How to treat GUI Options in IT Technical Texts for Authoring and Machine Translation

Johann Roturier and Sabine Lehmann

Symantec Limited and acrolinx GmbH

Abstract

This paper focuses on one aspect of controlled authoring in a localization and Machine-Translation context: the treatment of GUI options, which abound in the procedural sections of IT technical documentation. GUI options are technical terms that refer to the Software User Interface. The length and complexity of GUI options is a major problem for numerous NLP tasks, including MT. GUI options which have not been identified by NLP applications typically lead to erroneous analyses of sentences. However, few authors have focused on the identification and tagging of GUI options in IT documentation. This paper delineates an approach based on a controlled language checker that benefits both the human authoring process and Machine Translation.

DOI: 10.1075/jial.1.02rot
1 Introduction

1.1 Example of a Controlled Language Deployment

In the last three decades, publishers of technical documentation have often attempted to improve the comprehensibility of their technical source content by implementing a Controlled Language (CL). Nyberg, E., Mitamura, T., & Huijsen, W-O. (2003) give the following definition of a CL: "A CL is an explicitly defined restriction of a natural language that specifies constraints on lexicon, grammar, and style". For instance, the use of precise CL rules has proved useful to improve the usefulness of machine translation (MT) output (Rychtyckyj, 2002; Takako et al., 2007).

Symantec, a software publisher specialised in security and availability solutions, has been using a CL approach since 2005 to maximize the effectiveness of a third-party, rule-based MT system within its translation workflow (Roturier, 2006). To ensure that source English content complies with style guides and MT-specific rules, Symantec writing teams use a customised version of the acrolinx IQ™ checker, which is a QA tool for authoring documentation. Writers check their new XML topics using the acrolinx IQ™ plug-in for the XMetal editor to make sure that their documents comply with approved terminology, grammar, and style rules. Before translating these topics, the localisation department can examine reports to determine whether specific issues will have an impact on overall MT quality. This paper focuses on one aspect of controlled authoring: the treatment of GUI options, which abound in the procedural sections of IT technical documentation.
1.2 CL and GUI Options

The following example shows a sentence containing two GUI options ("Extract File" and "Restore from"). The example shows that capitalization is not always helpful in identifying GUI options and that the boundaries of GUI objects can be difficult to determine:

At the **Extract File** dialog box, in the **Restore from** box, type the following where `<CD-ROM drive>` is the drive letter of your CD-ROM drive.

Senellart et al. (2001) use the terms "token name" to refer to the actual button name, alert message or dialog box content, while they use the term "token identifier" for the actual GUI object with which end users interact. In this paper the terms GUI option (example: "Extract File") and GUI object (example: "dialog box") will be used to avoid any confusion. From a CL perspective, semantic tagging is sometimes mentioned as a means to remove ambiguity from source input (Bernth & Gdaniec, 2001; Nyberg et al., 2003). However, few papers have focused on the identification and subsequent tagging of GUI options in IT documentation. While Senellart et al. (ibid) focused on the identification and handling of GUI options from a translation perspective, this paper introduces an approach that benefits both the authoring process and the translation process.

1.3 Paper Structure

This paper first provides an overview of the challenges posed by GUI options when deploying NLP applications (such as a CL checker or an MT system). Section 3 discusses two of the strategies that can be used to address these issues. Section 4 describes a novel identification solution, which is concretized by the presentation of two style rules in section 5. Section 6
Johann Roturier & Sabine Lehmann elaborates the results of this solution. Finally, section 7 suggests additional benefits of this approach.

2 Challenges Posed by GUI Options

GUI options not only pose challenges for NLP applications, but are also highly important from an authoring perspective. These two issues are discussed in the following subsections.

2.1 NLP Applications

During the deployment of the CL checking technology presented in section 1.1, some technical writers reported that certain rules returned false positives and created translation problems because of the presence of GUI options in source content. Examples of such problems are presented below:

Sentence length: The rule states that no more than 25 words should be used in a sentence. This rule was sometimes violated because long GUI options were handled as separate words, as shown in the following example:

On the Protection Manager Servers page, select the Enforcer group in the View Servers pane and click Edit Group Properties to display the Settings dialog box.

If the CL checker does not deal specifically with the GUI options, this sentence contains 26 tokens. However, if all of the highlighted GUI options are tokenized as single entities, the length of the sentence is reduced to 21 tokens, which prevents the false alarm for sentence length.

