CS4311 Fall 2001

HATS Graphical User Interface Software Requirements Specification

Version <1.5>

11/01/2001

Document Control

Approval

The Guidance Team Dr. Roach, Dr. Gates and Mr. Leyva, and the customer, Dr. Victor Winter, shall approve this document.

Document Change Control

Initial Release:	5/1/2001
Current Release:	HATS GUI SRS 1.5
Indicator of Last Page in Document:	♦
Date of Last Review:	11/01/2001
Date of Next Review:	None scheduled
Target Date for Next Update:	None scheduled

Distribution List

This following list of people shall receive a copy of this document every time a new version of this document becomes available:

Customer:	Dr. Victor Winter
Guidance Team:	Dr. Ann Gates, Dr. Steve Roach, Francisco Javier Leyva
Software Teams:	Creative Team, Omega Team, Porcelain Team, Resilient Team, Suzaku Team

Change Summary

The following table details changes made between versions of this document

Version	Date	Modifier	Description
0.1	5/5/01	F. Leyva	Combined SRSs from Software Teams
0.2	5/15/01	F. Leyva	Edited to remove redundancies
0.3	5/22/01	S. Roach	Organized, reduced volume
0.4	5/23/01	F. Leyva	Use case revisions, diagrams
0.5	5/30/01	S. Roach	User interface descriptions
0.6	6/07/01	S. Roach	Added Scenarios for Use Cases
0.7	6/13/01	F. Leyva/S. Roach	Added More Scenarios for Use Cases
0.8	6/14/01	F. Leyva/S. Roach	Revisions to Scenarios for Use Cases
0.9	8/1/01	S. Roach	Modified HATS specification
1.0	8/3/01	Roach/Leyva	Version 1.0 sent to Winter for approval
1.1	8/15/01	Roach	Communication protocol revised
1.2	8/21/01	Roach	Protocol reviewed by Dr. Winter
1.3	8/27/01	Roach	SDT search clarified by Dr. Winter
1.4	10/01/01	4311 class	Typographic errors found by CS4311, HATS-SML interface clarified by Dr. Winter.
1.5	11/01/01		

TABLE OF CONTENTS

LIST OF TABLES

List of Figures

1. Introduction

1.1. Purpose and Intended Audience

The purpose of the Software Requirements Specification (SRS) is to clearly and precisely describe the requirements of the software system being developed, hereafter referred to as the HATS-GUI. The HATS-GUI is a Graphical User Interface (GUI) for the High Assurance Transformation System (HATS). This document shall serve as a reference guide to the developer for software design, implementation, and maintenance.

The SRS divides the system requirements into two major sections, behavioral requirements and non-behavioral requirements. Behavioral requirements describe the interaction between the system and its environment. Non-behavioral requirements are the ones that relate to the definition of the attributes of the product as it performs its functions.

The intended audience of this document is Dr. Winter, the Guidance Team, and the software development team. The SRS is an agreement on requirements between these parties regarding the software to be developed.

Text give in gray highlight refers to changes since the previous version of this document.

1.2. Scope of Product

The High Integrity Software (HIS) program at Sandia National Laboratories developed HATS in order to utilize transformation-based software development. Transformations are a well-known formal method for developing software. The intent of HATS is to develop software via high assurance transformations that have been proven to preserve the semantics of the transformed programs. Such a system benefits the high assurance software community by providing a tool with which to develop software.

The HIS program has requested the implementation of a platform-independent GUI to simplify user interaction with HATS. The system shall provide to the user the ability to prepare applications, utilize HATS to apply transformations, and display and manipulate the output resulting from execution of transformations. The development of a platform-independent GUI will give the members of the transformation and research community a chance to experiment using HATS. This will increase the use of HATS and will provide the HIS program with feedback from the members of the transformation and research communities that will aid in the development of this tool.

A prototype of this system has been developed, but is inadequate due to its lack of portability and its inability to display outputs adequately. The goal of developing the HATS-GUI is to make the system more user-friendly, make the user interface platform independent, and to increase the use of the HATS system.

1.2.1. Definitions

Table 1 below lists the definitions used in this document with respect to the HATS-GUI. The definitions given below are specific to this document and may not be identical to definitions of these terms in common use. The purpose of this section is to assist the user in understanding the requirements for the system.

Table 1: Definitions used in SRS

TERM	DEFINITION
Abstract Tree	See “Tree.”
Active window	The window in a graphical user environment that is currently accepting keyboard and mouse input.
Ancestor nodes	The ancestor nodes of a node *N* is the set of nodes **N*** such that every element of **N*** is either the parent of *N* or the parent of some node in **N***.

Application	A set of files describing a problem domain and a (possibly empty) set of transformation language programs, target files, and output files. An application is a stand-alone entity and contains all the data sets required to perform and view transformations. A set of associations and user-selected configurations is also part of the application.
Associations	Relations between files in an application. For example, a file may be mapped to the file used to generate this file or a file (or program) used to edit the file.
Child node	A node that has a parent node. The child relationship is the inverse of the parent relationship. (See *Edge*.)
Collapse	To hide the descendants of a displayed node.
Color Scheme	The selected colors and shapes used to represent different types of displayed nodes in a displayed graph of a syntax derivation tree.
Configuration	The set of characteristics related to an application that controls the behavior of the HATS-GUI. Table 6 lists the elements of a configuration.
Copy-and-paste	An operation that allows a user to copy text from one process into another process. Typically, copy-and-paste operations transfer data to and from an active process into an operating system buffer.
Core	A set of files consisting of one lexical specification file, one grammar file, and zero or one user-defined libraries.
Cursor	A display element in a graphical user environment that marks a location in a window or buffer.
Descendant nodes	The descendant nodes of a node *N* is the set of nodes **N*** such every element of **N*** is either a child of *N* or the child of some node in **N***.
Directory	A special file in a hierarchical file system that contains either files or other directories. Directory is synonymous with folder.
Displayed graph	The visual representation of an SDT or a part of an SDT. A displayed graph is a set of displayed nodes and the arcs connecting those nodes.
Displayed node	The visual, graphical display of a node. Every displayed node has a corresponding SDT node. Not every SDT node has a corresponding displayed node.
Domain	Language over which transformations take place. The domain is defined by the files in the core (see core).
Drag	An operation using a mouse or other pointing device where the user presses the pointing device button and moves the pointing device while the button is pressed.
Edge	A directed relation between two nodes in a tree. If edge e is a relation from node N_1 to node N_2, we write $e(N_1,N_2)$. N_1 is the parent of N_2. N_2 is the child of N_1.
Expandable node	A leaf node of a displayed graph whose corresponding tree node has immediate descendants.
External editor	Any editor available on the host OS that can be started as a separate process by the HATS-GUI.
File name	The name of a file in a file system. A file name consists of a path name and a unique file name. The root name of a file name is a file name without a path name and without a file extension.
File extension	A set of characters in a file name including and following the right-most period of the file name.
Folder	See *Directory*.
Grammar file	A file representing a set of rules that define the language of the domain over which transformations are defined.
High Assurance Transformation Systems (HATS)	The transformation system developed by HIS at Sandia. The system includes both the HATS-SML programs and a user interface. Together, these provide services that allow parsing and running transformation language programs.
HATS-SML	A set of service programs that provide the parsing, transforming, and pretty-printing functions of HATS.
Host OS	The operating system under which HATS-SML and the HATS-GUI are running.

Java	The programming language Java.
Java Virtual Machine (JVM)	An abstract computing machine that is responsible for the hardware and operating system independence of the Java platform.
Keyboard cursor	The cursor associated with the keyboard. This cursor may be moved by using keystrokes such as the arrow keys, the *tab* key, or the *enter* key.
Leaf node	A node in a displayed graph that does not have any displayed descendant nodes.
Level	The number of edges that must be traversed in order to reach a node in a tree starting from the root.
Lexical specification	A file defining the tokens to be recognized by a transformation language.
Linux	The operating system Linux, a Unix-like operating system available on most platforms.
Locator box	Rectangle drawn around a sub-tree in a navigation window indicating the sub-tree displayed in the SDT display window relative to the entire SDT.
Mouse cursor	The cursor associated with a pointing device such as a mouse. This cursor may be moved by moving the pointing device.
Navigation window	A displayed window containing an abstract representation of an entire SDT. The intent of the navigation window display is to facilitate navigation of large SDTs.
Node	An element of a tree or a displayed graph. Nodes in SDTs contain text tokens derived from parsing input strings and applying transformations. Nodes in displayed graphs (displayed nodes) are visual representations of nodes in SDTs. Nodes in SDTs should not be confused with displayed nodes.
Parent node	A node that has at least one child node. The parent relation is the inverse of the child relation. (See *Edge.*)
Parse	To generate an SDT from a non-SDT (i.e. string) input file such as a transformation language program or a target.
Parser	A program that parses either transformation language programs or target programs.
Parsed transformation language program	The SDT representation of a transformation language program. This is the form used by HATS. A parsed transformation language program is said to be parsed.
Path	A text string that represents the sequence of directory names and a file name that uniquely specifies the location of a file in a file system.
Platform	The combination of hardware and operating system software that comprise a computing system.
Pretty-print style file	The input file that specifies how to display the pretty-print format of the target language.
Pretty-printed text	A text string output by HATS representing the formatted result from applying a transformation language program to a target file. Formatting is done according to the pretty-print style.
Problem Domain	See domain.
Program SDT	An SDT generated by parsing a transformation language program.
Proper sub-tree	A tree T_1 is a proper sub-tree of a tree T_2 if T_1 is a sub-tree of T_2 and T_1 has fewer nodes than T_2.
Relative path	A path that starts at a directory other than the root directory.
Root node	The first and initial element in a tree represented by nodes.
Scroll	To move information across a display screen as if unrolling a scroll.
Smallest well-formed tree	Given a tree *T* and a set of nodes **N** in *T*, the smallest well-formed tree of **N** in *T* is a tree *S* such that *S* is a sub-tree of *T*, every element of **N** is in *S*, every descendant of the root of *S* in *T* is also an element of *S*, and *S* has no proper sub-trees that are smallest well-formed trees of **N** in *S*.

Sub-tree	A tree T_1:<N_1,E_1> is a sub-tree of a tree T_2:<N_2,E_2> if • $N_1 \subseteq N_2$, • $E_1 \subseteq E_2$, • If R is the root of T_1, $\forall (n \in N_2)((n \text{ is a descendant of } R) \rightarrow n \in N_1)$, and • $\forall (e \in E_2)\ (e(n_a,n_b) \wedge n_a \in N_1 \wedge n_b \in N_1) \rightarrow e \in E_1$.
Syntax Derivation Tree	A tree where nodes represent either the left-hand side of grammar productions used to parse a string or terminal symbols in the grammar. The children of nodes represent the right-hand side of the production used to parse the node.
Target file	The program to which the High Assurance Transformation System applies a transformation.
Target Program	A target file that contains a computer program.
Target SDT	An SDT generated by parsing a target file.
Terminal node	A node in a tree that has no descendants.
Time Stamp	The date and time of the most recent modification of a file. It is assumed that the Host OS keeps the time stamp for each file.
Tlp_parsed file	See parsed transformation language program.
Token	A text string used as a single unit, for example, a word. Tokens are described by the lexical specification.
Transformation	A rule that specifies the translation of one string into another string. HATS applies transformations to target files.
Transformation Problem	A target file in an application.
Transformation language program	A sequence of transformations along with constructs to control the application of the transformations.
Transformed Program	The program that results from applying a transformation language program to a target file.
Tree	A tree T<**N,E**> is an abstract data type composed of a set of nodes **N** and a set of directed edges **E**:**N**x**N**. An edge E_1 from node N_1 to node N_2 associates the parent node (N_1) with the child (or immediate descendant) node N_2. Each child node has exactly one parent. Each tree has exactly one distinguished node, the root node, which has no parent. The descendant nodes of a node are found by computing the transitive closure of the child relationship. No node may be a descendant of itself.
Unix	The Unix operating system developed by AT&T.
User-defined functions	A file that defines the semantics of some function symbols in the language of the domain.
Well-formed tree	See "Smallest well-formed tree."
Window	An element of a graphical user environment that contains at a minimum a border and a viewing area.
Windows	The Windows operating system developed by Microsoft.

1.2.2. Acronyms

Table 2 lists the acronyms and their associated definitions used in this document.

Table 2: Acronyms used in the SRS

ACRONYM	MEANING
DFD	Data Flow Diagram
GUI	Graphical User Interface
HATS	High Assurance Transformation System
HATS-GUI	HATS graphical user interface
HIS	High Integrity Software
JVM	Java Virtual Machine
ML	Meta language
OS	Operating System

SDT	Syntax Derivation Tree
SNL	Sandia National Laboratories
SRS	Software Requirements Specification
UTEP	The University of Texas at El Paso

1.3. Overview

The SRS is divided into four major sections: Introduction, General Description, Specific Requirements, and Appendices. Section 2, the general description, describes the product, its functionality, and its structure. It contains:

(1) Product Features, which describes from a high-level point of view the main features of the software;
(2) User Characteristics, which identifies the different types of system users, and describes their individual interaction with the system. Use-cases are used to define the user characteristics; a description of the actors, use-cases, and scenarios are included in this section;
(3) General Constraints of the system; and
(4) Assumptions and Dependencies of the system.

Section 3 describes specific requirements. It consists of

(1) External Interface Requirements, which describes the requirements for user, hardware, software, and communications interfaces;
(2) Behavioral Requirements, which divides behavioral requirements into the following categories: related real-world objects, stimulus, related features, and functional requirements;
(3) Non-Behavioral Requirements, which includes performance requirements, qualitative requirements, and design and implementation constraints; and
(4) Other Requirements, which will list all other requirements not included in the previous sections.

Appendix A contains the object model for the HATS-GUI.
Appendix B contains the data flow diagrams for the system.
Appendix C contains state descriptions of the HATS-GUI.
Appendix D describes communications between HATS and the HATS-GUI.
Appendix E and F describe searching outputs strings and trees.
Appendix G gives an example Syntax Derivation Tree (SDT) display.

1.4. References

[1] Creative Software, Omega Software, Porcelain Software, Resilient Software, Suzaku Software, *Interview Report*. February 2001.
[2] Creative Software, Omega Software, Porcelain Software, Resilient Software, Suzaku Software, *Feasibility Report*. February 2001.
[3] Creative Software, Omega Software, Porcelain Software, Resilient Software, Suzaku Software, *Prototype Report*. March 2001.
[4] Creative Software, Omega Software, Porcelain Software, Resilient Software, Suzaku Software, *Draft SRS*. May 2001.
[5] Roach, S., *Communications Protocol Memorandum*, e-mail message dated 2/22/2001, 5:02 p.m.
[6] Roach, S, *Memorandum from Dr. Winter*, e-mail message dated 2/28/2001, 5:53 p.m.
[7] Omega Software Solutions, *Personal Communication Notes on Dr. Winter Q&A*, April 20, 2001.
[8] Winter, V., *An Overview of HATS: A language Independent High Assurance Transformation System*, 1999.
[9] Leyva, F., Outputs from HATS, e-mail message dated 3/2/2001, 5:43 p.m.
[10] Leyva, F., Responses from Dr. Winter, e-mail message dated 3/26/2001, 3:55 p.m.
[11] Roach, S., vlwinter@sandia.gov:MEMO, e-mail message dated 2/28/2001, 5:53 p.m.
[12] Roach, S., Important HATS Information, e-mail message dated 4/2/2001, 7:15 p.m.
[13] Roach, S., Winter Responses, e-mail message dated 4/19/2001, 9:53 a.m.

[14] Winter, Victor L., Meeting with Dr. Winter. University of Texas at El Paso, Computer Science Building, Room 308. April 20, 2001. 3:00 p.m.
[15] Winter, Victor L., S. Roach, Personal communication 5/23/2001.
[16] Winter, Victor L., S. Roach, Personal communication 6/18/2001.
[17] Winter, Victor L., S. Roach, Personal communication 7/20/2001.
[18] Winter, Victor L., S. Roach, Personal communication 8/15/2001.
[19] Winter, Victor L., S. Roach, Personal communication 9/15/2001.
[20] Winter, Victor L., S. Roach, Personal communication 8/28/2001.
[21] Fuehrer, Gary., S. Roach, Personal communication 9/28/2001.
[22] Winter, Victor L., S. Roach, Personal communication 10/1/2001.

2. General Description

HATS was developed to explore transformation-based software development. Transformations are a well-known formal method for developing software. The HATS-GUI is intended to provide HATS users with an intuitive graphical interface to the HATS system. HATS is publicly available, and the HATS-GUI will be delivered with HATS.

The purpose of HATS is to perform program transformations in a provably correct fashion. This enables users to construct software by transforming a *target program*, or simply a *target*, written in an abstract language to a *transformed output program* written in a more concrete language. By demonstrating that the transformations preserve the semantics of the target program, the user has assurance that the transformed program is correct. An overview of the HATS architecture is given in Figure 1 below. HATS applies a sequence of transformations to a target program by following instructions in a *transformation language program*. A transformation language program consists of sequences of transformation instructions and constructs to control the application of transformations.

