Universal I18n Framework for Office Applications

Technical Overview

Dieter Loeschky, *Staff Engineer*

Shanmugam Senthil, *Member of Technical Staff*

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
1 (800) 786.7638
1.512.434.1511

Copyrights and Trademarks

Contents

Universal I18n Framework for Office Applications

Abstract

Internationalization (i18n) of an application is complete only if any locale support can be added without changing the application binary. Development platforms like win32® and Java® provide i18n APIs to internationalize applications that will run on windows and Java platforms only. A cross-platform office productivity application such as StarSuite® cannot use the i18n APIs provided by the underlying platform because these i18n APIs are inconsistent and insufficient to support desktop applications. The StarSuite I18n framework provides a rich set of i18n APIs to internationalize StarSuite applications using the Universal Network Objects (UNO) component model. This i18n framework is platform-independent and can run on any platform on which StarSuite is supported. The i18n framework is universally accessible to any CORBA or COM components irrespective of their programming language via the UNO remote bridges for CORBA and OLE.

Overview

StarSuite (the asian name of StarOffice®) is a complete, extensible, cross-platform personal productivity application suite. It comes along with several productivity applications such as StarSuite Writer for document authoring, StarSuite Calc for spreadsheets, StarSuite Impress for presentations and StarSuite Draw for image editing. Sun Microsystems has provided most of the source code of this product through open source licensing. The open source project is called OpenOffice.org.

The whole StarSuite architecture is based on a layered approach. The layered approach is one of the important facts to allow the easy porting of the technology to a wide range of different system platforms. There are four well-defined layers, each covering a special area of the functionality.

System Abstraction Layer
This layer encapsulates all system specific APIs and provides a consistent object-oriented API to access system resources in a platform-independent manner.

Infrastructure Layer
A Platform-independent environment for building applications, components and services is provided by this layer. It covers many aspects of an object-oriented API for a complete object-oriented platform including a component model, scripting, compound documents, etc.

Framework Layer
To allow the reuse of implementations in different applications, this layer provides the framework or environment for each application and all shared functionality like common dialogs, file access or configuration management

Application Layer
All OpenOffice.org applications are part of this layer. The way these applications interact is based on the lower layers.

StarOffice 5.2 is a single-byte application where components were built as shared objects and linked to build a huge monolithic application. It supports most of the European locales but not multi-byte locales such as Japanese, Chinese and Korean. The current version StarSuite 6.0 uses Unicode in order to support all European, Asian and BiDi languages. This article will discuss the details around the following issues:

- The layered architecture that provides platform independence.
- Tutorial on UNO component model with an example.
- Migration of StarSuite from single-byte to Unicode.
- Universal Internationalization Framework.
- Tips to help localization developers to customize and extend StarSuite for their market needs.