Grammar and style rules: False positives also occurred with regard to grammar and style rules. As an example, the grammar rule governing subject-verb agreement returned false positives because of POS tagging errors created by GUI options. As shown in the example above, GUI
options can include verbs used in unusual positions – such as "View" following an article, or "Edit" following a verb.

**Term extraction:** GUI options were often presented as candidate terms - in a truncated form or with the following GUI object as part of the candidate term. However GUI options should be completely excluded from term extraction.

**POS assignment:** Even though GUI options may contain prepositions or even punctuation marks, they correspond to a proper noun from a syntactical point of view. Ideally they should be assigned only one POS, (such as NNP from the Penn Treebank Tagset).

**Translation:** From a translation or localisation perspective, GUI options have to be translated according to the translation found in a software glossary, as the following example shows:

*To start the program, click Start.*

The capitalized word "Start", should be translated as "スタート" in Japanese, not "開始". In a translation workflow that uses an RBMT system, one solution might be to encode "Start" > "スタート" as a User Dictionary entry, but this entry may have side effects (for instance, if "Start" is used at the beginning of an imperative sentence). On the other hand, if the MT system is not fine-tuned to handle GUI options, the resulting translation output might be unusable, as shown in the following sentence and its corresponding French output (translated using a Systran 5.05 engine with the Computer/Data-Processing domain enabled):

*Enter information in the Connect to a Media Server dialog box.*

*Écrivez l'information dans le connecter dans une boîte de dialogue de serveur multimédia.*
To work around these problems, contextual information is required around the GUI option to ensure that UD entries are only used in specific contexts. All these problems occur because the NLP applications do not treat GUI options as single entities. GUI options should be treated as a "unit" whatever their length is. However, this approach presupposes that the NLP applications can identify them as such.

2.2 Authoring Perspective

From an authoring perspective, GUI options should match the strings that are present in the software itself. However, software strings may be updated by software developers on a regular basis, so it is sometimes difficult for technical writers to reflect the changes in their documentation. Typical discrepancies between software and documentation include capitalization, hyphenation, and spelling. If technical writers do not extract GUI options from their documentation to conduct regular cross-references with software strings (using semantic tagging), these discrepancies are bound to go uncorrected. This ultimately affects the end-user's experience and subsequent translation process. The discrepancies may even persist when a structured XML documentation model is used (such as DocBook or DITA) because not all GUI options are necessarily tagged.

3 Treating GUI Options as Single Entities

This paper proposes a two-fold solution to treating GUI options as single entities by using semantic tagging and token class encoding.

3.1 Semantic Tagging

In the typical information development work-flow described earlier, writers must ensure that their XML topics comply with a subset of the DocBook XML DTD (Walsh and Muellner,
How to treat GUI Options in IT Technical Texts

2006). The full DocBook DTD contains various tags related to GUI options (guibutton, guiicon, guilabel, guimenu, guimenuitem and guisubmenu). Yet to date, the only tag in use has been <guimenuitem>, which sets off some of the GUI options that end users have to interact with. This standard was initially designed to highlight GUI options with visual formatting in the final deliverable - usually a PDF file or a help file. The final deliverable can be obtained by transforming XML topics using an XSL-FO technology. Figure 1 illustrates this, in which the following XML sentence has been transformed:

*Click* <guimenuitem> Yes </guimenuitem> *to install the Enterprise edition.*

![Click Yes to install the Enterprise edition.](image)

*Figure 1:* Rendering of a guimenuitem element

When this <guimenuitem> tag is present in the source text, an MT engine can be configured to handle it in a specific manner. For instance, the use of XSL files can achieve this objective when using certain versions of Systran software (Senellart & Senellart, 2005). The next section focuses on how the acrolinx tokenizer handles GUI options.

### 3.2 GUI Options as Tokens

The acrolinx tokenizer has three main tasks. The first is to split incoming text into "tokens". Tokens typically correspond to words. In general, it is useful to define longer sequences as tokens if they form a meaningful unit. Examples would be compounds such as "vice versa", but also more text-specific units like GUI options. The second task is to assign "token classes" to these tokens. The token class is the name associated with the type of token. As an example, the token "Test" has the token class "FirstCapitalWord", "123" the token class "Number". Finally, the tokenizer must split the incoming text into sentences (segmentizer). In
applications like the acrolinx IQ™ checker, which deal with huge amounts of genuine and unstructured data, the tokenizer has a prominent status. The errors occurring at that level can engender mistakes that might be visible to the technical writers. It is therefore important to treat GUI options as one token in order to ensure that the issues discussed in section 2.1 can be easily addressed.