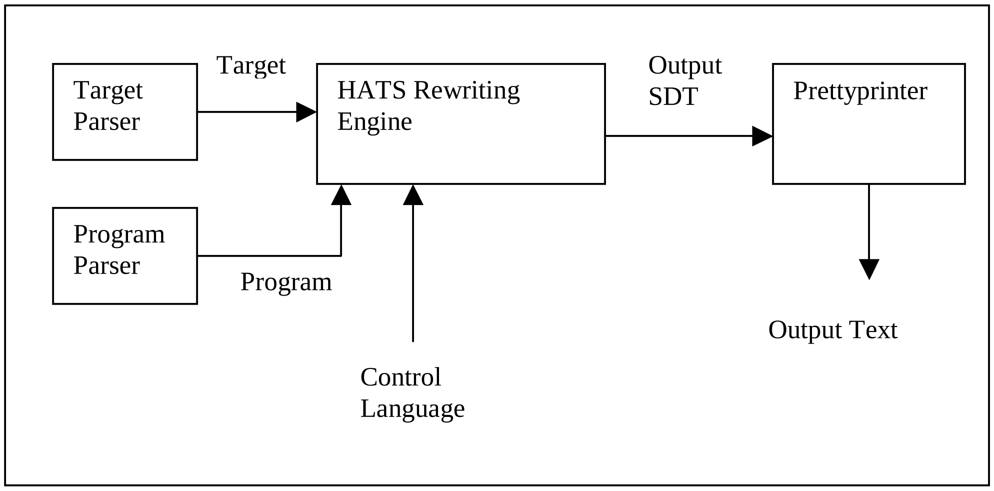


Figure 1 HATS Architecture

HATS provides a set of services that execute transformation language programs. It is written in the language ML, which has been ported to most common computer platforms. HATS services are invoked by executing ML programs. There is one ML program for each function that HATS provides. The collection of programs is referred to in this document at HATS-SML and is described in Appendix D.[1] HATS-SML executes transformation language programs written in a particular problem domain. The *core* of a problem domain describes the language over which transformations may occur. It consists of three elements, each of which is stored in a separate file: a description of tokens in a language, a description of a context-free grammar for a language, and a library of user-defined functions that evaluate some of the function symbols in the language.

An *application* consists of a domain, rules for transforming input strings to output strings, and sets of inputs and outputs. These rules are applied to a target program. The result of applying a sequence of transformations to a target program may be an SDT, a pretty-printed text string, or both. The pretty-printed text string is generated by applying pretty-print rules to an SDT. The pretty-print rules are contained in a pretty-print style file. There may be any number of pretty-print style files, target programs, transformation language programs, or output files in a given application. Additionally, there may be a parsed version of a transformation language program. A summary of the files, file extensions, and names is given in Table 3.

[1] The term *HATS-SML* refers to the set of services provided by the ML programs for applying transformations to targets.

Table 3: Files Composing an Application

File Type	EXTENSION	CONTENT DESCRIPTION
Grammar File	.grm	Definition of the language over which transformations take place. This is part of the core, and exactly one is required for each problem domain.
Lexer Specification File	.spec	Descriptions of tokens in the language over which transformations take place. This is part of the core, and exactly one is required for each problem domain.
User-Defined Functions	.sml	Function definitions written in ML that define semantics for evaluating function symbols in the language. These functions are evaluated semantically, not just syntactically. This is part of the core, and exactly one file is required for each problem domain.
Target file	.tgt	Input program to be transformed. There may be any number of target files in an application.
Parsed target file	.tgt.parsed	SDT corresponding to a parsed target file. There may be zero or one parsed target files for each target file.
Transformation language program	.tlp	Sequence of transformations to be applied to a target file. There may be any number of transformation language program files in an application.
Parsed Transformation language program	.tlp.parsed	SDT corresponding to a parsed transformation language program. There may be zero or one parsed transformation language programs for each transformation language program.
Pretty-print Style file	.sty	Directives that control the formatting of a transformed target file. Any number of pretty-print files may appear in an application.
Pretty-printed output	.txt	Formatted output obtained by applying a transformation language program to a target file.
SDT output	.sdt	Syntax derivation tree obtained by applying a transformation language program to a target file.
Parser		A parser program generated from the grammar and lexical specifications to parse transformation language program files and target files.
Script file	.scr	A file containing a sequence of HATS-SML programs to be executed in order with minimal user intervention.

The HATS-GUI is a system that will provide a user with an intuitive interface to HATS-SML. This system will facilitate the creation, storage, retrieval, and modification of the files in an application. The HATS-GUI will also facilitate the parsing and execution of sequences of transformation language programs, display error messages generated by HATS-SML, and facilitate the display and exploration of transformation outputs in the form of SDTs and pretty-printed text. It is the expectation of the client that by making the interface portable, the number of users of HATS will increase, and that the reliability of HATS-SML will increase due to the increased exercising of its capabilities resulting in error detection and correction.

2.1. Product Features

The HATS-GUI shall provide a user with the ability to construct and use transformations. The HATS-GUI shall facilitate the following activities:

- management of applications;
- parsing of transformation language programs and target programs;
- execution of transformation language programs; and
- examination of SDT and pretty-printed output from transformations.

The subsections that follow provide an overview of each of the mentioned features. The flow of data between these functions is illustrated in the Data Flow Diagrams (DFD) given in Appendix B. The transition of states that the system will be in is illustrated in the State Transition Diagram given in Appendix C.

2.1.1. Management Of Applications

The HATS-GUI shall facilitate the creation, deletion, and modification of the files related to an application. As a practical matter, all files related to an application shall be stored in the same directory or folder. In addition, the HATS-GUI shall manage file associations within an application. File associations are relationships between files. For example, target files will be associated with the most recently used pretty-print style files. The HATS-GUI shall allow the user to examine and modify these associations.

2.1.2. Parsing Of Transformation Language Programs and Target Files

The HATS-GUI shall orchestrate the parsing of transformation language programs and target files. Parsing entails translating a human-readable text file into a syntax derivation tree. Parsing can be a time-consuming operation, particularly for multiple look-ahead grammars. In order to expedite execution of transformations, parsed versions of files can be saved.

Two types of files must be parsed. A transformation language program is a text file describing a set of transformations to be applied to a target file. A target program is a text file containing the input string to be transformed. Parsing is done through the use of a parser program, which HATS-SML generates. Parser generation requires the lexical specification and the grammar as input.

2.1.3. Execution of Transformation Language Programs

The HATS-GUI shall orchestrate the execution of transformation language programs. In order for HATS-SML to execute a transformation program, the program must be parsed and saved as an SDT. The transformation language program specifies which target programs will be transformed. The user creating the transformation language program is responsible for correctly identifying the target program. Input file names are embedded in the transformation language program.

During execution of a transformation language program, HATS-SML creates SDTs as output. These SDTs contain a parsed version of the transformed (output) string suitable for use as input to another transformation or as input to the pretty-print service. (The pretty-print service transforms a target SDT into a text string according to formatting rules contained in a pretty-print style file.) In addition to the output file, the transformation language program may specify that debugging and error outputs. The HATS-GUI shall capture data written to these streams and make that data available to the user.

2.1.4. Examination of SDT and Pretty-Printed Output From Transformations

The HATS-GUI shall facilitate the display of transformation output and error messages received from HATS-SML in response to parsing or transformation. There are several types of output that can be received from HATS-SML. During a given session, data sets may be created, saved as files, used as inputs, or deleted. The HATS-GUI will associate saved files with the application. The HATS-GUI will make a list of these files available to the user and allow the user to select files for display. The display of SDTs and pretty-printed text shall allow the user to manipulate the display by choosing portions of the tree to display and by allowing the user to search for text strings and sub-trees.

2.2. Use Cases

There are three actors in this system: a user, HATS-SML, and the host O. S. The user will access the HATS-GUI primarily to apply transformations to target files. A user may set up an application for another user. Before applying a transformation to a target file, a user must first prepare the application. A user may also choose to view output from a previous transformation. These actors and use cases are discussed below.

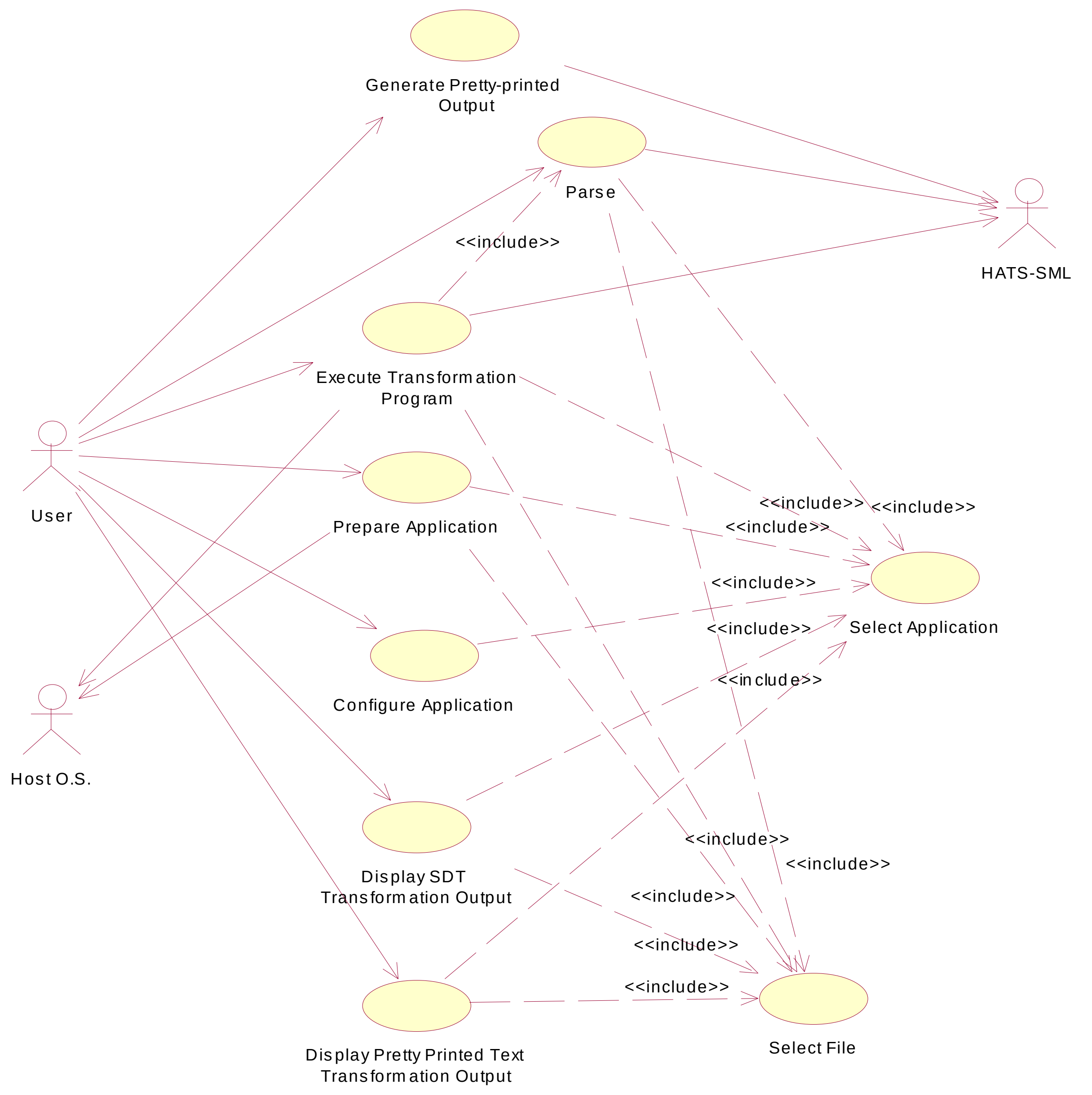


Figure 2 - Use Case Diagram

2.2.1. Actors

<u>User</u>: A user is someone who interacts with the HATS-GUI in order to prepare and execute program transformations. Users of the HATS-GUI will be experts in a given domain and have a high level of understanding of transformations as a tool for constructing software. These users are expected to be computer savvy, possessing experience with the operation of editors and software tools, hierarchical file systems, and menu navigation. A limited amount of direction and assistance will be necessary.

<u>HATS-SML</u>: HATS-SML provides the following services: make a parser file, parse a target file, parse a transformation language program, apply a transformation language program to a target file, and pretty-print an output target SDT. HATS-SML may generate parsed transformation language programs, SDTs, pretty-printed text, or error messages as output. The protocol for the use of HATS-SML is detailed in Appendix D.

Host OS: This is the operating system under which the HATS-GUI and HATS-SML run. The host OS is expected to provide file services and provide a buffer for copy-and-paste operations between applications.

2.2.2. Use-case Descriptions

The following use cases describe possible interactions between the HATS-GUI, the user, HATS-SML, and the host operating system. They do not describe every possible interaction. In the descriptions that follow, steps in a scenario are numbered sequentially. Alternatives to a particular step are suggested by the flag "ALT <n>" where <n> is an alternative listed below the scenario. Steps in the alternative replace steps in the main scenario. The step numbers in the alternative indicate what steps in the main scenario are being replaced. For example, the step number "A1-2A" indicates this step is in the alternative numbered 1 (A1) and this step is the first step replacing step 2 in the main scenario. If more than one alternative is listed, either alternative may be taken.

2.2.2.1. Use-case 1: Parse

Use Case Summary: The user requests the parsing of a transformation language program or a target file. The file to be parsed and the parser file are inputs to HATS-SML. HATS-SML generates an SDT. The output is saved in the application.
Actors: User, HATS-SML, Host OS
Used by: Execute Transformation
Preconditions:

1. An application has been selected (Refer to Use Case 3).
2. The transformation language program and core domain files have been created (Refer to Use Case 5).

Scenario 1:

1. User selects the option to parse a transformation language program from currently open application (ALT 1).
2. User selects a transformation language program file (.tlp) to parse (Refer to Use Case 4) (ALT 2).
3. The HATS-GUI compares the time stamps on the lexical specification and grammar files to the time stamp on the application's parser file. The parser file is the youngest of the three files (ALT 3, ALT 6).
4. The HATS-GUI sends a message to the Host OS to start either the ParseTarget or the ParseProgram program and sends the names of the target file or program file, the output file, and the parser file to the Host OS.
5. HATS-SML parses the input file (ALT 7) and writes the output to the output file.
6. End of use case.

ALT 1: The user selects the option to parse a target file instead of a transformation language program.
A1-2A: The user selects a target file to parse (Refer to Use Case 4) (ALT 2).
A1-2B: Use case continues with step 3.

ALT 2: A file is already selected.
A2-2A: Step 2 is skipped.
A2-2B: Use case continues with step 3.

ALT 3: No parser file exists.
A3-3A: The HATS-GUI sends a message to the Host OS to start the MakeParser program and sends the lexical specification and grammar file names and the parser file name.
A3-3B The Host OS starts the MakeParser program.
A3-3C: MakeParser writes the output parser file (ALT 5).
A3-3D: Use case continues with step 4.

ALT 5: Error on creation of parser file.

A5-3A: HATS-SML encounters an error creating the parser file. The HATS-GUI writes the error message to the error window.
A5-3B: End of use case.

ALT 6: Parser file is not youngest.
A6-3A: Use case continues with step A3-3A.

ALT 7: Parser error
A7-5A: HATS-SML encounters an error parsing the input file. The HATS-GUI writes the error message to the error window.
A7-5B: End of use case.

2.2.2.2. Use-case 2: Execute Transformation Program

Use Case Summary: The user requests the application of a transformation language program to a target file. HATS-SML applies the transformation language program to the target.
Actors: User, HATS-SML, Host OS
Preconditions:

1. Target and transformation language programs have been prepared.
2. An application has been selected (Refer to Use Case 3).

Scenario 1:

1. User selects the option to execute a transformation language program from the currently selected application.
2. User selects a transformation language program file (Refer to Use Case 4) (ALT 1).
3. The HATS-GUI compares the time stamp on the lexical specification and grammar files to the time stamp on the parser file. The parser file is younger (ALT 2, ALT 3).
4. The HATS-GUI compares the time stamp of the parsed program file and the parser, the transformation language program, and the user-defined library files. The parsed program file is younger than any of these files (ALT 4, ALT 5).
5. The HATS-GUI clears the displays for the standard error and standard output.
6. The HATS-GUI sends a message to the Host OS to start the *ApplyTransformations* program and sends the names of the parsed program file and the user-defined library file to the Host OS. The HATS-GUI redirects the standard error and standard output streams of the *ApplyTransformation* process and collects data written to these streams for later display.
7. The HATS-SML *ApplyTransformation* program writes output SDTs to files named in the transformation language program. (The transformation language program contains all references to input and output files. The HATS-GUI does not handle file input and file output for *ApplyTransformaion*.)
8. End of use case.

ALT 1: A transformation language program file has already been selected.
A1-2A: Step 2 is skipped.
A1-2B: Use case continues with step 3.

ALT 2: No parser file exists.
A2-3A: The HATS-GUI creates a program parser file (Refer to Use Case 1).
A2-3B: Use case continues with step 4.

ALT 3: The parser file is not younger.
A3-3A: Use case continues with step A2-3A.

ALT 4: No program.parsed file exists.
A4-4A: The HATS-GUI creates a parsed program file (Refer to Use Case 1).
A4-4B: Use case continues with step 5.

ALT 5: Program.parsed file is not younger.

A5-4A: Use case continues with step A4-4A.