StarSuite Architecture

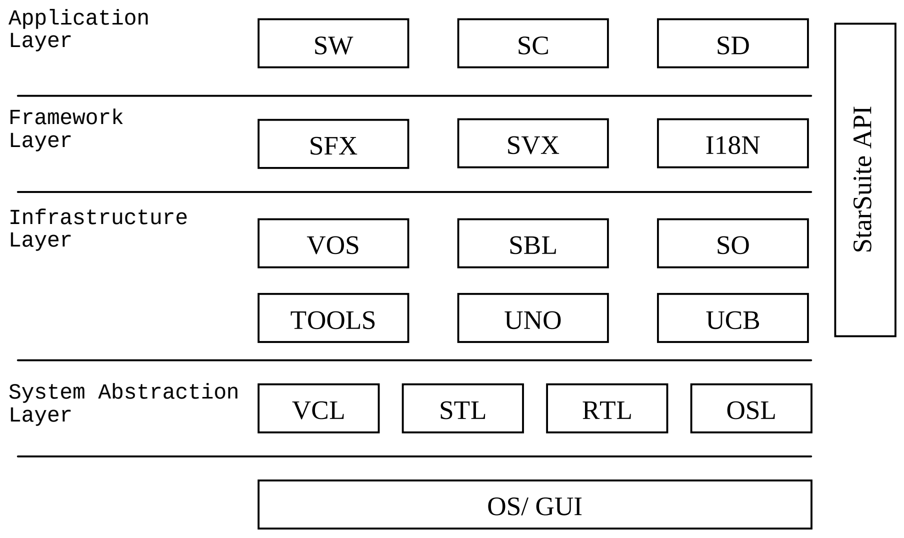

Fig 1: StarSuite Architecture

System Abstraction Layer

The System Abstraction Layer (SAL) abstracts the system level functions required to run StarSuite. The SAL is divided into many modules. All platform-dependent implementation takes place below this layer or is part of optional modules. In order to port StarSuite to another platform, the SAL should be re-implemented with platform-dependent code and recompiled with the rest of the modules. To reduce the porting effort, the set of functionality provided by the SAL is reduced to a minimal set available on every platform. It is not mandatory to port all SAL modules to run StarSuite. Some modules like telephony and speech recognition are optional. StarSuite can still run without these modules with limited functionality. Also, for some systems the layer includes some implementations to emulate some functionality or behavior. For example on systems where no native multi-threading is supported, the layer can support so called *user land* threads. The SAL is made up of the following modules.

Operating System Layer

The Operating System Layer (OSL) encapsulates all the operating system specific functionality for using and accessing system specific resources like files, memory, sockets, pipes, etc. The OSL is a very thin layer with an object-oriented API. In contrast to the upper layer, this object-oriented API is a C-API. This allows easy porting of this layer to different platforms using different implementation languages. For embedded systems or Internet appliances, for example, an assembler language can be used to realize the implementation.

Runtime Library

The Runtime Library (RTL) provides all semi-platform-independent functionality. There is an implementation for string classes provided. Routines for the conversion of strings to different character sets are implemented. The memory management functionality resides in this module.

Standard Template Library

As generic container library, the Standard Template Library (STL) is used. It supplies implementations for lists, queues, stacks, maps, etc.

Visual Class Library

The Visual Class Library (VCL) is one of the core libraries of OpenOffice.org technology. The VCL encapsulates all access to the different underlying GUI systems. The implementation is separated into two major parts. One is completely platform-independent and includes an object-oriented 2D graphics API with metafiles, fonts, raster operations and the whole widget set used by the OpenOffice.org suite. This approach virtually guarantees that all widgets have the same behavior independently of the GUI system used on the different platforms.

The look&feel and the functionality of the widgets are also the same on all platforms.

Because of this design, VCL does not encapsulate the native widgets or controls of the underlying GUI system. The platform-dependent part implements a 2D-graphic drawing canvas used by the platform-independent parts. This canvas redirects all functionality directly to the underlying GUI system. Currently, there is implementation for Win32, X-Windows, OS/2 and Apple Macintosh. The access to the printing functionality, clipboard and drag&drop is also realized inside the VCL.

Infrastructure Layer

Virtual Operating System Layer

To make the usage of system resources like files, threads, sockets, etc. more convenient, the Virtual Operating System layer (VOS) encapsulates all the functionality of the operating system layer into C++ classes. The C++ classes offer easy access to all system resources in an object-oriented way.

Tools Libraries

There are different small libraries forming a set of tool functionality. There is an implementation for structured storages available. Other implementations provide a generic registry, typesafe management and persistence of property data.

Universal Network Objects

In order to eliminate the complexities in integrating several StarSuite components, Universal Network Objects (UNO) defines a communication model among distributed objects. The Universal Network Objects (UNO) is an interface-based object model (like COM or CORBA) that is used to integrate all StarSuite components. The UNO is designed to be as efficient as COM with additional features. The UNO object model defines interfaces with its own Interface Definition Language (IDL). A *UNO component* can implement this interface in any programming language like C++ or Java. The UNO component registers itself into a platform-independent binary repository called the *UNO repository* against a *UNO service name*. The *UNO Runtime Environment* (URE) locates, instantiates and controls the life cycle of a UNO component requested by the UNO service name. The URE uses the UNO repository to locate and instantiate the UNO components. The URE can also locate and communicate with UNO objects instantiated on a remote host. The URE can automatically marshal parameters if the target component is developed using a different language or running on a remote host. If the URE discovers that the target component is developed with the same language and running under the same process, the interaction between them is the same as a function call. If all StarSuite components are run in a single process, there is no overhead in making a function call from one UNO component to another.

UNO components can make use of other components developed with CORBA/COM and vice versa using URE remote bridges for CORBA and OLE. The UNO object model is network-aware so that the different StarSuite components can be running in different machines and interacting seamlessly without any special modification. This feature is the key architecture used in Sun Webtop. Sun Webtop is the next generation of office productivity tools that run by way of a browser and do not need to be installed on local machines. The client that runs on the browser has a bare minimum of components to display the StarSuite front end (like VCL) and the client interacts with the back-end server that runs heavyweight components like StarSuite Writer using UNO. Sun Webtop aims to supply Office Suite applications by using browsers and the Internet.

