements could be met. The set_axiom syntax described in section 5.1.4 is a very flexible implementation of a similar (but extended) syntax for SPASS. However, two other partial implementations of the above requirements are also provided, for different purposes. The in-line syntax (see section 5.1.2) for definition of modal axioms was developed to allow easy experimentation with the software, following the pattern of shorthand notation used in the theory section 2. It intersperses the definition of the modal axioms that are to be used inside the modal formula. Second, and more importantly, the command-line flags syntax of SPASS was extended to provide easy *scripted* access to the most commonly used features of axiomatic translation (including the control of translation options, see section 5.1.3). In addition, the needs of a web-based interface were best served by extending the pre-existing SPASS flags mechanism. This is particularly suited for the web-based interface because these options can all be set on the command-line that is supplied to SPASS, and are hence an ideal format to be derived from pressing web-based buttons.

- *Task 5: Reorganize Axioms and Conjectures of input problem.*

In initial stages of coding and testing, only a single axiom formula was allowed as the input problem. Since it is easily possible to re-organize any problem, either manually or with an awk-program, into this format, no need was seen to provide an alternative input mechanism. However, it emerged that the SPASS group has many 1000's of pre-existing SPASS .dfg files which are impossible to modify. Hence, it was necessary to provide a mechanism that can handle the input of problems containing an arbitrary number of axioms and conjectures. The approach taken was to collect all the conjecture formulae and axioms formulae into a single axiom formula, by automated reorganization of the input problem, and then to process that single axiom. The user needs no control over this process, but is should be informed of modifications that have taken place.

- *Task 6: Standardize input problem.*

In ref [1] the axiomatic translation is described in terms of only $\Box$, $\neg$, and $\wedge$ connectives (and $\perp$). This is done to simplify the translation process, and while it is not strictly necessary, is attractive since the implementation is likewise simplified. The user needs only to be informed of the transformations that have been made, without any control over them. It was thought prudent to include some other transformations of the input formula (for example, collapse of some modal tautologies) since these simplify the input formulae greatly.

- *Task 7: Perform Axiomatic Translation of input problem.*

The axiomatic translation must be implemented as defined in [1]. Several optimizations of the axiomatic translation are recommended in [1] (see for example section 2.1 and 2.2.4). These should be implemented along with other simple obvious enhancements such as elimination of duplicate terms (since these slow down the resolution process). The user needs some control over the translation process, for example enabling / disabling the inclusion of shortcut terms (see section 5.1.3).

- *Task 8: Encode modal axioms for the input problem, considering each Axiom in strict order.*

It must be possible to :

  - o Specify the modal-axioms to be included in the axiomatic translation of the target formula. (At a minimum, the user must have access to at least the range of modal-axioms as defined in [1]).
  - o Specify the modal-axioms translated in both the combinations described in [1], that have been proven complete, and in arbitrary combinations with the order in which the axioms are translated being under user control.
  - o Specify the mode applied during translation of the modal-axioms, for example choosing to perform the translation with and without compositional terms.
  - o Specify translation of modal-axioms in classical (semantic) mode rather than axiomatic mode, being able to mix translation of different axioms in arbitrary combinations of classical and axiomatic modes.

- *Task 9: Set precedence.*

The precedence assigned to the new predicate symbols must be considered. It is often necessary to set this precedence correctly, overriding the default precedence set by the SPASS resolution prover, in order to ensure decidability. Extended-SPASS should be capable of correctly setting the precedence, and the user should be able to override these using their own values using the pre-existing (`set_precedence` syntax).

- *Task 10: Pass collected translated terms to SPASS.*

The terms that are passed to the SPASS resolution prover should be reported to the user (the `Final Term`). It is worth ensuring that the format of these terms be suitable for cutting-and-pasting into another .dfg file. This is also important for testing by Jasper (see section 7.1) which captures this `Final Term.`

In addition, the format of the new predicate terms produced must be considered. First, SPASS has a restriction on the length of the names of predicate terms, defined by `symbol__SYMBOLMAXLEN`. (Note: violation of this restriction raises no errors, although the result of calculations will be unreliable). This restriction should be honored in the names produced by the axiomatic translation, producing informative names where possible (of the format, e.g. $Q_{box\_r\_not\_p}$), and simple cryptic names (of the format Q_{int}) elsewhere. Second, some potential applications of axiomatic translation (for example, the use proofs to construct modal tableau [1]) benefit from use of a uniform format for the predicate naming scheme, Q_{int}. Where a cryptic name has been used, the user should be informed of the informative name that has been displaced.

- *Task 13: Clean up memory allocated to data, etc in EML module.*

It was required that the new code added to SPASS should not lead to memory leaks. The criterion was that there should be no more leaks detected by Valgrind [`http://valgrind.org/`] that would be expected for the original SPASS 3.0. This is quite simple to achieve, since most of the memory allocated is tracked by the `memory` module in SPASS, and automatically freed by statements at the end of `top.c` at exit.

4.3 Informal Requirements for the User Interface.

The user interface available for SPASS was a little basic (see WEBSPASS http://www.cs.man.ac.uk/schmidt-bin/web-mspass.cgi), and so an upgraded interface was also required. The interface should be presented as a user-friendly web-based interface, either for use on school or personal web-servers. The prototype need only be suitable for use in the Firefox browser. (This provides a very wide range of supported platforms [`http://www.mozilla.com/en-US/firefox/`]). The interface should allow the user to perform the following tasks, in an integrated and coordinated fashion :

- Allow the user to enter and edit a plain-text SPASS input file, with all the normal editor functionality (copy, edit, paste, etc).
- Allow the user to execute SPASS using the input .dfg file drawn from the editor.
- Provide an interface for execution of command-line switches, supported by a selectable list of available switches (both from SPASS, and those added to support axiomatic translation in extended-SPASS), indicating their default values.
- Allow the user to view the results of SPASS execution.
- Allow the user to view the input file, with associated line numbers, on the same page as the results of execution (this is useful for debugging scripts, where errors are reported relative to a particular line number).
- Provide a history mechanism to record the results of execution of SPASS, with all of Script, Command Options and Output, allowing recording and viewing of multiple such results sets.
- Allow the script to be copied back from the history mechanism to the main editor.
- Provide read only sample scripts, that can be chosen, viewed, and copied to the main editor, and provide a large repository of sample scripts to aid the user.
- Allow the user to gain Help both from the parent web sites, and from other users via a Wiki interface.

- Provide some help (for example, as tool-tips) for cryptic menu options, and provide feedback when error conditions are detected.

4.4 Informal Requirements for a test suite.

SPASS itself has no publicly available test suite. Hence, it difficult to prove that the new functionality added to SPASS during this project does not disturb pre-existing functionality. As mentioned above (section 4.1), was decided to avoid disturbances of the pre-existing SPASS functions by adopting a defensive programming strategy. However, the situation is different for the test suite associated with the new axiomatic translation code. It is required here to provide a test suite to:

- Exercise as much of the optional functionality (related to axiomatic translation) in extended SPASS as possible, providing illustrative examples of input in as many modes as possible.
- Test the results of axiomatic translation (as reflected in the reported `Final Term`) against expected results, and assess the results automatically.
- Test the results of resolution (satisfiable or unsatisfiable) of the axiomatic translation of *well-known formulae* against expected results, assessing those results automatically.
- Use a wide range of examples problems to access the viability of the axiomatic translation technique, for example in terms of length of proof, and speed of calculation.

All these requirements are met by the software delivered during this project.