2.2.2.3. Use-case 3: Select Application

Use Case Summary: The user selects an application.
Actors: User, Host OS
Used by:

1. Parse Transformation Language Program (Refer to Use Case 1).
2. Run Transformation (Refer to Use Case 2).
3. Prepare Application (Refer to Use Case 4).
4. Display SDT Transformation Output (Use Case 5).
5. Display Pretty-printed text Transformation Output (Refer to Use Case 6).

Preconditions: Desired application exists (Refer to Use Case 5).
Scenario 1:

1. System requests from Host OS a list of existing applications within current directory.
2. Host OS returns list of files within current directory.
3. System displays list of existing applications within current directory (ALT 1, ALT 2).
4. User selects an application within the list (ALT 2).
5. System sets selected application as the current application.
6. End of use case.

ALT 1: No applications exist in current directory.
A1-3A: User cannot select any application to perform operations on it.
A1-3B: End of use case.

ALT 2: User chooses to move up or down a directory level.
A2-3A: User selects a subdirectory or the parent directory.
A2-3B: System changes current directory.
A2-3C: Use case continues with step 1.

2.2.2.4. Use-case 4: Select File

Use Case Summary: The user wants to perform some operation, and the system requests a filename. A list of file names in the currently selected application is presented. The user selects one file name in the application.
Actors: User, Host OS
Used by:

1. Parse Transformation Language Program (Refer to Use Case 1).
2. Run Transformation (Refer to Use Case 2).
3. Prepare Application (Refer to Use Case 4).
4. Display SDT Transformation Output (Refer to Use Case 5).
5. Display Pretty-printed text Transformation Output (Refer to Use Case 6).

Preconditions: An application has been selected (Refer to Use case 3).
Scenario 1:

1. System requests from Host OS a list of files in the current directory.
2. Host OS only returns list of files in the current directory.
3. System displays this list of files (ALT 1, ALT 2).
4. User selects a file within the list (ALT 2, ALT 3).
5. System sets selected file as the current file.
6. End of use case.

ALT 1: No files exist in current directory.
A1-3A: User cannot select any application to perform operations on it.
A1-3B: End of use case.

ALT 2: User chooses to move up or down a directory level.
A2-3A: User selects a subdirectory or the parent directory.
A2-3B: System changes current directory.

A2-3C: Use case continues with step 1.

ALT 3: User enters path of file instead of selecting from list.
A3-4A: User types path and file name.
A3-4B: System/OS accesses file using path (ALT 4).
A3-4C: System prompts user to accept copying file into current application. User has option to confirm or cancel.
A3-4D: User confirms (ALT 5).
A3-4E: System copies file into current application.
A3-4F: Use case continues with step 5.

ALT 4: User-entered file not available.
A4-4A: System presents user with error “file not found.”
A4-4B: End of use case.

ALT 5: User does not confirm.
A5-A3-4A: System does not copy file into current application.
A5-A3-4B: Use case continues with A3-4F.

2.2.2.5. Use-case 5: Prepare Application

Use Case Summary: The user wants to perform some operation on a selected application. These operations include open, close, save, save as, create new, edit files within application, and configure the application.
Actors: User, Host OS
Scenario 1: Close Application
Preconditions: Application has been selected (Refer to Use Case 3).

1. User selects option to close the currently selected application
2. System asks for confirmation from user to close application.
3. User confirms to close application (ALT 1).
4. System checks that the changes to currently selected application’s configuration have been saved (ALT 2).
5. System sets currently selected application to ‘none’.
6. End of use case.

ALT 1: User cancels operation
A1-3: User selects cancel option.
A1-4: System cancels the operation.
A1-5: End of use case.

ALT 2: Changes to currently selected application have not been saved.
A2-4A: System asks user to confirm save application.
A2-4B: User chooses to save application (ALT 3).
A2-4C: System saves the changes made to the application configuration before closing (Scenario 2).
A2-4D: End of use case.

ALT 3: User chooses not to save application.
A3-4: System closes the application without saving the changes made to the application.
A3-5: End of use case.

Scenario 2: Save Application
Preconditions: Application has been selected (Refer to Use Case 3).

1. User selects option to save currently selected application.
2. System requests Host OS to save configuration file for the selected application.
3. Host OS saves the configuration file for the selected application into a non-volatile storage (ALT 1).
4. System informs user that application has been saved.
5. End of use case.

ALT 1: Host OS operating system was unable to save the current application.
A1-3A: System informs that application has not been saved.
A1-3B: End of use case.

Scenario 3: Save Application As
Preconditions: Application has been selected (Refer to Use Case 3).

1. User selects to save a currently open application using a different directory name.
2. System requests user for a name to save the open application.
3. User enters or selects a name for the application to be saved (ALT 1).
4. System confirms that application name is unique (ALT 2).
5. System requests Host OS to create new directory with the given name.
6. Host OS creates new application with the entered name (ALT 3).
7. System requests Host OS to copy core files, configuration files, and .tlp files to the new directory.
8. Host OS copies all files from selected application to created application (ALT 3).
9. System informs user that application has been saved with entered name.
10. End of use case.

ALT 1: User chooses to cancel operation
A1-3: User selects cancel option.
A1-4A: System cancels the operation and the application is not saved.
A1-4B: End of use case.

ALT 2: Application already exists.
A2-4A: System informs user that an application already exists with this name.
A2-4B: End of use case.

ALT 3: Host OS experiences file system error.
A3: End of use case.

Scenario 4: Edit Files in an Application
Preconditions: Application has been selected (Refer to Use Case 3).

1. User selects option to edit a file within the currently selected application.
2. User selects file to edit (Use Case 4).
3. System checks application to determine if editor is associated with this file. An editor is associated with the file if the configuration explicitly associates this file with an editor or an editor is associated with files of this type. An editor is associated with this file (ALT 1).
4. The HATS-GUI sends message to Host OS to start the preferred editor in new process and passes file name to editor.
5. Host OS initiates process, puts editor in process, sends the filename as argument.
6. End of use case.

ALT 1: No editor is associated with this file.
A1-3A: The HATS-GUI prompts the user to select a configured editor.
A1-3B: The user selects a configured editor.
A1-3C: The HATS-GUI prompts the user to associate the editor with the file or the file type.
A1-3D: The user selects associating the editor with the file type (ALT 2, ALT 3).
A1-3E: The HATS-GUI associates the editor with the file type by updating the application configuration.
A1-3F: The selected editor becomes the preferred editor.
A1-3G: Use case continues with step 4.

ALT 2: User selects associating editor with specific file.
A2-3E: The HATS-GUI associates the editor with the specific file by updating the application configuration.
A2-3F: Use case continues with step A1-3F.

ALT 3: User selects cancel.
A3-3: End of use case

Scenario 5: Create New Application
Preconditions: None.

1. User selects option to create a new application.
2. Close existing application if one is selected (Refer to Use Case 5 – Scenario 1).
3. System asks user to enter a name and location for new application.
4. User enters name and location for new application (ALT 1).
5. System requests Host OS to create new application.
6. Host OS creates new application (ALT 2).
7. System informs user that new application has been created.
8. End of use case.

ALT 1: User selects to cancel operation.
A1-4A: System cancels operation. No application is created.
A1-4B: End of use case.

ALT 2: An application with the same name already exists.
A2-6A: System presents error message "Application already exists."
A2-6B: End of use case.

2.2.2.6. Use-case 6: Configure Application

Use Case Summary: The user selects an application. The user configures the settings of the selected application. The settings of an application include, available editors, SDT display colors, SDT display shapes, default number of levels to expand, file associations between files and editors, and file associations between target files and transformation language program.
Actors: User, Host OS
Preconditions: Application has been selected (Refer to Use Case 3).
Scenario 1: Configure Editor List.

1. User selects option to configure editors.
2. The system prompts for editor name and configuration information. Editor configuration information must be sufficient to start the editor on the Host OS.
3. User enters editor name and parameters (ALT 1).
4. System saves editor configuration in application configuration.
5. End of use case.

ALT 1: User selects to cancel.
A1-3: End of use case.

Scenario 2: Configure SDT display.

1. User selects option to configure the colors and shapes of the graphical representation of an SDT.
2. System displays a list containing types of displayed nodes and their corresponding current configured colors. The types of displayed nodes are given in Table 4.
3. User selects a type of node from list. (ALT 1).
4. System displays a list of supported colors and shapes.
5. User selects a color and a shape from the list for the selected type of displayed node (ALT 2).
6. System sets the selected values as the color and shape for the type of displayed node selected.
7. End of use case.

ALT 1: User cancels the 'configure display' operation.
A1-3A: System cancels operation. No changes in the default colors of are made.
A1-3B: End of use case.

ALT 2: User cancels operation for a particular type of displayed node selected.

A2-5A: System cancels operation. No change to the default color for a particular type of displayed node is made.
A2-5B: Use case continues with step 2.

Table 4: Types of displayed nodes

Node Type	**Description**
Root	Displayed node corresponding to the root node of the SDT.
Non expandable node	Displayed node corresponding to a terminal node in an SDT. The terminal node has no children in the SDT, and the displayed node cannot be expanded.
Non expandable node, hidden parent	A non-expandable displayed node D that corresponds to an SDT node N where N has a parent node P, but there is no display node corresponding to P. (Although not technically correct, one can consider this the case where the parent of D is not displayed.)
Expandable node	Displayed node drawn as a leaf node (no displayed descendants) but whose corresponding SDT node is non-terminal.
Expandable node, hidden parent	An expandable displayed node D that corresponds to an SDT node N where N has a parent node P, but there is no display node corresponding to P. (Although not technically correct, one can consider this the case where the parent of D is not displayed.)
Internal nodes	A displayed node drawn with a parent and descendants.

Scenario 3: Configure Expand Levels.

1. User selects option to configure the default number of levels to expand in response to a single mouse click.
2. System asks user to select or enter a number of levels to expand.
3. User enters or selects a number of levels to expand.   Number entered is a natural number.
4. System sets the entered or selected number of levels as the default number of levels to expand.
5. End of use case.

Scenario 4: File Associations: Editors.

1. User selects option to associate file types with editors (ALT 1).
2. System displays a list of configured editors. (Refer to Use Case 6.)
3. System displays a list of file types.
4. User selects a file type from list (ALT 2).
5. User selects an editor from list (ALT 2).
6. System associates the selected file type with the selected editor. When the user tries to edit a file of the type selected, the system will request the Host OS to initiate a process with the associated editor. (Refer to Use Case 5.)
7. End of use case.

ALT 1: User selects option to associate a particular file with an editor.
A1-1A: System displays a list of configured editors. (Refer to Use Case 6.)
A1-1B: System displays the list of files in the currently selected application (ALT 3).
A1-1C: User selects a file from list (ALT 2).
A1-1D: User selects an editor from list (ALT 2).
A1-1E: System associates the selected file with the selected editor. When the user tries to edit the selected file, the system will request the Host OS to initiate a process with the associated editor. (Refer to Use Case 5.)
A1-1F: End of use case.

ALT 2: User cancels operation.
A2-1A: User selects cancel option.
A2-1B: System cancels operation. No changes in the associations are made.

A2-1C: End of use case.

ALT 3: File is already selected.
A3-1: Use case continues with step A1-1D.

Scenario 5: Save Configuration

1. User selects option to save configuration.
2. System saves configuration with the application.
3. End of use case.

2.2.2.7. Use-case 7: Display SDT Transformation Output

Use Case Summary: The user wants to display and manipulate a graphical representation of an SDT. Manipulation of a graphical representation of an SDT includes selecting and unselecting displayed nodes, navigation through the display using a navigation window and scrolling, expanding and collapsing displayed nodes, hiding displayed nodes, searching for SDT sub-trees, and finding smallest well-formed trees.
Actors: User, Host OS
Preconditions: Transformation of a target file was successful and transformation output has been stored and associated with the application.

Scenario 1: SDT Display.

1. User selects option to view and manipulate a graphical representation of an SDT.
2. User selects SDT to view and manipulate (Refer to Use Case 4).
3. System displays a graphical representation of the selected SDT in an on-screen work area.
   The work area of the SDT display contains the following elements:
   a) A window that displays the displayed graph representation of the SDT.
   b) A text window that displays the labels of the leaf nodes in the currently displayed graph.
   c) If either window is too small to contain all the information to be presented, the windows will have scroll bars to facilitate scrolling. The system will respond to dragging of scroll buttons (or pressing arrow keys) by shifting the display in the appropriate direction.
   d) If the SDT is too large for the window, a separate, small window with a compressed view of the SDT will be displayed. In this window, a displayed graph of the entire SDT will be drawn. The nodes of this displayed graph will be smaller than the displayed nodes in the SDT display work area. It is not necessary that the user be able to read node labels. It is only necessary that the user be able to see the high-level structure of the SDT. A small box named the *locator box* will be displayed showing the part of the SDT currently displayed in the main SDT display window.
4. User is able to manipulate the displayed graph representation of the SDT. (See the remaining scenarios.)

Scenario 2: Select nodes.

1. User selects one or more displayed nodes. Displayed node selection methods are described here.
   - A single node can be selected by a right mouse click on the node.
   - A single node can be selected via the keyboard by moving the keyboard cursor to a node and pressing the *enter* key. The keyboard cursor is moved between nodes using the *tab* key.
   - Several nodes can be selected by drawing a mouse box around the desired nodes.
   - Several nodes can be selected by holding the *shift* key down and selecting individual displayed nodes.

   When a user selects a set of nodes using these operations (and the *shift* key is not held down), the previously selected set of nodes (if any) is unselected.
2. System highlights the smallest well-formed tree of the selected nodes in the displayed graph.
3. System finds corresponding portion of leaf text of the selected nodes in the displayed graph.
4. System highlights the corresponding portion of leaf text.
5. If both SDT and Pretty-printed text are displayed, the system will find the corresponding pretty-printed text and highlight that text in the Pretty-print text window.
6. End of use case.

Scenario 3: Navigation window navigation.

1. The user clicks the mouse in the navigation window SDT display.
2. The system determines the node in the displayed graph closest to the mouse cursor and redraws the displayed graph with this node in the center of the main display.
3. The system updates the locator box in the navigation window display.
4. End of use case.

Scenario 4: Expand

1. User points the mouse cursor at a node and clicks the left mouse button (ALT 1).
2. System displays the tree by expanding descendants of the node in the displayed graph. The number of levels to expand is given by the application configuration. If fewer than this number of levels exist, then all descendants of the chosen node are displayed
3. System changes the shape and color of selected node as specified in the application configuration to indicate that it has become an expanded node.
4. End of use case.

ALT 1: User selects to expand selected nodes.
A1-1A: User selects a set of nodes (Refer to Scenario 2).
A1-1B: User selects to expand selected nodes.
A1-1C: Use case continues with step 2.

Scenario 5: Collapse

1. User points the mouse cursor at a node and clicks the left mouse button.
2. System removes all descendants of the selected node from the displayed graph.
3. System changes the shape and color of selected node as specified in the application configuration to indicate that it has become a collapsed node.
4. End of use case.

Scenario 6: Hide nodes

1. User selects one or more displayed nodes. (Refer to Use Case 7.)
2. User selects option to hide selected displayed nodes.
3. System removes the selected nodes from the displayed graph.
4. System changes the types of the displayed nodes remaining on the screen to indicate that there are hidden nodes in the displayed graph.
5. End of use case.

Scenario 7: Search

1. User selects option to search SDT for a desired sub-tree.
2. System asks user for the criteria to search nodes. Search criteria for SDTs are presented in Appendix E.
3. User enters search criteria.
4. System finds the first matching sub-tree in the SDT starting at the current keyboard cursor location (ALT 1).
5. System creates a displayed graph for the sub-tree matching the search criteria. The displayed nodes of the matching sub-tree are highlighted. If no nodes match the search criteria, none are highlighted.
6. If both SDT and Pretty-printed text are displayed, the pretty-printed text corresponding to the highlighted nodes of the SDT is highlighted.
7. End of use case.

ALT1: Failed search.
A1-4: The system displays a message indicating that the search has failed.
A1-5: End of use case.

Scenario 8: Repeat Search

1. User selects option to repeat a search SDT for a desired sub-tree.
2. System finds the next matching sub-tree in the SDT starting at the current keyboard cursor location.

3. System creates a displayed graph for the sub-tree matching the search criteria. The displayed nodes of the matching sub-tree are highlighted. If no nodes match the search criteria, none are highlighted.
4. If both SDT and Pretty-printed text are displayed, the pretty-printed text corresponding to the highlighted nodes of the SDT is highlighted.
5. End of use case.

Scenario 9: Copy Text From Text Window.

1. System displays text of leaf nodes in currently displayed graph in a text window.
2. User selects a portion of displayed text.
3. System highlights selected portion of text in the text window.
4. User selects option to copy selected text. Minimally, the system will copy selected text in response to the *Ctrl-Insert* key combination.
5. System puts the selected and copied text in the Host OS copy/paste buffer.
6. End of use case.

Scenario 10: Refresh Display.
Precondition: The system is displaying an SDT generated by a transformation program. The transformation program is executed while the SDT is being displayed. The SDT file is overwritten by the *ApplyTransformation* program.

1. The user selects the option to refresh the display of the SDT.
2. The system disposes of the currently displayed SDT.
3. The system reads the SDT file and displays the newly generated SDT. The display of the SDT contains approximately the same number of nodes as the previously displayed SDT.
4. End of use case.

2.2.2.8. Use-case 8: Display Pretty-Printed Text Transformation Output

Use Case Summary: The user wants to display and manipulate a text representation of the result of applying a transformation language program to a target file. Manipulation of text includes selecting text and searching for sub-strings.
Actors: User, Host OS
Preconditions: Transformation of a target file was successful and transformation output has been stored and associated with the application.