4 Identifying GUI Options

4.1 Using Xpath and Regular Expressions

GUI options can be extracted from source documentation with a basic XPath expression (such as //guimenuitem or //guilabel if a DockBook DTD is used). However, untagged GUI options may require a manual (and time-consuming) extraction process if they are to be cross-referenced with software strings (as suggested in section 2.2). This manual process may, however, be alleviated if a regular expression-based approach is used, as shown by the following example:

```regex
\b[\w\d]+\b\[IiOo]n |\[Tt\]o |\[Ch\]eck |\[Ff\]rom |\[Uu\]sing )the \){[A-Z][^,<>:.\]+}\b(?:drop[- ]?down list|drop[- ]?down box|option|dialog box|field|drop[- ]?down menu|menu|tab|box|window|dialog|check box|text box|panel?\|folder\|list\|icon\|login prompt)\b[,\.]
```

Greediness is the major problem of this approach. With this approach, the substring "Advanced tab of the Enforcer Settings" is extracted from the following string:

*On the Advanced tab of the Enforcer Settings dialog box,*

Section 4.2 discusses how a two-pronged strategy can deal with this greediness issue.
4.2 Using a Customised Solution Based on the Acrolinx IQ™ Technology

Token class definition based on context: The tokenizer allows the definition of rules based on the context. A token class can be defined on the tokens which precede and follow the token. With the token class GUI option in mind, such an approach presupposes relatively consistent texts with respect to the use of GUI options and a good knowledge of the syntactic context in which the GUI options might occur. As shown in section 4.1, these conditions were met. The context which had initially been used for the solution based on regular expressions was also used to define the rules. A simple example of such a rule is shown below:

```plaintext
UIStringsSymantec ->
  precontext(seq(OR("Under"), +(WhiteSpace)))cend,
  postcontext([Comma])cend,
  OR([FirstCapitalWord], [FirstCapitalWordMixed],
      [CapitalWord], [CapitalWordWithHyphen], [OperatingSystem],
      [TrademarkSymantec]);
```

The rule expresses that the token class name is "UIStringsSymantec". The class must be preceded by "Under" and followed by a comma. The token class itself has to be one of the following token classes (which have been defined on an upper level): FirstCapitalWord, FirstCapitalWordMixed, etc. The sentence "Under Options, click OK" is then tokenized as follows:

```plaintext
Under FirstCapitalWord
Options UIStringsSymantec , Comma
click LowerWord
OK CapitalWord
```
The GUI option "Options" is correctly recognized and tokenized as "UIStringsSymantec". Token class definition based on elements: The preceding approach had one significant drawback. The rules were very efficient for GUI options which were not tagged with the <guimenuitem> presented in section 2. But for the GUI options that were already tagged, the tokenization turned out to be partially redundant: why write complex rules to identify the GUI option if that information is already available via the element <guimenuitem>? However at that stage of the project, the tokenizer was not able to access any information about the element. A subsequent version of the tokenizer extended the formalism and made this information available to the rules.

With that extension, the following simple tokenizer rule was written:

```
GuiMenuItemSymantec ->
    seq((OR([LowerWord],[],...)): matchContext("guimenuitem");
```

The rule expresses that the token class name is "GuiMenuItemSymantec" and that it matches any number of token classes occurring in the context <guimenuitem> (the expression in the OR parenthesis has been simplified here).

The sentence "Click <guimenuitem>Central Quarantine account</guimenuitem>" is then tokenized as follows:

```
Click FirstCapitalWord
    Central Quarantine account GuiMenuItemSymantec
```

There is no interaction between the two token classes "UIStringsSymantec" and "GuiMenuItemSymantec": if a token is a potential match for both classes, the latter takes precedence. This ensures that only tokens with no predefined tagging are defined as
"UIStringsSymantec". In principal the token class "UIStringsSymantec" should not occur if all the GUI options were correctly tagged, but Symantec text does not always apply this element.