The key advantage of using UNO to develop all StarSuite components is the flexibility to run the whole StarSuite application on a fat client as well as on a thin browser without modifying the application binary. Even though the UNO was developed for StarSuite, it can also be used independently outside StarSuite. The UNO Development Kit (UDK)[3] provides a set of tools for developing a UNO component, which can be platform-independent. The UDK has the following components:

UNO IDL compiler – Parses IDL files and updates binary repository

cppumaker – Generates C++ header files from IDL

javamaker – Generates Java interface files from IDL

UNO Runtime Environment (URE) - Works on the binary repository to locate a component to provide a specified service, performs parameter data marshalling if the components are developed using different languages. It is bundled with VOS in order to be able to develop platform-independent components.

regcomp, regview – Utilities to register a component in the binary repository and view the contents of binary repository in plain text.

VOS library – The URE comes with a VOS library that abstracts most of operating system calls, hence it makes it a lot easier to develop components using the UNO. The UNO components are available to other CORBA/COM components and vice versa through remote bridges. This feature makes StarSuite components accessible to components with CORBA/COM.

All the components in framework layer and above are implemented as UNO components. This helps to create a very flexible system and also the extension of the system at runtime. The interaction between the components is the same; either StarSuite runs as a monolithic application on the client or runs in Sun Webtop as a client/server application. Developing a UNO object can be illustrated by a simple example.

Example- Develop a UNO component to convert from inches to centimeters and vice versa and write another program that uses this UNO component.

Step #1 – Write the component interface using IDL.

```
module com {module sun { module star { module util
/* name space com.sun.star.util – uniquely identify the interface*/
interface convertMeasurement :com::sun::star::Xinterface
{
   // Name of interface must be inherited from Xinterface
   float inchToMeter(float num);
   float meterToInch (float num);
}
```

The interface name is `convertMeasurement` that is uniquely identified by name space `com.sun.star.util`.

Step #2 – Use `unoidl` command to create/update UNO repository (say `applicat.rdb`) with new interface

Step #3 – Assume that you decided to implement the interface in C++. Use command `cppumaker applicat.rdb org.openoffice.tools.ConvertMeasurement` IconvertMeasurement.hxx

The IconvertMeasurement.hxx looks like:

```
name com {name sun {name star { name util { //C++ namespace
class XconvertMeasurement : public com::sun::star::Xinterface {
   virtual salFloat inchoMeter (salFloat num) =0;
    //salFloat – abstract level of  float;
   virtual salFloat meterToInch (salFloat num) =0;
}
```

Step #4 – Start your implementation by developing a C++ object that extends the abstract class `XconvertMeasurement` and implements the pure virtual functions.

Step #5 – The component is registered against a UNO service name. Add the following two C functions to your C++ component. The code is not complete for more details refer to the UNO guide.

```
void component_writeInfo() {
   registerKey (“MeasurementConverter”, “FooConverterImpl”)
}//  This  implementation  is  uniquely  identified  with  name
fooConverterImpl  in  UNO  repository,  this  function  registers
FooConverterImpl against a UNO service name MeasurementConverter

::com::sun::star::uno::Reference<ConvertMeasurement> foo_builder ( ) {
   return new ConvertMeasurement();
} // This function is C function acts as builder of ConvertMeasurement
object

void * component_getFactory(String implName) {
   if implName==”FooConverterImpl” {
      createSingleFactory(pServiceManager, implName, foo_builder);
```

```
        // let URE know foo_builder is the function that can construct
        object FooConverterImpl
        }
    }
```

Step #6 **–** Compile the components and convert into DLL.

Step #7 **–** Use `regcomp` command to register the DLL into UNO repository. `Regcomp` command calls the `component_writeinfo` function in the DLL with handle to repository. The function registers the service name offered by the `ConvertMeasurement` component against its own unique implementation name.

The development of the component is complete.