Scenario 1: Display Pretty-Printed Text

1. User selects option to view and manipulate a pretty-printed text from a particular successful transformation.
2. User selects pretty-printed text to view and manipulate. (Refer to Use Case 4.)
3. System displays the pretty-printed text in a work window.
    a) The work area of the display of pretty-printed text contains a text window that displays the text.
    b) If the window is too small to contain all the information to be presented, the windows will have scroll bars to facilitate scrolling. The system will respond to dragging of scroll buttons (or pressing arrow keys) by shifting the display in the appropriate direction.
4. User is able to manipulate the pretty-printed text. (See the remainder of the scenarios.)
5. End of use case.

Scenario 2: Select text.

1. User selects a portion of the displayed pretty-printed text. Text selection methods are described here.
    - Text can be selected by clicking and dragging the mouse across an area of text.
    - Text can be selected by holding the shift key and moving the keyboard cursor using the arrow keys.

    When a user selects text using these operations, the previously selected text (if any) is unselected.
2. System highlights selected portion of the displayed pretty-printed text.

3. If both pretty-printed text and graphical representation of SDT are displayed, the system finds the corresponding displayed nodes of the SDT of the selected portion of pretty-printed text and highlights the corresponding displayed nodes of the SDT.
4. End of use case.

Scenario 3: Search
1. User selects option to search pretty-printed text for desired text.
2. System asks user for the criteria to search text. Text search criteria are given in Appendix F.
3. User enters or selects criteria to search text.
4. System finds those portions of pretty-printed text that match the search criteria and highlights those portions of the text that match (ALT 1).
5. If both the pretty-printed text and the graphical representation of an SDT are displayed, the system finds the corresponding nodes of the SDT of the selected portion of pretty-printed text and highlights the corresponding nodes of the SDT.
6. End of use case.

ALT1: Failed search.
A1-4: The system displays a message indicating that the search has failed.
A1-5: End of use case.

2.2.2.9. Use-case 9: Generate Pretty-printed Output

Use Case Summary: The user requests the application of a pretty-print style file to a target SDT. The necessary file names are passed to HATS-SML, HATS-SML applies the pretty-print style to the SDT, and a formatted text file is generated.
Actors: User, HATS-SML
Preconditions:
1. A target SDT file has been generated and a pretty-print style file exists.
2. An application has been selected. (Refer to Use Case 3.)

Scenario 1:
1. User selects the option to pretty-print a target from the currently selected application.
2. User selects a target SDT (Refer to Use Case 4) (ALT 1).
3. No pretty-print style file is associated with the target file. The HATS-GUI prompts the user to select a pretty-print style file.
4. User selects a pretty-print style file (Refer to Use Case 4) (ALT 2).
5. HATS-GUI generates a file name by appending ".txt" to the target SDT root name.
6. The HATS-GUI verifies that no file with this name exists in the current directory. This file becomes the output file name (ALT 3).
7. The HATS-GUI sends a message to the Host OS to start the Pretty-print program and sends the target SDT file name, the pretty-print style file name, and the output file name to Pretty-print as command line arguments.
8. HATS-SML applies the style file to the target SDT and writes the output to the output file specified on the command line (ALT 6).
9. HATS-SML terminates.
10. End of use case.

ALT 1: A target SDT has already been selected.
A1-2A: Step 2 is skipped.
A1-2B: Use case continues with step 3.

ALT 2: A pretty-print style file is associated with the target file.
A2-1: Use case continues with step 5.

ALT 3: A file already exists with the output file name.
A3-6A: HATS-GUI prompts the user to save the output under a new name. The prompt includes a text window containing the current output file name, a cancel option, and an OK option.

A3-6B: The user enters a new name and selects OK (ALT 4, ALT 5).
A3-6C: Use case continues with step 6.

ALT 4: User selects the Cancel option.
A4-1: End of use case.

ALT 5: User selects OK option without changing file name.
A5-6A: The output file name remains unchanged. The existing file will be overwritten.
A5-6B: Use case continues with step 7.

ALT 6: The Pretty-print encounters an error.
A6-8A: No text file is generated. The HATS-GUI displays an error message.
A6-8B: End of use case.

2.3. General Constraints

The general constraints on the development of the system are as follows:

- The system shall be platform independent and function properly on any operating system. Specifically, the system shall be tested under Sun Solaris, Windows 2000, and Linux. It is intended for the system to run under any variation of the Microsoft Windows operating system later than Windows 95.
- The system shall be designed in such a way as to minimize the number of windows opened by the application. For example, it is not acceptable to open a new window for every output file generated by HATS-SML at the time each file is received by the HATS-GUI.
- The system shall be developed using the Java programming language.
- The system shall be completed by December 2001.

2.4. Assumptions and Dependencies

The following assumptions are made with respect to the HATS-GUI:

- A JVM shall have been installed on the system under which the HATS-GUI is running.
- An ML interpreter shall have been installed on the system under which the HATS-GUI is running.
- HATS shall run on the same system on which the HATS-GUI is running.
- HATS shall support only one user at a time.
- Users shall have a high level of sophistication and shall not need guidance in the form of extensive help messages.
- The Host OS shall support a hierarchical, tree-structured file system.
- The development team shall use this SRS to implement the system.
- HATS-SML version 2.0 shall be available by October, 2001.

3. Specific Requirements

3.1. External Interface Requirements

This section contains the specification of requirements for user, hardware, software and communication interfaces among different components and their external capabilities.

3.1.1. User Interfaces

[SRSreq 01] Any operation requiring the user to supply a file name shall allow the user either to select the files from a list derived from files in the currently open application or to key in a file name, a relative path name, or a complete path name.

[SRSreq 02] For any operation where the user is prompted to select from a list, the user shall be able to cancel the operation.

[SRSreq 03] When collecting generated output files from HATS-SML, the HATS-GUI shall overwrite any file whose name is generated automatically. For example, given a transformation language program *X.tlp*, the file *X.tlp.parsed* is generated when the parser is executed. If *X.tlp.parsed* existed prior to executing the parser, it shall be overwritten without prompting the user.

[SRSreq 04] When saving or copying files, the HATS-GUI shall use the following sequence of actions:

- The HATS-GUI shall prompt the user for a file name for the file to be saved.
- The HATS-GUI shall examine the contents of the application directory.
- If a file with the given name already exists, the HATS-GUI shall prompt the user to overwrite the existing file. If the user agrees to overwrite the file, the HATS-GUI shall:
    - attempt to write the new file using a temporary name,
    - delete the previously existing file,
    - then rename the new file.
- If the user does not agree to overwrite the existing file or if system errors occur while writing or renaming the new file or deleting the previous file, the new file shall not be written and the previous file shall be left unchanged.
- In the case of an error, the HATS-GUI shall notify the user of the error.

For example, the user shall be able to save an SDT generated during transformation program execution. If the user specifies the name of an existing file, the existing file is only overwritten after the user confirms that it is permissible to do so.

[SRSreq 05] Output pretty-printed text files and SDTs shall only be displayed when the user requests.

[SRSreq 06] The main interface shall be menu driven. A summary of the menus available is given below in Table 5. Items in the first row of Table 5 are top-level menu items. Items in following rows are second-level menu items.

Table 5: Menu Items for HATS-GUI

File	Application	Configure	Run	View
Select Edit Save-as Delete Exit	Select Close Save Save-as Delete	Editors Node Display File Associations HATS-SML	Generate Parser Parse Target Parse Program Execute Transforms Pretty-Print	SDT Text Standard error Standard output Select Text Copy Text

[SRSreq 07] The *File* menu shall contain menu items related to manipulating individual files and for closing the HATS-GUI application.

[SRSreq 08] The *Application* menu shall contain items related to manipulating entire applications.

[SRSreq 09] The *Configure* menu shall contain items related to modifying the configuration of the application.

[SRSreq 10] The *Run* menu shall contain items related to executing the HATS-SML programs.

[SRSreq 11] The *View* menu shall contain items related to viewing HATS-SML output.

[SRSreq 12] The *File/Select* menu item shall provide an interface for selecting files within the currently open application.

[SRSreq 13] The *File/Edit* menu item shall provide an interface for starting an editor as an external process. The editor shall be started according to information in the application configuration.

[SRSreq 14] The *File/Save-as* menu item shall provide an interface for saving a previously selected file under a different name.

[SRSreq 15] The *File/Delete* menu item shall provide an interface for deleting a selected file. The HATS-GUI shall prompt the user to confirm deletion prior to completing this action.

[SRSreq 16] The *File/Exit* menu item shall provide an interface for exiting the HATS-GUI.

[SRSreq 17] The *Application/Select* menu item shall provide an interface for selecting and opening an application.

[SRSreq 18] The *Application/Close* menu item shall provide an interface for closing an application.

[SRSreq 19] The *Application/Save* menu item shall provide an interface for saving application configuration information.

[SRSreq 20] The *Application/Save-as* menu item shall provide an interface for saving an application and its associated files under a different name.

[SRSreq 21] The *Application/Delete* menu item shall provide an interface for deleting an application and all the files in the application directory. The HATS-GUI shall prompt the user to confirm deletion prior to completing this action.

[SRSreq 22] The *Configure/Editors* menu item shall provide an interface for the user to enter editor configuration information. This information shall be used when editors are started as processes.

[SRSreq 23] The *Configure/Node* Display menu item shall provide an interface for configuring the display of nodes.

[SRSreq 24] The HATS-GUI shall allow the user to specify the number of levels of nodes to expand during SDT viewing.

[SRSreq 25] The HATS-GUI shall allow the user to select the colors for types of displayed nodes in a displayed graph. At a minimum, the system shall provide eight colors (red, yellow, white, black, green, blue, orange, and violet) for displayed nodes.

[SRSreq 26] The HATS-GUI shall allow the user to select the shapes for types of displayed nodes in a displayed graph. At a minimum, the system shall provide two shapes, rectangles and ellipses, for displayed nodes.

[SRSreq 27] The *Configure/File Associations* menu item shall provide an interface for associating files in an application. File associations are shown in Table 6.

[SRSreq 28] The *Configure/HATS-SML* menu item shall provide an interface for the user to enter implementation-dependent information for starting HATS-SML processes.

[SRSreq 29] The *Run/Generate Parser* menu item shall provide an interface for generating a parser program.

[SRSreq 30] The *Run/Parse Target* menu item shall provide an interface for selecting and parsing target programs.

[SRSreq 31] The *Run/Parse Program* menu item shall provide an interface for selecting and parsing transformation language programs.

[SRSreq 32] The *Run/Execute Transforms* menu item shall provide an interface for selecting and executing transformation language programs.

[SRSreq 33] The *Run/Pretty-print* menu item shall provide an interface for selecting SDTs and pretty-print style files and formatting the SDT according to the style.

[SRSreq 34] The *View/SDT* menu item shall provide an interface for viewing, navigating through, and searching SDTs output from the execution of transformation programs.

[SRSreq 35] The *View/Text* menu item shall provide an interface for viewing and searching pretty-printed text.

[SRSreq 36] The *View/Standard Error* menu item shall provide an interface for viewing error strings collected from HATS-SML.

[SRSreq 37] The *View/Standard Output* menu item shall provide an interface for viewing debugging information collected from HATS-SML.

[SRSreq 38] The *View/Select Text* menu item shall provide an interface for selecting pretty-printed text or text strings associated with SDT displays.

[SRSreq 39] The *View/Copy Text* menu item shall provide an interface for copying selected text into an operating system buffer.

3.1.2. Hardware Interfaces

There are no local hardware interface requirements.

3.1.3. Software Interfaces

3.1.3.1. HATS-SML

[SRSreq 40] The HATS-GUI shall interface with HATS-SML (version 2.0) in order to parse transformation language programs, execute transformation language programs, and create output SDTs, error messages, pretty-printed text, and parsed transformation language programs. The interface shall consist of starting HATS-SML programs and passing command line arguments and capturing data written to the standard output and standard error streams of the HATS-SML processes. The HATS-SML programs and their command line arguments are described in Appendix D.

[SRSreq 41] The GUI shall allow the user to create, modify, copy, or delete applications or files regardless of the current status of HATS-SML.

[SRSreq 42] If the HATS-GUI attempts to start a HATS-SML program and fails, the HATS-GUI shall notify the user by displaying the following message: "Unable to start HATS-SML." The HATS-GUI shall provide any other details related to the failure that are available from the Host OS. Examples of information available from an operating system include indications that a specified file does ot exist, that a file cannot be opened due to file protection violations, or that errors occurred during I/O operations.

3.1.3.2. Host OS

[SRSreq 43] The HATS-GUI shall be able to communicate with Sun Solaris, Linux, and Windows 2000.

[SRSreq 44] Data sets shall be saved as files using file names with the appropriate extensions. See Table 3 for description of the files and extensions.

[SRSreq 45] The user shall be able to initiate processes from an interface provided by the HATS-GUI. The HATS-GUI shall optionally pass command line arguments to the process. Command sequences for initiating processes are part of the application configuration.

[SRSreq 46] After initiating processes for editing files, the HATS-GUI is not responsible for communications with the process other than passing command line arguments.

[SRSreq 47] The HATS-GUI shall interact with the HOST OS file services to perform the following tasks:

- Create new files
- Delete existing files
- Open an existing file for read and/or write
- Read from an open file
- Write to an open file
- Close an open file.

[SRSreq 48] The HATS-GUI shall interact with the HOST OS to find files in directories. (This capability is necessary for the HATS-GUI to display file lists.)

[SRSreq 49] The HATS-GUI shall interact with the HOST OS to compare time stamps for files. Time stamps indicate the date and time of the most recent update to a file. These time stamps must be reported with a precision of one second or less.

3.2. Behavioral Requirements

3.2.1. Related Real-world Objects

The real-world objects are applications, inputs and outputs, display windows, and HATS-SML. Requirements related to these objects are described here.

3.2.1.1. Application

The HATS-GUI manages several different types of files (Table 3). Some of these files are input to the transformation process, and others are output from the transformation process. The HATS-GUI does not apply transformations. It invokes services from HATS-SML to accomplish transformations. (See Appendix B for data flow descriptions of the transformation process.) A general description of files and applications is presented in Section 2.

[SRSreq 50] The HATS-GUI shall allow a user to create an application minimally consisting of a lexical specification file, a grammar file, a user-defined functions file, and application configuration information. The user shall be able to name this application.

[SRSreq 51] All files associated with a given application shall be stored in one directory.

[SRSreq 52] The directory in which application files are stored shall be named with the application name.

[SRSreq 53] There shall be exactly one file describing the lexical specification of inputs associated with an application. It shall have the file extension *.spec*.

[SRSreq 54] There shall be exactly one file describing the grammar of inputs associated with an application. It shall have the file extension *.grm*.

[SRSreq 55] There shall be exactly one file describing the interpretation of function symbols associated with an application. This file shall be written in ML and have the file extension *.lib*.

[SRSreq 56] <deleted>

[SRSreq 57] The HATS-GUI shall suggest a file name for the user whenever it prompts the user to enter a file name. The HATS-GUI shall suggest the same file name for files copied to a different directory. The HATS-GUI shall suggest a file name with a sequentially numbered extension for files copied to the same directory. For example, if *NatLang.txt* exists, a suggested file name would be *NatLang.txt.2*.

[SRSreq 58] An application shall have associated with it an application configuration. The application configuration elements are given in Table 6.

Table 6: Application Configuration Elements

Element	Description
File Associations	• The grammar and lexical specification files are associated with the parser. • The parsed transformation language program file is associated with the transformation language program file. • The parsed target file is associated with the target file. • The transformation output SDT file names are associated with the transformation language program file. • The library and program parser are associated with the transformation language program file. • The input target files are associated with a transformation language program. • The transformation language program files are associated with the output SDT file. • The two most-recent pretty-print style files used to format transformation output are associated with the output SDT file. • The target file and pretty-print style files are associated with the text output file resulting from a transformation.
Editor Configurations	Command sequences to initiate editors under the Host OS
Editor Associations	Associations of editors to file types Associations of editors to specific files
Node Display	Colors for displayed node types Shapes for displayed node types
Node expansion	Number of nodes to expand when expanding displayed nodes
Parser file	Name of the parser file associated with the language of the application.

3.2.1.2. Input and Output

The types of outputs generated by HATS-SML are described in section 2.1.4.

[SRSreq 59] The HATS-GUI shall display output written to the standard output and standard error streams in a tab-selectable window. Text in this window shall be cleared prior to executing individual HATS-SML programs, unless the programs are contained in a program script. In the case of a script, the window contents shall be cleared prior to executing the script. Refer to section 3.2.3.2 for a discussion of the execution of script files.

[SRSreq 60] <deleted>

[SRSreq 61] <deleted>

3.2.1.3. HATS-SML

Refer to Section 3.1.3 for the requirements related to HATS-SML.

3.2.1.4. Display Windows

[SRSreq 62] The system shall open a new window when the user selects to display output.

[SRSreq 63] Display windows opened by the system shall have buttons for closing the windows.

[SRSreq 64] A navigation window shall be available. Navigation windows are discussed in section 3.2.3.3.4.

[SRSreq 65] Each active display window shall have a mouse cursor, which indicates the location of the pointing device in the window.

[SRSreq 66] Each display window shall have a keyboard cursor to track the current location in the display.