5 Developing two style rules

By introducing these two token classes, companies A and B solved the problems described in section 2.1: they reduced false alarms and improved the accuracy of POS assignment. Additionally, the two token classes help address two important problems with respect to the use of GUI options in IT texts. The first problem is that all of the GUI options which should be tagged with <guimenuitem> may not be actually annotated as such. The second problem is that some of the <guimenuitem> tags may be wrongly placed since the GUI objects should be outside of the element.

5.1 Rule A: add_guimenuitem_tags

We decided to add a style rule to the acrolinx IQ™ rule formalism (Bredenkamp et al., 2000) which would mark potential GUI options where the <guimenuitem> tag is missing. However, this corresponds exactly to the content of the token class UIStringsSymantec, since this token class tries to identify potential GUI options. Developing the style rule "add_guimenuitem_tags" basically came down to marking the string of the token class "UIStringsSymantec":

```plaintext
@uistringsSymantec ::= [ TOKCLASS "UIStringsSymantec" ];
TRIGGER(80) == @uistringsSymantec^1 -> ($uistring)
            -> { mark: $uistring; }
```

Rule B covered the exceptions to the rules.
5.2 Rule B: Exclude_UI_Objects

Rule B was developed based on the fact that the token class "UIStringsSymantec"
sometimes slightly overgenerated due to a greedy match: UI objects such as "check box", "dialog
box", "tab", "window", etc. were wrongly tokenized as part of "UIStringsSymantec". As an
example:

"Inventory tab in the Properties window"

was tokenized as "UIStringsSymantec", although in fact it had to be tagged with two
<guimenuitem> tags:

  <guimenuitem>Inventory</guimenuitem> tab in the
  <guimenuitem>Properties</guimenuitem> window

Rule B can thus be seen as an extension of Rule A, since it flags the token class
"UIStringsSymantec" (as does Rule A), but only those cases which contain UI strings.

@exception ::= [ TOK " (click|dialog
box|field|icon|list|menu|OK|pane|panel|select|tab|window)"
    TOKCLASS "UIStringsSymantec" ];
TRIGGER(80) ==
@exception ^1
-> ($exception)
-> { mark: $exception;}

The rule makes writers aware that if they add the tag <guimenuitem>, the GUI object
("tab","window") must be excluded.
6 Evaluation data and results

6.1 Evaluation Data

To evaluate the two rules described in section 5, we created an evaluation corpus of XML documents (in the region of 100,000 tokens). We used two sets of product documentation (pertaining to the security and availability domains) to create this corpus. These documents were not final documents. They had been developed by various product writing teams but still had to go through an editing phase. These documents contained 964 guimenuitem elements, following the standard usage presented in section 3.1.

6.2 Evaluation Setup

Rule A and Rule B were added to a development rule set and used with the acrolinx IQ™ Batch Checker to extract occurrences of untagged GUI options. Two reports were then generated and post-processed to extract contextual information for verification purposes. Due to the size of the corpus, we decided to focus on precision and use a pragmatic approach to evaluate recall: after calculating precision scores for each rule, we took a sample of the corpus to determine the types of constructs that were not found by the rule. The second part of section 6.3 describes these constructs.

6.3 Evaluation Results

Rule A flagged 311 occurrences of untagged GUI options, and Rule B reported 8 occurrences of GUI option combinations. A careful examination of the data showed that the precision of each rule was as follows: Rule A: 304/311 (97%) and Rule B: 7/8 (87%). The precision of Rule A was high (97%) despite the high number of varied structures that were found in the data – in the following examples, the GUI option appears in bold:
Johann Roturier & Sabine Lehmann

- Single capitalized noun: On the Action menu, click Connect.
- Series of capitalized nouns (including acronyms): On the Add DHCP Server dialog box, enter the following information for the server that you are adding:
- Series of capitalized nouns with unusual syntax: lists the options in the Media Verify dialog box.
- Series of lowercase words following a capitalized word: To update the signature file, select Download signature file, and specify the download URL.
- Series of lowercase words (including prepositions) following a capitalized word: In the Edit Server Groups dialog box, ensure the group you want to edit is selected in the Choose a group to edit option.