[SRSreq 67] The keyboard cursor shall be movable by the arrow keys.

[SRSreq 68] The keyboard cursor shall be set to the location of the mouse cursor when the pointing device is clicked.

3.2.2. Stimulus/Response

This section details the response the HATS-GUI shall make when it receives stimulus either from the user, the Host OS, or HATS-SML.

3.2.2.1. Menu Items

3.2.2.1.1. File Menu

[SRSreq 69] When a user selects the *File/Select* menu item, the HATS-GUI shall provide the user with a list of files in the currently open application and allow the user to select a file. This file becomes the *currently selected* file. If no application is currently open, the HATS-GUI will prompt the user to select and open an application, then prompt the user to select a file.

[SRSreq 70] When a user selects the *File/Save As* menu item, the HATS-GUI shall prompt the user to enter a new name. A new file shall be created in the currently open application with the newly entered file name. The contents of the selected file shall be written into the new file, and the new file shall become the currently selected file.

[SRSreq 71] When a user selects the *File/Delete* menu item, the HATS-GUI shall prompt the user to confirm the deletion operation. If the user confirms, the currently selected file shall be deleted. No file shall be currently selected at the end of this operation.

[SRSreq 72] When a user selects the *File/Edit* menu item, the HATS-GUI shall request that the operating system start a process and load that process with the editor specified by the application configuration. The name of a selected file shall be passed to the editor as a command line argument. The editor started by the HATS-GUI shall be the editor associated with the file in the application configuration, or if there is no editor associated with the file, then the editor associated with the file type of the file in the application configuration. If no editor is associated with either the file or the file type, then the default editor shall be used. If no default editor has been selected, then the HATS-GUI shall present a list of the configured editors and allow the user to select an editor. If no editors have been configured, then the HATS-GUI shall inform the user that an editor must be configured before starting an editor.

[SRSreq 73] When a user selects the *File/Exit* menu option, the HATS-GUI shall request the host operating system to kill all currently active HATS-SML processes started by the HATS-GUI and halt the HATS-GUI process. If changes have been made to the configuration but have not been saved, the user shall be prompted to save the configuration. If the user agrees, the configuration shall be saved. If the user does not agree, the configuration shall not be saved.

3.2.2.1.2. Application Menu

[SRSreq 74] When a user selects the *Application/Select* menu item, the HATS-GUI shall display a list of directory names in the current working directory, allow a user to navigate the directory structure, and allow a user to select a previously created application. This application becomes the *currently selected* application.

[SRSreq 75] When a user selects the *Application/Close* menu item, the HATS-GUI shall close the currently selected application. No application is currently selected at the end of this operation. If changes have been made to the configuration but have not been saved, the user shall be prompted to save the configuration. If the user agrees, the configuration shall be saved. If the user does not agree, the configuration shall not be saved.

[SRSreq 76] When a user selects the *Application/Save* menu item, the HATS-GUI shall save all application configuration information to nonvolatile storage.

[SRSreq 77] When a user selects the *Application/Save As* menu item, the HATS-GUI shall prompt the user for a new application name. The HATS-GUI shall create a new directory at the same level as the currently selected application. All files in the application directory, including all subdirectories, shall be copied to the new directory, and the new directory shall become the currently selected application.

[SRSreq 78] When a user selects the *Application/Delete* menu item, the HATS-GUI shall prompt the user to confirm the deletion operation. If the user confirms, all files in the application directory shall be deleted and the application directory itself shall be deleted. No application is currently selected at the end of this operation.

[SRSreq 79] When deleting an application directory, if the application directory contains a subdirectory, the system shall confirm the deletion of the subdirectory with the user before continuing. If the user does not confirm the deletion of the subdirectory, the application director shall not be deleted.

3.2.2.1.3. Configure Menu

[SRSreq 80] When a user selects the *Configure* menu options, the HATS-GUI shall provide a list of elements in the application configuration, allow the user to select an element, then prompt the user for a value for the configuration element. The application configuration elements are given in Section 3.2.1.1. This information becomes part of the application configuration.

3.2.2.1.4. Run Menu

[SRSreq 81] When a user selects the *Run/Generate Parser* menu option, the HATS-GUI shall generate the parser using the MakeParser HATS-SML program as described in Appendix D.

[SRSreq 82] When a user selects the *Run/Parse Target* menu option, the HATS-GUI shall parse the selected target program by using the ParseTarget HATS-SML programs as described in Appendix D. If no target program is selected, the HATS-GUI shall first prompt the user to select a target program.

[SRSreq 83] When a user selects the *Run/Parse Program* menu option, the HATS-GUI shall parse the selected transformation language program by using the ParseTlp HATS-SML programs as described in Appendix D. If no target program is selected, the HATS-GUI shall first prompt the user to select a transformation language program.

[SRSreq 84] When a user selects the *Run/Execute Transforms* menu option, the HATS-GUI shall execute the ApplyTransformations HATS-SML programs as described in Appendix D. This process is further described in Section 3.2.3.

[SRSreq 85] When a user selects the *Run/Pretty-print* menu option, the HATS-GUI shall execute the Pretty-print HATS-SML programs as described in Appendix D. This process is further described in Section 3.2.3.

3.2.2.1.5. View Menu

[SRSreq 86] When a user selects the *View/SDT* menu option, the HATS-GUI shall prompt the user to select an SDT file in the current application. The file list shall include SDTs in the application. The viewing of outputs is detailed in Section 3.2.3.

[SRSreq 87] When a user selects the *View/Text* menu option, the HATS-GUI shall prompt a user to select a pretty-printed text file in the current application. The viewing of text outputs is detailed in Section 3.2.3.

[SRSreq 88] When a user selects the *View/Standard Error* menu option, the HATS-GUI shall display the standard error screen showing all outputs written to the standard error stream since the most recent clearing of this data. The viewing of errors is detailed in Section 3.2.3.

[SRSreq 89] When a user selects the *View/Standard Output* menu option, the HATS-GUI shall display the standard error screen showing all outputs written to the standard output stream since the most recent clearing of this data. The viewing of output is detailed in Section 3.2.3.

3.2.2.2. Mouse and Keyboard actions

Section 3.2.3.3.4 details keyboard and mouse selection and navigation.

[SRSreq 90] When viewing an SDT, a single left mouse click on an expandable displayed node shall expand the node by displaying the node's children or collapse the node by hiding the node's children.

[SRSreq 91] When a displayed node is expanded, the number of levels of children added to the displayed graph shall be determined by the application configuration.

[SRSreq 92] When viewing an SDT, a single left mouse click on an already-expanded displayed node shall collapse the displayed node. When a displayed node is collapsed, all descendants of the node are removed from the displayed graph.

3.2.2.3. HATS-SML Completion

[SRSreq 93] The HATS-GUI shall collect process exit codes from HATS-SML processes. Process exit codes of 0 indicate nominal completion of the process. Process exit codes other than 0 indicate failure.

3.2.3. Related Features

3.2.3.1. Prepare Application

Selecting, deleting, and copying applications are described in Section 3.2.2. Editing files is discussed in Section 3.2.2.1.1.

[SRSreq 94] The GUI shall allow the user to import a file into the selected application. Importing a file consists of copying the file into the application directory.

3.2.3.2. Parse/Execute Transformation

The HATS-GUI shall facilitate parsing transformation language programs and target files. The HATS-GUI shall orchestrate executing transformation language programs. Parsing is accomplished by sending a parser file and an input file to HATS-SML.

To execute a transformation language program, the parsed transformation language program and the user-defined library file are sent to HATS-SML. During execution of a transformation language program, HATS-SML writes a sequence of new SDTs. These output SDTs are specified in the transformation language program. The HATS-GUI shall make these data sets available for viewing and/or saving as permanent files in the application. In addition to the output file, the transformation language program may specify that debugging output be written to the standard output stream. Errors encountered during program execution are written to the standard error stream. The HATS-GUI shall capture data written to these streams and make that data available to the user.

Parsing and running transformations are also described in section 3.2.2.

[SRSreq 95] The HATS-GUI shall use the protocol and program signatures listed in Appendix D when executing HATS-SML programs.

[SRSreq 96] To perform a transformation in an application, the HATS-GUI shall execute the following sequence of actions:

- The HATS-GUI shall confirm that the transformation language program exists. (It is an error if the program does not exist.)
- The HATS-GUI shall check that the parser file is younger than the grammar and lexical specification files and that the parser file was generated from the grammar and lexical specification files. If this is not the case, the HATS-GUI shall generate a new parser file.
- The HATS-GUI shall check that a parsed version of the transformation language program exists and is younger than the parser file and the transformation language program file. If it is not, the HATS-GUI shall generate a new parsed version of the transformation language program.
- The HATS-GUI shall start the *FindTarget* program using the transformation language program as input.

- The HATS-GUI shall take the list of target program files generated by *FindTarget*, and for each file in the list it shall check that a parsed version of the target program exists and is younger than the target parser file and the target program file. If it is not, the HATS-GUI shall generate a new parsed version of the target program.
- The HATS-GUI shall start the *ApplyTransformation* program in HATS-SML and pass the parsed transformation language program file name, the user defined library file name, and the output file name to the *ApplyTransformation* program. The HATS-GUI shall redirect the standard output and standard error streams and capture any text written to those streams.

[SRSreq 97] <deleted>.

[SRSreq 98] The HATS-GUI shall allow the user to continue working while transformations are taking place. Thus a user shall be able to edit files, view output, and transmit other problems to HATS-SML while waiting for HATS-SML to generate output.

[SRSreq 99] The HATS-GUI shall allow the user to execute a *script* file. A script file shall contain a linear sequence of HATS-SML programs. The HATS-GUI shall begin with the first command and execute the commands sequentially until either an error occurs or all commands in the script file have been executed.

[SRSreq 100] Prior to starting the execution of a script file, the display screens for the standard error and the standard output shall be cleared.

3.2.3.3. Display Transformation Output

There are several types of output that can be received from HATS-SML:

- error messages written to the standard error stream resulting from parser generation, parsing, or program execution;
- parsed transformation language programs;
- parsed target files;
- transformation language program parser files;
- target program parser files;
- text output written to the standard output stream; and
- pretty-printed text.

[SRSreq 101] The HATS-GUI shall allow a user to select output files in the application to display.

[SRSreq 102] The HATS-GUI shall allow a user to display selected output files. The manner in which the output is displayed shall depend on the type of the output.

[SRSreq 103] <deleted>

3.2.3.3.1. Displayed Node and Text Selection

[SRSreq 104] The user shall be able to select an arbitrary set of displayed nodes in a displayed graph. Selected displayed nodes shall be indicated visually. The following methods shall be used to select displayed nodes.

- A single node shall be selected by right-clicking the mouse cursor on the node.
- A single node shall be selected by moving the keyboard cursor to the node (using the tab key) and pressing the enter key.
- Clicking and dragging the pointing device selection rectangle around a set of displayed nodes shall select a set of adjacent nodes.
- A set of nodes shall be selected by holding the shift key while selecting nodes using the methods specified previously in this section.

[SRSreq 105] The user shall be able to deselect all selected displayed nodes by

- clicking either the right or left mouse buttons while the mouse cursor is not pointed at any displayed nodes and the shift key is not depressed; or

- by pressing the *escape* key.

[SRSreq 106] When viewing an SDT, selecting nodes shall result in the highlight of all displayed nodes in the smallest well-formed tree of the selected nodes.

[SRSreq 107] When the system displays a navigation window, the user shall be able to select a current SDT location by clicking the mouse cursor in the navigation window. The center of the displayed graph will be the displayed node closest to the mouse cursor when the mouse is clicked.

[SRSreq 108] When both an SDT and a pretty-printed text display corresponding to a single transformed target file are open, highlighting and cursor location motion shall be mirrored in the two windows. Thus, when a displayed node is selected in the SDT display, the corresponding text in the pretty-print window is highlighted. When the user scrolls to the end of the pretty-printed text display, the display of the SDT should also scroll to the corresponding displayed nodes.

3.2.3.3.2. Display Windows

[SRSreq 109] The user shall be able to resize windows dynamically. When displayed graph windows are resized, the HATS-GUI shall respond by utilizing the space available.

[SRSreq 110] The display of an SDT shall also result in the display of text from leaf nodes in a text display area.

[SRSreq 111] The text displayed in the text display area shall correspond to the concatenation of the node labels (separated by spaces) of the leaf nodes of the displayed graph (in-order traversal, left to right). If a leaf displayed node corresponds to a non-terminal SDT node, the label in the text window shall be surrounded by pointed brackets (<>). For example, if a non-terminal *TERM* appears in a leaf node, its text representation shall be <*TERM*>. Appendix G gives an example of the display of text associated with an SDT.

[SRSreq 112] The HATS-GUI shall provide for the copying of text from the text display area to a system buffer available for pasting into other applications (such as an editor).

[SRSreq 113] The user shall be able to display more than one SDT at a time. Each SDT shall be displayed in a separate window.

3.2.3.3.3. Display of SDTs

In the following we make the distinction between an SDT and a displayed graph. An SDT is an abstract tree containing a set of nodes. A displayed graph is a graphical representation of a subset of the nodes in an SDT. The displayed graph places the visual representation of a node above the visual representation of its descendant nodes. In a displayed graph, a displayed node that does not have any displayed descendant nodes is a leaf node. This is not to be confused with terminal nodes in the SDT. In the SDT, there are two types of nodes: terminals and non-terminals. Terminal nodes correspond to tokens in the language. Non-terminal nodes correspond to elements on the left-hand side of productions of the grammar of the language. The types of nodes in a displayed graph are listed in Table 4.

[SRSreq 114] The HATS-GUI shall provide for the manipulation and display of SDTs.

[SRSreq 115] SDTs shall be displayed as directed acyclic graphs with nodes and edges. Nodes in the displayed graph correspond to nodes in the SDT. Edges in the displayed graph correspond to the parent-child relationship between nodes in the SDT.

[SRSreq 116] Initially, a displayed graph will have a single displayed node. That node will correspond to the root of the SDT.

[SRSreq 117] When a displayed node and its children are displayed simultaneously, the parent node shall be displayed higher than the children.

[SRSreq 118] All displayed children of a single node shall appear at the same height.

[SRSreq 119] The user shall be able to control the display by choosing the color and shapes of displayed nodes based on node type. The HATS-GUI shall use the shape and color of displayed node types stored in the application configuration to display nodes.

[SRSreq 120] The user shall be able to expand displayed nodes (provided it is possible to expand the selected nodes). The HATS-GUI shall change the graph display by adding the descendant nodes to the displayed graph (thus changing the expandable nodes to internal nodes).

[SRSreq 121] The user shall be able to collapse displayed nodes. The HATS-GUI shall change the displayed graph by removing all descendants of the selected displayed nodes (thus changing the internal nodes to non-expandable leaf nodes).

[SRSreq 122] The user shall be able to hide a set of selected displayed nodes. The HATS-GUI shall change the display by removing the selected nodes from the display and changing the parent and descendant nodes of the removed nodes to nodes with hidden descendants and parents, respectively.

[SRSreq 123] The user shall be able to issue an unhide instruction that reverses the hide instruction for all displayed nodes.

[SRSreq 124] After issuing the unhide instruction, a user shall be able to issue a rehide instruction that reverses the unhide instruction. All displayed nodes hidden before the most recent unhide instruction shall become hidden.

[SRSreq 125] The HATS-GUI shall allow the user to expand the SDT by a predetermined number of node levels. The number of levels expanded is set in the application configuration.

[SRSreq 126] The HATS-GUI shall provide the function to fully expand a sub-tree regardless of the node expansion level in the application configuration.

[SRSreq 127] The user shall be able to view multiple SDTs simultaneously.

[SRSreq 128] SDT displays shall also provide the user with a view of text associated with the leaves of the displayed sub-tree. This text shall be copy-and-pasteable in the host operating system. Thus, text displayed in a HATS-GUI window can be pasted into other applications such as text editors that may be running concurrently with the HATS-GUI.

[SRSreq 129] SDT displays shall provide the user with the option to refresh the display. When a display is refreshed, the current display is erased, the SDT file is read, and a new SDT display is generated. The new SDT display shall approximate the previous SDT display by expanding the SDT to the same number of nodes, if this is possible.

3.2.3.3.4. Navigation

[SRSreq 130] If a displayed graph or a text display is too large to fit inside its window, the windows shall have scroll bars attached to them. Dragging scrollbar buttons shall scroll the display. Pressing the arrow keys of the keyboard shall also scroll the display.

[SRSreq 131] For text windows, the keyboard cursor location shall move according the following rules:

- A left arrow key moves the cursor to the left one character. If the cursor is at the beginning of a line, the cursor is moved to the end of the previous line. If there is no previous line, the cursor does not move.
- A right arrow key moves the cursor to the right one character. If the cursor is at the end of a line, the cursor is moved to the first character of the next line. If there is no line after the current line, the cursor is not moved.
- An up arrow key moves the cursor up one line. If there is text in the same column, the column does not change. If there is no text in the column above the current cursor location, the cursor is moved to the end of the previous line. (A space is considered text.)
- A down arrow key moves the cursor down one line. If there is text in the same column, the column does not change. If there is no text in the column below the current cursor location, the cursor is moved to the end of the previous line. (A space is considered text.)

[SRSreq 132] If keyboard cursor movement causes the keyboard cursor to move to text or graph nodes not currently displayed, the display will scroll the minimal amount to display the current cursor location.

[SRSreq 133] For displayed graph windows, the keyboard cursor location shall move according the following rules:

- A left arrow key moves the cursor to the next unhidden node to the left. The node to the left does not need to be a sibling of the current node, but it does need to be at the same level as the current node. If there are no nodes to the left, the cursor does not move.
- A right arrow key moves the cursor to the next unhidden node to the right. The node to the right does not need to be a sibling of the current node, but it does need to be at the same level as the current node. If there are no nodes to the right, the cursor does not move.
- An up arrow key moves the cursor to the closest ancestor that is not hidden. If there is no such node, the cursor does not move.
- An down arrow key moves the cursor to the leftmost, nearest descendant that is not hidden. If there is no such node, the cursor does not move.

[SRSreq 134] The pointing-device cursor shall track the motion of the pointing device. The keyboard cursor shall move to the text element or graph node closest to the pointing-device cursor when the pointing device is clicked.

[SRSreq 135] If the displayed graph of the entire SDT is too large for the window, a separate, small window with a compressed view of the SDT shall be displayed. The compressed view is called the *navigation window* display. In this navigation window, a representation of the entire SDT will be drawn. A small box will be displayed showing the part of the SDT currently displayed in the main tree display window.

[SRSreq 136] The user shall be able to turn the navigation window on and off.

[SRSreq 137] The HATS-GUI shall provide the user the capability to search SDTs. The search criteria are described in Appendix E. The user shall enter a search pattern (see Appendix E and the following requirement). The HATS-GUI shall search the SDT for a matching pattern. When a match is found, the HATS-GUI shall highlight the matching displayed nodes and center the display on the left-most, top-level displayed node in the matching pattern. The keyboard cursor is set to this node. If no matches are found, the HATS-GUI shall display a message stating "No Match Found."

[SRSreq 138] To initiate an SDT search, the user shall be able to use the following sequence of actions.

- The user selects a set of nodes using the mouse cursor.
- The user presses the Ctrl-Insert key combination.
- The user selects the tree search option.
- The HATS-GUI presents a prompt for entering the search pattern.
- The user presses the Ctrl-Shift-Insert key combination.
- The HATS-GUI fills the search pattern entry box with text concatenated from the nodes highlighted at the time of the Ctrl-Insert key press.
- The user selects initiation of search.

[SRSreq 139] When searching for SDT patterns, search begins at the SDT node corresponding to the node at the current keyboard cursor location. This becomes the original starting location. Search continues top-to-bottom, left-to-right until the end of the SDT is encountered. When the end of the SDT is encountered, search will continue from the root of the SDT until arriving back at the starting location.

[SRSreq 140] The HATS-GUI shall provide for repeating a search. Repeating a search shall find the next matching sub-tree starting from the current keyboard cursor location. Search terminates when either a matching sub-tree is found or the search arrives at the original starting location.

[SRSreq 141] The HATS-GUI shall provide for repeating a search in the reverse direction. Searching in the reverse direction continues right-to-left, bottom-to-top. Search terminates when either a matching sub-tree is found or the search arrives at the original starting location.

[SRSreq 142] The HATS-GUI shall provide the user the capability to search the text display for text sub-strings. The search criteria are described in Appendix F. The user shall be prompted to enter a text string describing a string pattern. The HATS-GUI shall search the text for a matching pattern. When a match is found, the HATS-GUI shall highlight the matching text and center the display on the left-most element of the matching text. If no matches are found, the HATS-GUI shall display a message stating "No Match Found."

[SRSreq 143] When searching for text patterns, search begins at the current keyboard cursor location and continues left-to-right, top-to-bottom until the end of the text is encountered. When the end of the text is encountered, search will continue from the start of the text until arriving back at the starting location.

3.2.3.3.5. Display of Pretty-Printed Text

Pretty-printed text is text formatted with white space and line breaks. Pretty-printed text is generated from the terminal nodes of an SDT by applying a pretty-print style to an SDT. The HATS pretty-printer is a general transformation system. The output of the pretty-printer may contain text not found in terminal nodes of the SDT and may not contain text from every terminal SDT node. In the most extreme cases, the output from the pretty printer may not have any tokens from the terminal nodes of the input SDT. The lack of correspondence of terminal node text to pretty-printed text may make association actions on displayed graphs to pretty-printed text difficult. In the following, we assume that terminal-node text appears in the output from the pretty-printer.

[SRSreq 144] The formatting of the displayed pretty-printed text shall be faithful to the formatting represented in the pretty-printed text file. No additional white space shall be inserted, and a fixed width font shall be used to display the text.

[SRSreq 145] The user shall be able to select pretty-printed text files to display.

[SRSreq 146] Selected pretty-printed text shall be displayed in a separate window.

[SRSreq 147] Windows for pretty-printed text shall allow for scrolling left, right, up, and down if the text is too large to display in the window.

[SRSreq 148] The HATS-GUI shall provide the user the capability to search the text display for text sub-strings. The search criteria are described in Appendix F. The user shall be prompted to enter a text string describing a string pattern. The HATS-GUI shall search the text for a matching pattern. When a match is found, the HATS-GUI shall highlight the matching text and center the display on the left-most element of the matching text. The keyboard cursor shall be set to this location. If no matches are found, the HATS-GUI shall display a message stating "No Match Found."

[SRSreq 149] When searching for text patterns, search begins at the location of the keyboard cursor and continues left-to-right, top-to-bottom until the end of the text is encountered. When the end of the text is encountered, search will continue from the start of the text until arriving back at the starting location.

3.2.3.3.6. Display of Errors

[SRSreq 150] The HATS-GUI shall not halt as a result of any error messages received from HATS-SML.

[SRSreq 151] The HATS-GUI shall collect error messages from HATS-SML in a single display area. This display area shall be initially clear, and shall contain all error messages generated by HATS-SML since the HATS-GUI was started. The display area shall be a window accessible via a window tab. The display shall list all error received from HATS-SML in the order received.

[SRSreq 152] Errors shall not be displayed until requested by the user.

[SRSreq 153] The user shall be able to clear the error window without restarting the HATS-GUI.

3.2.3.3.7. Display of SDT and Pretty-Printed Text from Common Target file

[SRSreq 154] When both an SDT and a pretty-printed text display corresponding to a single transformed target file are open, navigation in the SDT window shall result in navigation in the pretty-printed text window. Searching, selecting, and cursor location motion should be mirrored in the two windows. Specifically, when a node is selected in the displayed graph, the corresponding text in the pretty-print window is highlighted. For this purpose, the corresponding text is all the text that corresponds to terminal nodes of the SDT that are descendants of the SDT nodes that correspond to the selected display nodes.

[SRSreq 155] When both an SDT and a pretty-printed text display corresponding to a single transformed target file are open, navigation in the pretty-printed text window shall result in navigation in the SDT display. Selecting, and cursor location motion should be mirrored in the two windows. Specifically, when text is selected in the text window, the displayed nodes corresponding nodes in the SDT are highlighted.

3.3. Non-behavioral Requirements

3.3.1 Performance Requirements

[SRSreq 156] The HATS-GUI shall display and SDT with 10,000 nodes in five seconds or less.

[SRSreq 157] The HATS-GUI shall display 1,000,000 node SDTs in the same amount of time, plus or minus two seconds, as it takes to display a 10,000 node SDT.

[SRSreq 158] The HATS-GUI shall allow a user to request transformations while HATS-SML is performing transformations or parsing.

[SRSreq 159] The HATS-GUI shall allow a user to navigate through the display of an SDT with 10,000 nodes so that scrolling a screen width with any portion of the SDT displayed takes less than 3 seconds.

[SRSreq 160] The complexity of the text search algorithm shall not exceed $O(n^2)$ for n nodes.

[SRSreq 161] The complexity of the tree search algorithm shall not exceed $O(n^2)$ for n nodes.

[SRSreq 162] The complexity of the scrolling algorithms for text and displayed graphs shall not exceed $O(n^2)$ for n displayed nodes.

3.3.2 Qualitative Requirements

3.3.2.1 Security

[SRSreq 163] The user shall not be required to log in or authenticate his/her identity when using the system.

3.3.2.2 Portability

It is intended that the HATS-GUI run on any platform that supports ML and Java. HATS-SML version 2.0 is intended to eliminate non-portable communications techniques.

[SRSreq 164] The GUI shall run on Windows 2000, Sun Solaris 8, and Linux operating systems without modification of the Java code.

3.3.3 Design and Implementation Constraints

[SRSreq 165] The system shall be implemented in the Java 1.3 programming language.

[SRSreq 166] One algorithm for displaying an SDT has been implemented and found to be unacceptable:

- Draw SDT into large virtual window with a smaller viewing area;
- Zoom in on portions of the tree;

This algorithm results in excessive memory usage and slow performance. This algorithm shall not be used in the HATS-GUI.

3.4. Other Requirements

There are no other requirements at this time.