In this last example, the preposition "to" introduces a complement of the noun "group". Without deeper linguistic information, however, it would be difficult to identify purposive structures accurately. Rule A returned seven false positives. These can be classified into the following categories:

- Missing source words (it should read "dialog under the Advanced Settings tab"): The number of times the user can postpone a download and for how long is controlled by the options on the Host Integrity Setting dialog Advanced Settings tab.
- Unusual pre-modifier for the word "dialog" (the word "editor" indicates a type of dialog): In the Custom Requirement editor dialog, select the node in the tree where you want to add the action.
- Greediness: Then, select Incremental from the Backup Method tab.
Three other occurrences of greediness issues were found in the data. Yet this problem would not have a significant impact in a typical authoring environment because technical writers would be able to reduce the scope of incorrectly flagged GUI options. If this approach was used to transform source text in an automatic manner without human intervention, however, this problem would be more serious. The only invalid flag for Rule B occurred in the following sentence:

*To view subnodes, double-click Jobs or click the icon next to the node.*

This sentence was missing a standard `<guimenuitem>` tag around the word "Jobs", which explains why Rule B triggered in this particular context. From a recall perspective, Rule A did not catch GUI options in the following contexts:

- GUI option used as post-modifier: *The option Enable appears only if the robotic library is disabled.*

- GUI option's context spanning over multiple segments: *You can select all types of jobs, or any combination of backup, restore, media rotation, or utility jobs. All, which lets you view logs for all jobs.*

- GUI option containing two words separated with a slash: *You can remove the administration console from a Windows computer by using the Windows Add/Remove Programs option.*

- List of GUI options with which the user may not interact: *The Jobs node contains subnodes that are called All Jobs, Scheduled Jobs, Active Jobs, Job Logs, and Schedule View.*
• Unusual post-modifiers (such as "filter" or "node"): For example, suppose you select a certain user in the Users filter and Backup under the Job type filter. The Storage Devices node lets you view the following:

The last category shows that the rule relies heavily on the use of standard terminological modifiers. Nonetheless, a minor rule update would improve recall when dealing with texts originating from different subdomains. Overall, these results suggest that the high precision of the rule would not have a negative effect on the productivity of technical writers and could be implemented in a production environment. Section 7 will report on how writers can use these two rules in an effective manner.

7 Application and further benefits

The application of the approach presented in section 5 and evaluated in section 6 can be implemented in two ways. It can be implemented in an interactive manner within an XML editor supported by acrolinx IQ™, as shown in Figure 2:

![Figure 2: A violation of Rule A is reported in the XMetaL editor](image)

In Figure 2, a contextual menu warns the user that the GUI option "Applied Host Integrity Policy Changes" should be tagged with <guimenuitem>. The acrolinx IQ™ Batch Checker can also be used to obtain a report on the number of untagged GUI options in a set of documents. This
approach is extremely useful when dealing with legacy documentation that has never been localised (to estimate how much work will be involved to handle GUI options).

While all technical writers may not necessarily see the benefits of tagging content that will not be formatted in final deliverables, the following four points should be kept in mind. First, without tagging GUI options, it is difficult to ensure consistency between software and documentation. Second, the final appearance of deliverables may change in the future. It is much easier to make an update to an XSLT file than to edit a set of source files manually. Third, translation requirements may change in the future. Again, it is easier to make an update to an XSLT file used for translation purposes than to ask translators or post-editors to spend time checking for GUI options. Regardless of whether MT is used in the subsequent translation workflow, the use of semantic tagging is useful because it also gives translators indications about the boundaries of GUI options. Finally, this approach may be seen as a first step towards the automatic tagging of content. While tags currently have to be manually inserted one by one, this will ideally change in the future to improve the productivity of writers.

8 Conclusion

This paper has presented a novel approach to identify GUI options with a view towards helping writers use semantic tagging in English technical documentation. The evaluation of this approach showed that high accuracy can be achieved when source documents are already well structured and when standard terminology is used despite the complexity of GUI options. The high accuracy of the rules should allow for a smooth introduction within a production environment, with benefits for both source consistency and translation efficiency. The rules indeed solve the challenges discussed in section 2. While this approach focused on data
originating from a single software publisher, it may also benefit other stakeholders from the IT industry. Software publishers rely extensively on online knowledge bases to help their customers solve problems, so having a consistent way to identify and treat GUI options improves the experience of end users.
How to treat GUI Options in IT Technical Texts

References


Johann Roturier & Sabine Lehmann