4. Appendix A: HATS-GUI Class Diagram

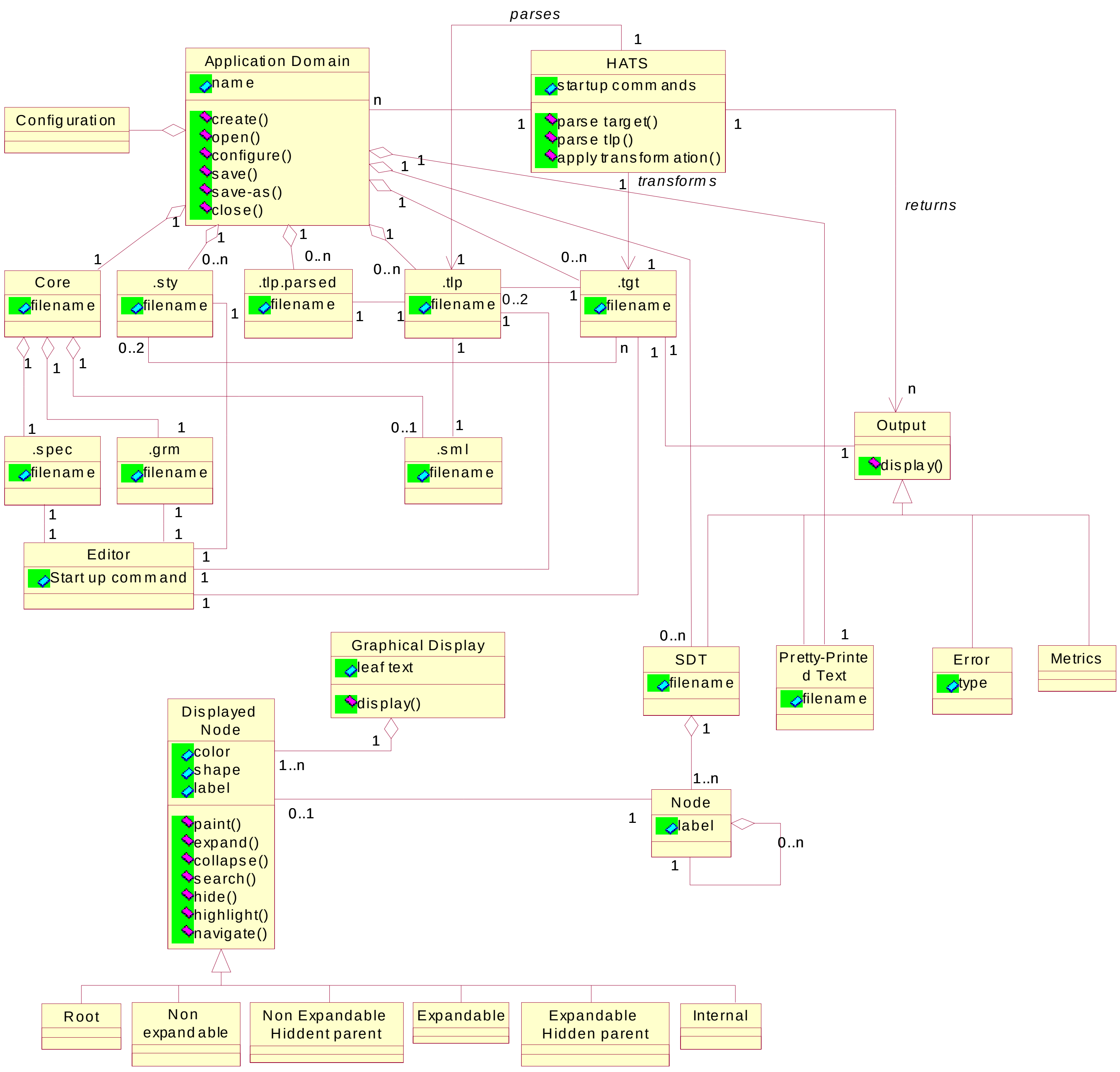


Figure 3: HATS-GUI Class Diagram

5. Appendix B: HATS-GUI Data Flow Diagram

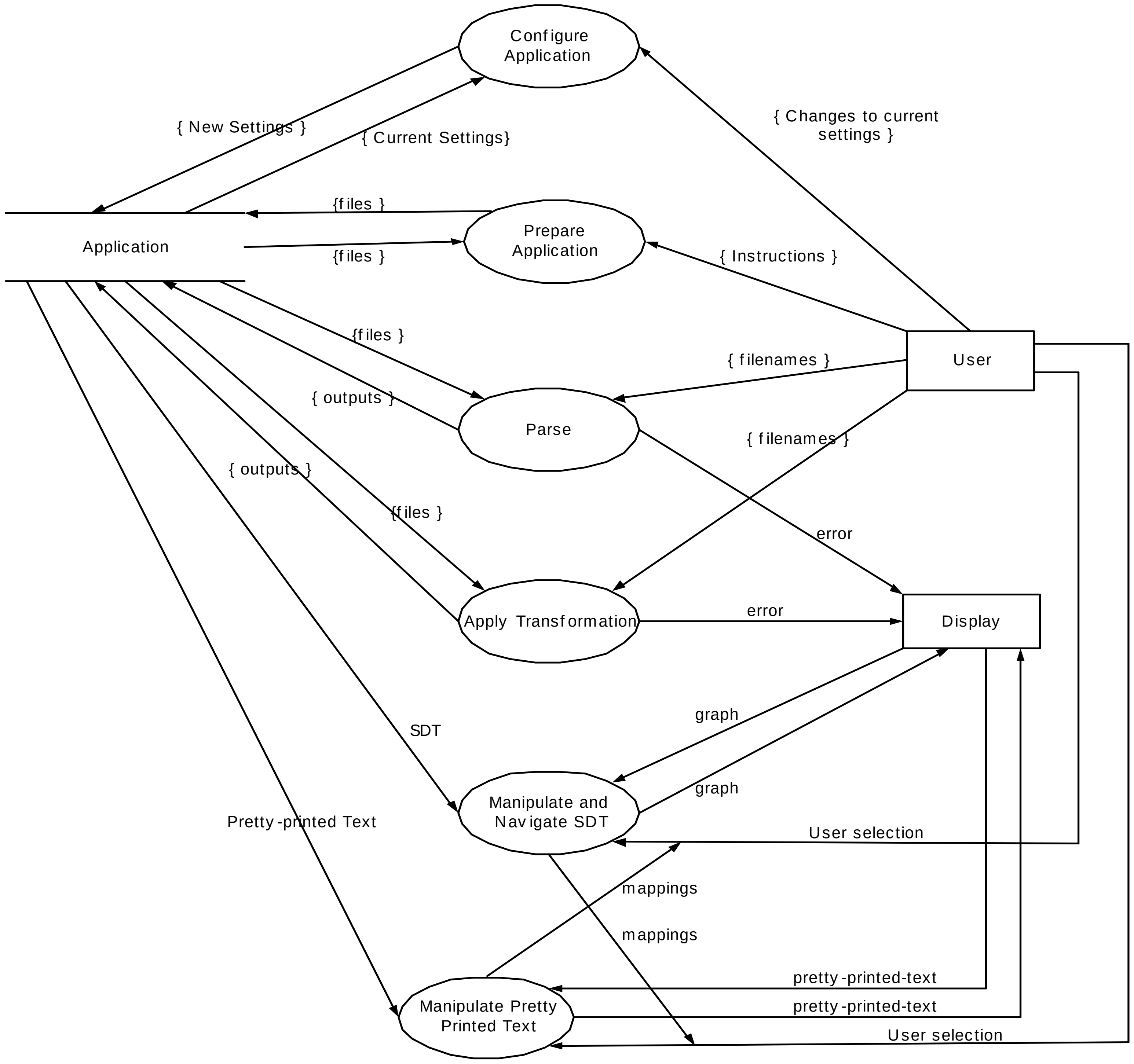


Figure 4: HATS-GUI Data Flow Diagram -- Level 1

HATS-GUI Data Flow Diagram – Level 2

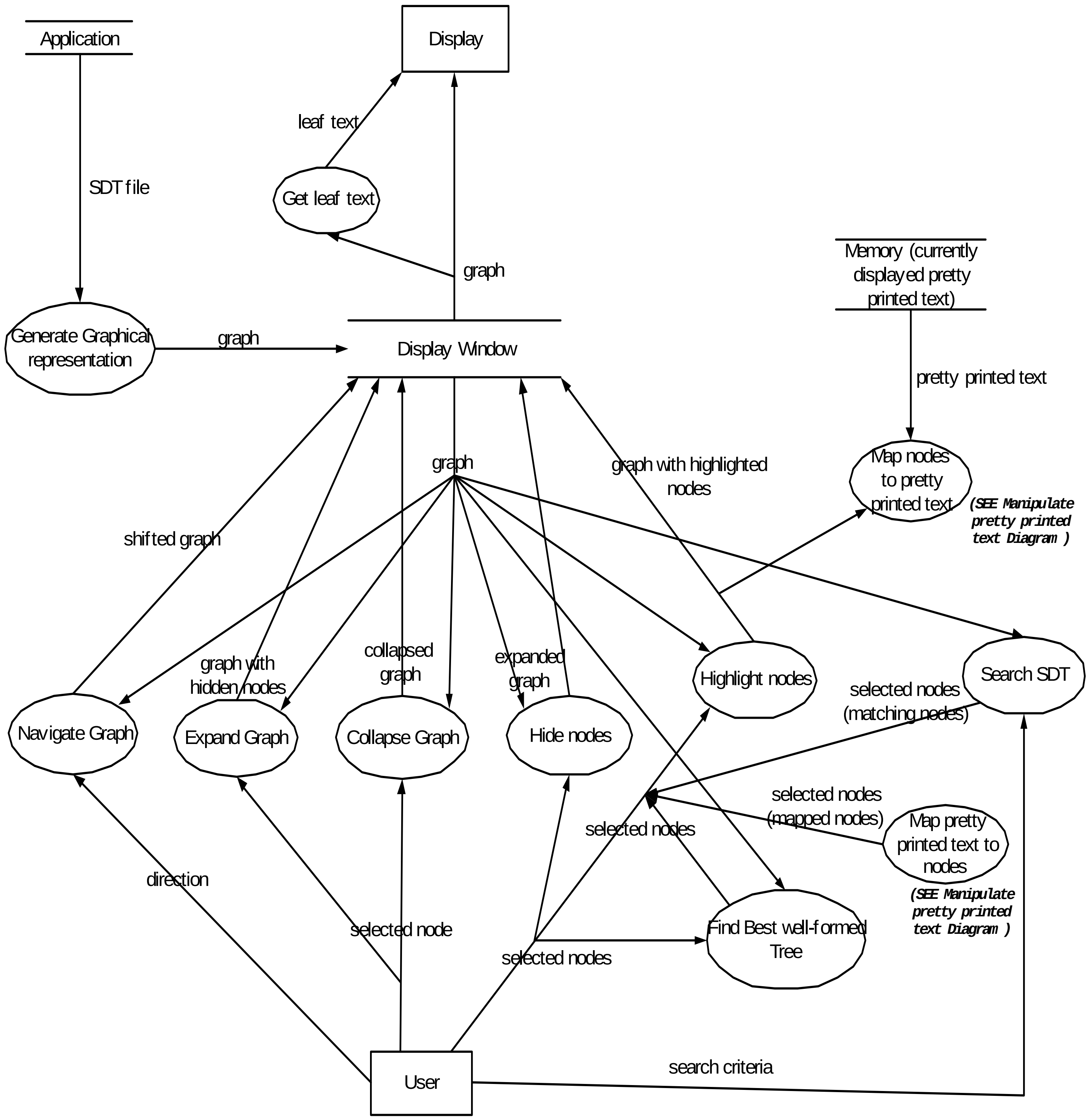

Application
Display
leaf text
SDT file
Get leaf text
graph
Memory (currently displayed pretty printed text)
Generate Graphical representation
graph
Display Window
pretty printed text
graph
graph with highlighted nodes
Map nodes to pretty printed text
(SEE Manipulate pretty printed text Diagram)
shifted graph
collapsed graph
expanded graph
graph with hidden nodes
Highlight nodes
Search SDT
selected nodes (matching nodes)
Navigate Graph
Expand Graph
Collapse Graph
Hide nodes
selected nodes (mapped nodes)
selected nodes
Map pretty printed text to nodes
(SEE Manipulate pretty printed text Diagram)
direction
selected node
Find Best well-formed Tree
selected nodes
search criteria
User

Figure 5: HATS-GUI Data Flow Diagram: Manipulate and Navigate SDT

HATS-GUI Data Flow Diagram – Level 2

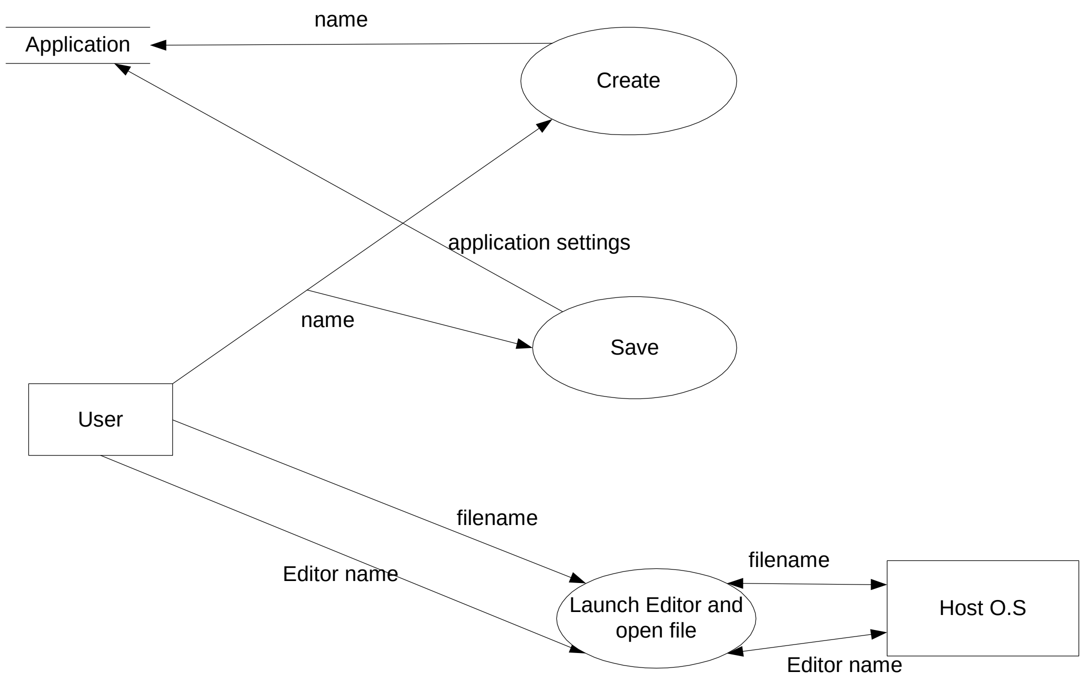


Figure 6: HATS-GUI Data Flow Diagram: Prepare Application

HATS-GUI Data Flow Diagram – Level 2

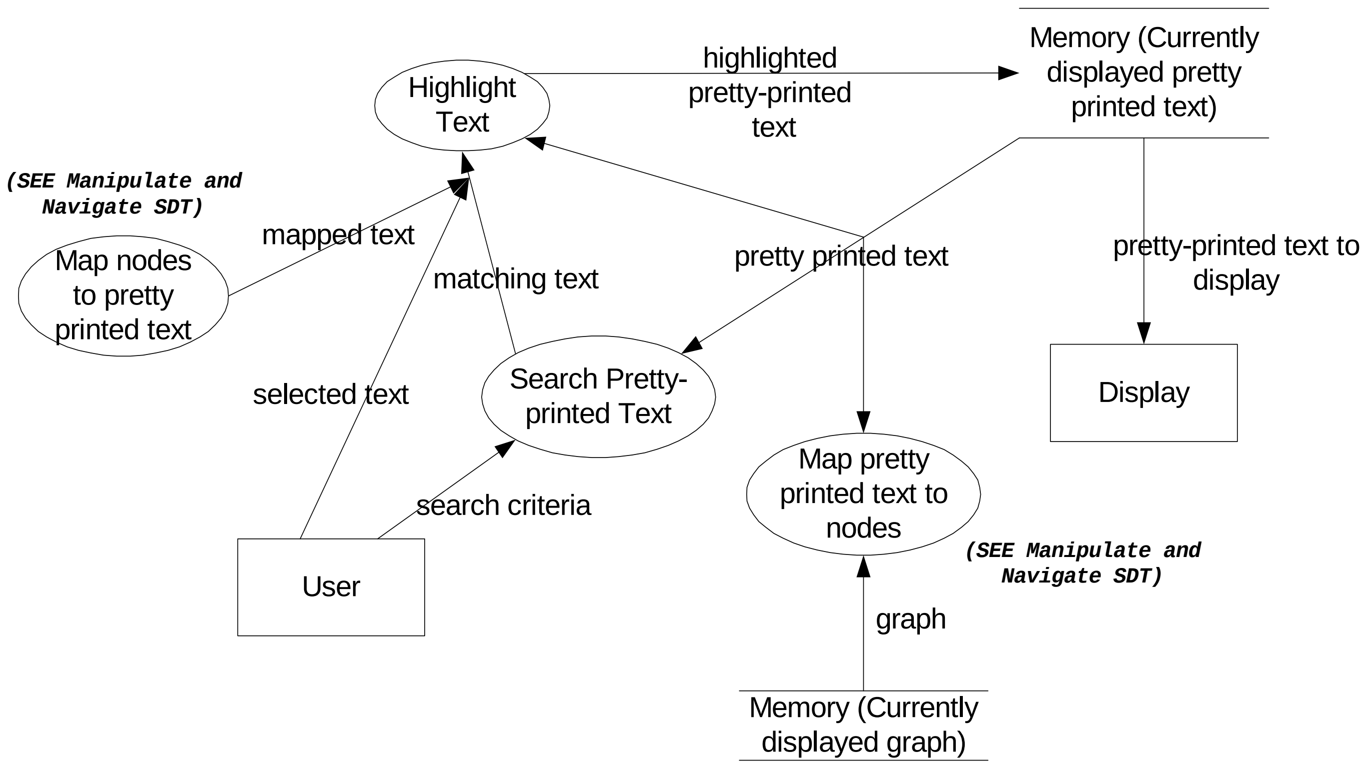

Highlight Text
highlighted pretty-printed text
Memory (Currently displayed pretty printed text)
(SEE Manipulate and Navigate SDT)
mapped text
Map nodes to pretty printed text
matching text
pretty printed text
pretty-printed text to display
Display
selected text
Search Pretty-printed Text
Map pretty printed text to nodes
search criteria
User
(SEE Manipulate and Navigate SDT)
graph
Memory (Currently displayed graph)

Figure 7: HATS-GUI Data Flow Diagram: Manipulate Pretty-Printed Text

HATS-GUI Data Flow Diagram – Level 2

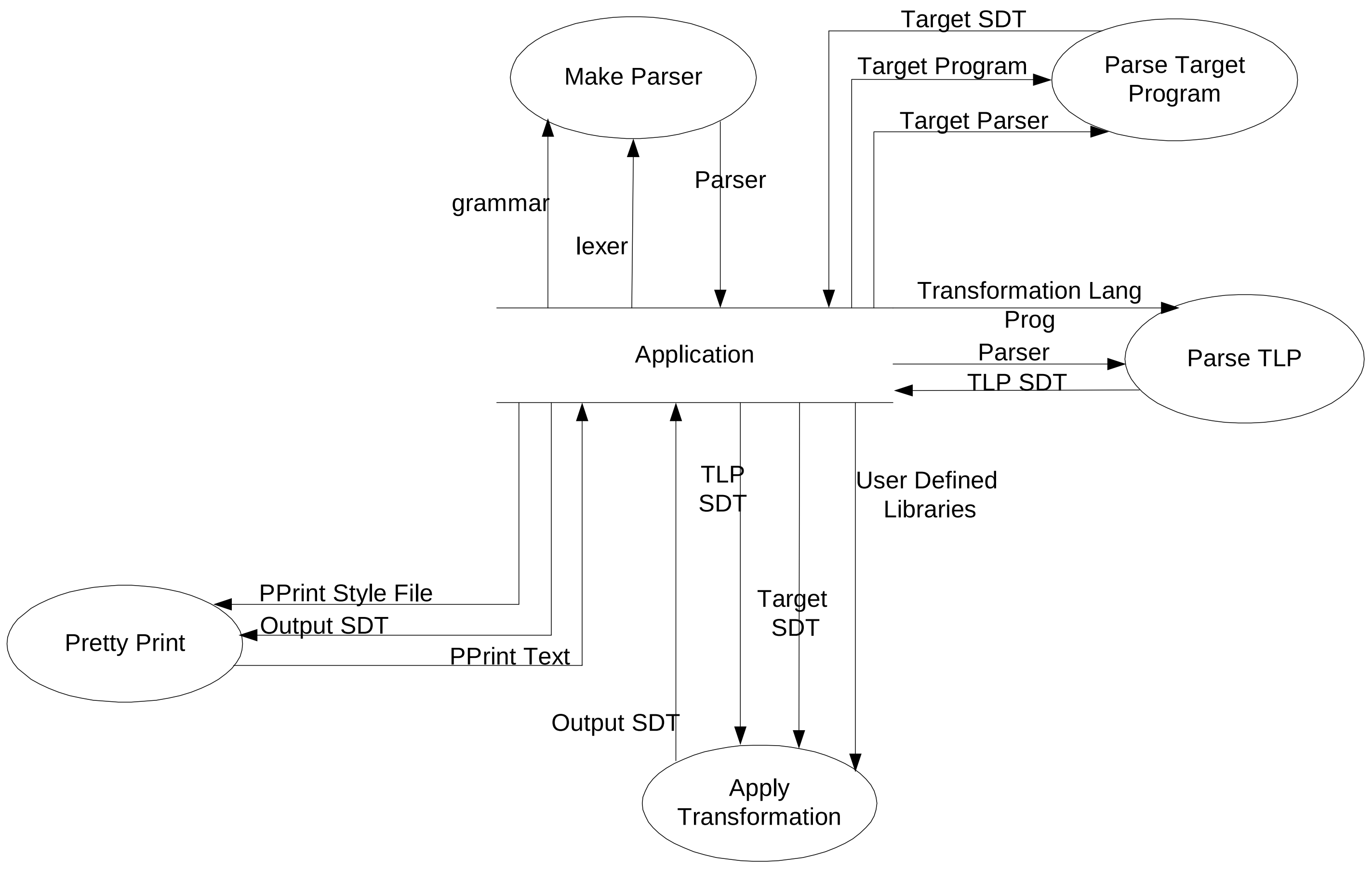


Figure 8: HATS-GUI Data Flow Diagram: Apply Transformation

6. Appendix C: HATS-GUI State Transition Diagrams

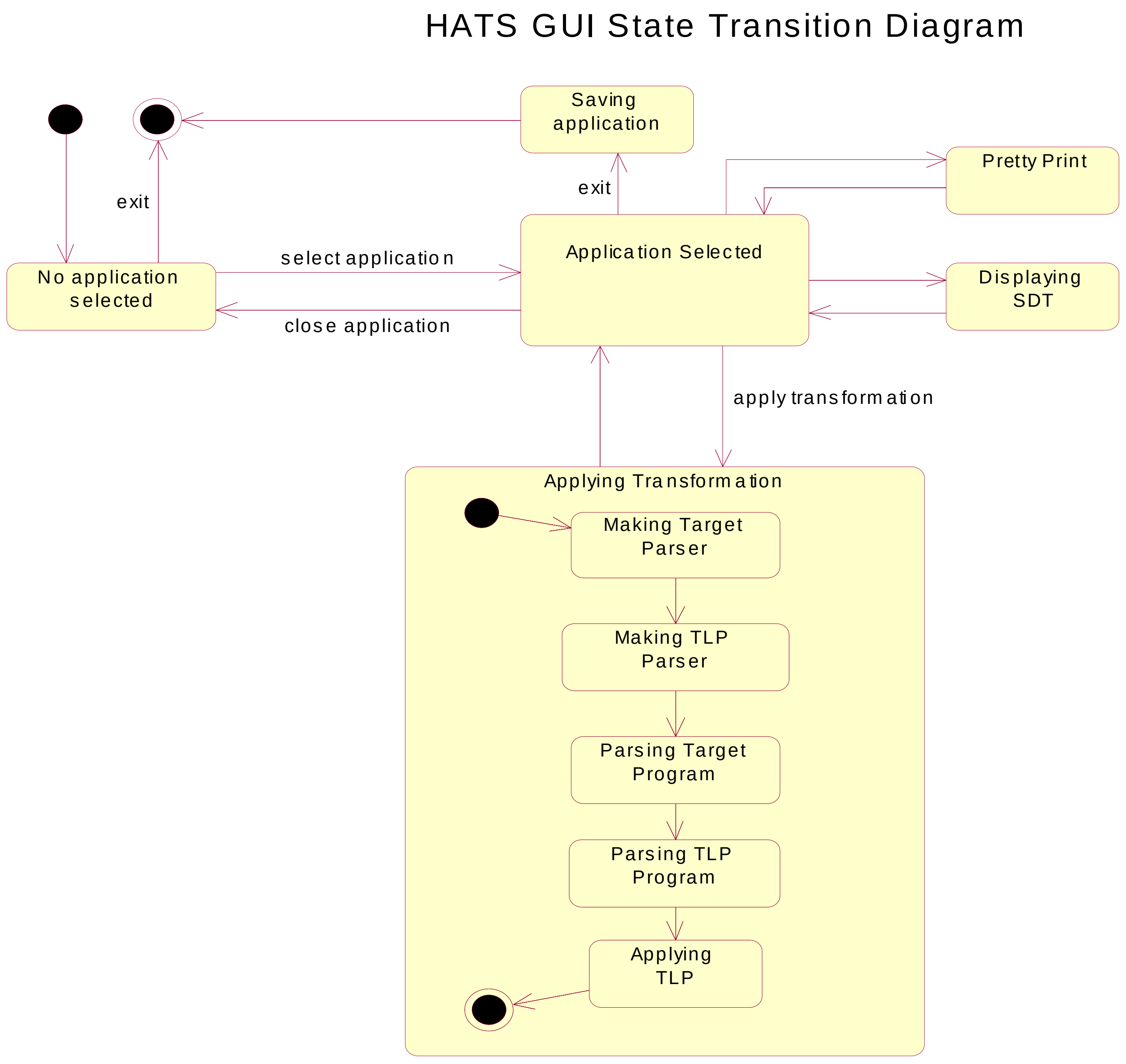


Figure 9: HATS-GUI State Diagram

7. Appendix D: HATS-SML Communication Protocol

HATS-SML is a collection of programs that are invoked with command line arguments. Collectively, they provide the functionality of HATS. Table 7 below lists the programs, their arguments, and a description of each program. The arguments to each program are positional, and specify input and output file names for each program. A file name may be a complete path name. A description of each file is given following Table 7.

Table 7: Summary of HATS-SML Programs

MakeParser <lexer> <grammar> <parser-program> Generates a parser for transformation language programs and target programs in a given language. MakeParser generates a parser file for translation language programs that is written to the file path specified by the third argument (parser-program). MakeParser returns 0 if the parser-program file generation is successful.
ParseTarget <parser-program> <target> <targetSDT> Generates a parsed target file from a target-parser file and a target. The parser-program file must have been generated previously by MakeParser. ParseTarget generates an SDT file representing the target. This file is written to the path specified by the third argument (targetSDT). This program returns 0 if the targetSDT file generation is successful.
ParseTLP <parser-program> <program> <programSDT> Generates a parsed transformation language program file from a translation language program parser (parser-program) and a transformation language program (program). The tlp-parser file must have been generated previously by MakeParser. ParseTLP generates an SDT file representing the translation program. This file is written to the path specified by the third argument (programSDT). This program returns 0 if the programSDT file generation is successful
FindTargets <program> <targetlist> Generate a list of target files that are used as inputs for a transformation language program. The target file names are listed in the file <program> and are written, one file name per line, to the file <targetlist>.
ApplyTransformations <functions> <programSDT> Executes a transformation language program. The inputs are a user defined functions file (functions) and a program SDT (generated by ParseTLP). The output files are written according to path names embedded in the program. This program returns 0 if the transformation program executes successfully.
PrettyPrint <style> <targetSDT> <pretty-print> Generates a formatted output file from an SDT. The inputs are a pretty-print style file (style) and a target SDT (targetSDT) generated by ApplyTransformations or ParseTarget. The output is a formatted text string written to the path specified by the third argument (pretty-print).

The sequence of actions to invoke HATS-SML actions is similar for all HATS-SML actions. The sequence is depicted below. The host operating system initiates a HATS-SML process on behalf of the HATS-GUI at the request of the HATS-GUI. The HATS-GUI provides the path names of input and output files (as command line arguments when the request is passed to the host operating system). The HATS-SML process reads the input files, processes, then writes the output files (assuming there are no errors). The HATS-SML process terminates, sending a process return code to the host operating system, which forwards the return code to the HATS-GUI.

Error messages, if they are generated by the HATS-SML process, are written to the standard error stream for the process.

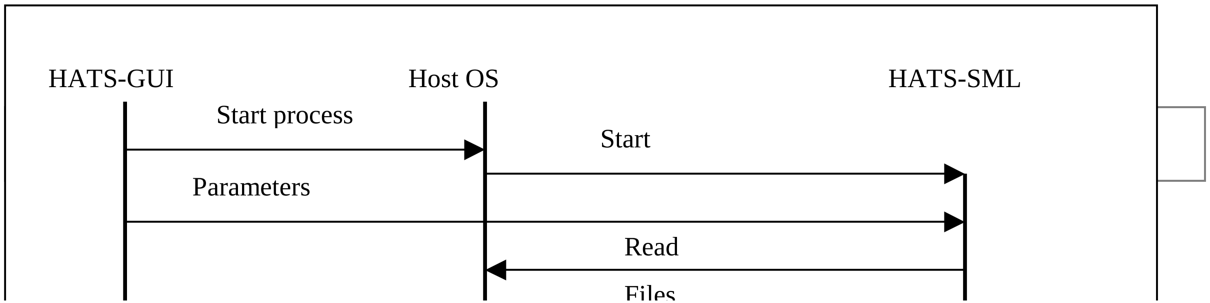


Figure 10: HATS-GUI/HATS-SML Message Sequence Chart

8. Appendix E: SDT Search Criteria

8.1. Overview

The HATS-GUI shall support searching the SDTs for matching sub-trees. This appendix describes searching SDTs for sub-trees. Search in SDTs is done on tokenized strings. The smallest unit of search is a token (as opposed to a character in most text editor searches). Tokens are language specific, and thus SDT searches depend on the lexical specification and grammar of the language of the SDT.

SDT searching is a function performed by the GUI in which a selected SDT is searched for occurrences of a specified "pattern". Some facility should be provided for "first match", "next match", and so on. The location in the SDT where a match occurs should be brought into view and highlighted in the SDT display window. Also, the corresponding text should be highlighted in the text window that is associated with the SDT display window.

8.2. Definitions

8.2.1. Well-formed SDT string of a node

Given a node *N*, the well-formed SDT strings of *N* are defined recursively by the following.

- The node label of *N* is a well-formed SDT string of *N*. (Note that node labels have pointed brackets around them if the node is non-terminal.)
- The left-to-right ordered concatenation of the labels of all of the children nodes of *N* is a well-formed SDT string of *N*.
- The left-to-right ordered concatenation of the well-formed SDT strings of the children nodes of *N* is a well-formed SDT string of *N*.

8.2.2. β derives α

Let β denote a non-terminal in the target grammar. β derives a string α if there is some finite sequence of transformations that convert β to α.

Assume that a target program contains a non-terminal β and the string α has been derived from β. The SDT for the transformed target program will contain β. The string α will be a well-formed SDT string of β. We say that α is derived from β.

8.2.3. Search pattern

A search pattern is a string w such that w is a sub-string of a string α and α is a string derived from some non-terminal β in the target grammar.

8.3. Searching

Given a search pattern ω, the GUI should highlight the smallest derivation string α containing ω. That is, find the smallest string α, and corresponding non-terminal β, such that

1. ω is a sub-string of α, and
2. β derives α.

Note that α is smallest iff there does not exist a β ' and α ' such (1) that α ' is a sub-string of α, (2) ω is a sub-string of α ', and (3) β ' derives α '.

Intuitively, what we are looking for is the smallest well-formed SDT that contains ω as leaf elements. Also, note that ω may contain non-terminal symbols.

8.4. Example

For example, assume we have the following grammar describing simple expressions.

E: <T> * <T> | <T>

T: <F> + <F> | <F>

F: 0 | 1 | … | 9 | (<E>)

Suppose we have the string

(7 * (3 + 6)) * 8

A syntax derivation tree for this expression given this grammar (where all the tokens are single characters) is given below. The nodes are numbered with a subscript to assist in identification for this example.

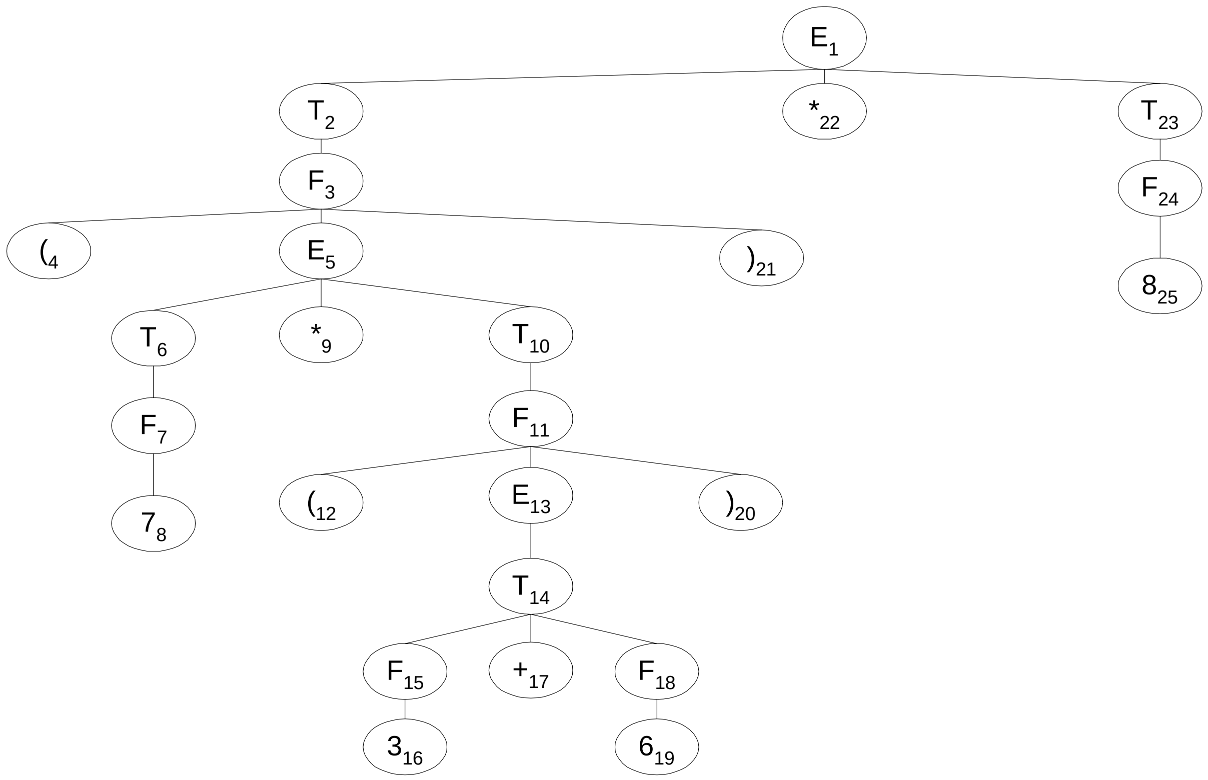


Figure 10: Tree Search Example

The strings “<E>”, “<T> * <<T>”, and “7 * (<E>)” are all well-formed SDT strings from node 5. Each of these strings is derived from “<E>”.

Table 8 describes the results for some example searches.

Table 8: Results from Example Tree Searches

Search Expression	Search Results
3+6	The sub-tree with root T_{14}.
(<E>)	The sub-tree [F_3 [$(_4$, E_5, $)_{21}$]] and the sub-tree [F_{11} [$(_{12}$, E_{13}, $)_{20}$]]
3+<F>	The sub-tree [T_{14} [F_{15} [3_{16}], $+_{17}$, F_{18}]].
+ <T>	No match.

9. Appendix F: Text Search Criteria

Text searching applies to both pretty-printed text and text windows associated with an SDT display. The only difference between the two is the specification of non-terminal symbols appearing in the SDT text windows.

A *search string* is a sequence of characters which includes the special characters $, *, [,], \, and ^. A sub-string in text being searched *matches* a search string if, reading left to right in both strings, there is a character or a set of characters in the sub-string that matches, in order and without skipping any characters, the characters in the search string. Table 9 shows the interpretation of the special characters.

Table 9: Special Symbols in Text Search Strings

Symbol	Interpretation	Example
$	Matches a single character.	“abc def geh” matches the search string “abc def g$h”
*	Matches a set of characters.	“abc def geh” matches the search string “abc*”
^	Start of line.	“abc def geh” matches the search string “^g”
\	Inhibits interpretation of following character	For example, “\$” matches the “$” character. “abc $def geh” matches the search string “abc \$def” Other characters that must be preceded by a “\” are “\”, “*”, “^”, “[“, and “]”.
[…]	Matches one of the characters contained in the brackets.	“abc” matches the search string “a[kbl]c”

Example: Suppose we have the following text.

> "When in the Course of human events, it becomes necessary for one people to
> dissolve the political bands which have connected them with another”

Table 10 gives some example search strings and the results, searching from the start of the string.

Table 10: Results from Example Text Searches

Search Expression	Search Results
an ev	The sub-string “an ev” of “human events”.
ec$me	The sub-string “ecome” of “becomes”.
Course of * bands	The sub-string “Course of human events, it becomes necessary for one people to dissolve the political bands”
[hv]en	The words “when” and “events” have sub-strings matching this expression.
^d	Only matches the d in “dissolve”.

10. Appendix G: Example SDT Display

TARGET PROGRAM

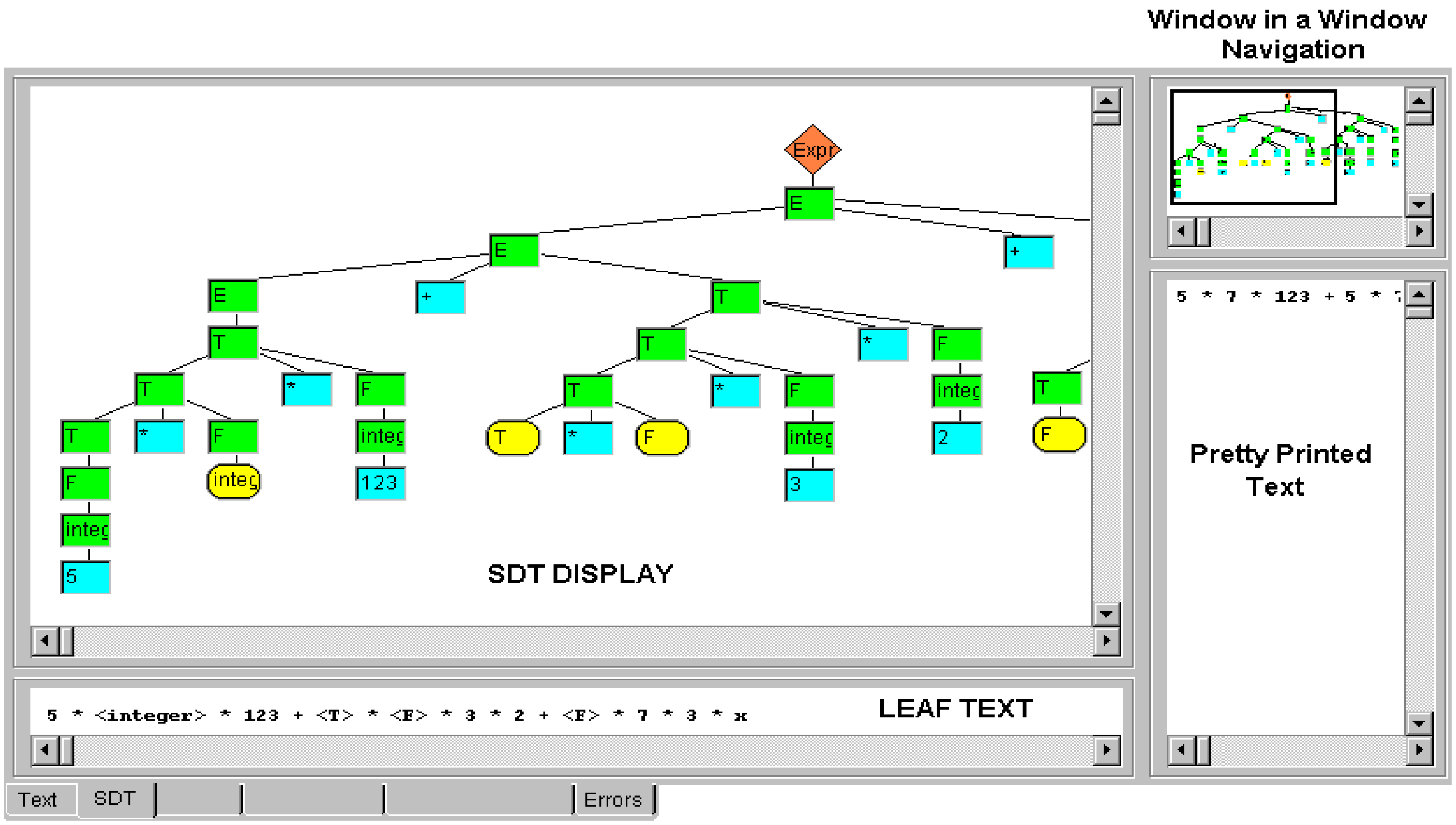
```
5 * 7 * (123 + 3 * ( 2 + x))
```

SDT

```
(Expr(E(E(E(T(T(T(F(integer(5))))(*)(F(integer(7))))(*)(F(integer(123)))))
(+)(T(T(T(T(F(integer(5))))(*)(F(integer(7))))(*)(F(integer(3))))(*)(F(int
eger(2))))))(+)(T(T(T(T(F(integer(5))))(*)(F(integer(7))))(*)(F(integer(3))
))(*)(F(id(ident(x)))))))
```

PRETTY-PRINTED TEXT

```
5 * 7 * 123 + 5 * 7 * 3 * 2 + 5 * 7 * 3 * x
```



```
5 * 7 * 123 + 5 * 7
```

```
5 * <integer> * 123 + <T> * <F> * 3 * 2 + <F> * 7 * 3 * x
```

♦ End of Document