Step #8 **–** Develop a program

```
    Xmsf = <create a instance of URE>
    XI = xmsf -> createInstance (“MeasurementConverter”);
    // Load any interface that provides the service MeasurementConverter
    Reference<XmeasurementConverter> xm(XI, UNO_QUERY);
        Xm->inchToMeter (12.2);
```

When the application calls `createInstance` function with a UNO service name, the URE searches the UNO repository and gets the object implementation name. The URE loads the DLL and calls the function `component_getFactory()` to get the builder for this component. Using the builder, the actual object is constructed and returned. The object is constructed using the builder function in order to control the lifecycle of the object. More details can be found in the UNO Developers Guide. The UDK utilities can also be used to add or remove implementation of a service. The calling program need not be recompiled even when the implementation of the service is replaced with another component. Interfacing the components through a component model helps isolate the components and eliminate complicated linking dependency.

Universal Content Broker

The Universal Content Broker (UCB) allows all upper layers to access different kinds of structure content transparently. The UCB consists of a core and several Universal Content Providers that are used to integrate different access protocols. The current implementation provides content for the HTTP protocol, FTP protocol, WebDAV protocol and access to the local file system.

The UCB not only provides access to the content, it also provides the associated meta-information to the content. Actually, both synchronous and asynchronous modes of operation are supported.

Compound Objects

The Compound Object (SO) implementation provides the functionality for building compound documents, in which, for example, a spreadsheet is embedded in a word-processing document.

The current implementation provides a platform-independent implementation of all this functionality for compound documents and for embedding visual controls like multi-media players or different kind of viewers. All the content of a compound document is stored in a structured storage. The current implementation is compatible to the OLE structure storage format. This allows access to OLE compound documents on every platform where OpenOffice.org is available. On the Microsoft Windows platform, the implementation interacts with the OLE services and thus allows tight integration of all OLE-capable applications.

Scripting and Basic Library

The scripting functionality provided with StarSuite is a BASIC dialect featuring an interpreter that parses the source statements and generates meta-instructions. These instructions can be executed directly by the supplied meta-instructions processor or can be made persistent in modules or libraries for later access. All functionality supplied by the upper-level application components is accessed via a scripting interface in the component technology. This helps ensure that new components using the component technology can be fully scriptable without a huge amount of effort.

The scripting interfaces are also implemented as components that enable easy integration of other scripting languages. The interfaces provide functionality like core reflection and introspection similar to the functionality of the Java platform.

Framework Layer

Application Framework Library

The Application Framework Library (SFX) provides an environment for all applications. All functionality shared by all applications and not provided by any other layer is realized here. For the framework, every visual application has to provide a shell and can provide several views. The library provides all basic functionality so only the application-specific features have to be added.

The framework is also responsible for content detection and aggregation. Template management is provided here as well as configuration management. The framework is related to the compound documents in some ways, because of the functionality for merging or switching menu bars and toolbars. In addition, this library makes it possible for applications to be customized.

SVX Library

The SVX library provides shared functionality for all applications not related to a framework. Therefore, part of the library is a complete object-oriented drawing layer which is used by several applications for graphic editing and output. Also a complete 3D-rendering system is part of the drawing functionality.

The common dialogs for font selection, find and replace, transliteration etc. are all part of this library. Also, the complete database connectivity is realized here.

Application Layer

All applications such as the word processing application, spreadsheet application, presentation application, drawing application, charting application, etc. form this layer. All of these applications are realized as shared libraries, which are loaded by the application framework at runtime. The framework provides the environment for these applications and also provides the functionality by way of which these applications can interact.

StarSuite Internationalization

In StarOffice 5.2, applications were monolithic and supported only single-byte locales (western and eastern European). The StarOffice 5.2 i18n framework was based on a class `International` with tables containing data for specific language/country values. Separators (date, decimal, thousands,...), currency symbols et al, were obtained by calls to methods like `GetNumDecimalSep()`. Class `International` also provided methods for character classification, `toUpper()`, `toLower()`, case insensitive `StringCompare()` and so on. The StarOffice 5.2 i18n framework APIs supported single-byte only and linked locale data with the binary. Since locale data was bound with binary data at the time of compilation, it was necessary to recompile the product for any locale data modification.

The decision was made to use Unicode in StarSuite 6.0 in order to enable Western European, Eastern European, Chinese, Japanese and Korean scripts with the new i18n framework based on UNO. Migration was done in a phased manner with the first step being to move the single-byte code base into Unicode, then continuing to use the single-byte i18n framework with some special wrapper classes and finally implementing the new Unicode-based i18n framework to replace the old one.

Transition from Single Byte Character to Unicode

StarOffice 5.2 was built on a single-byte character model and could not support any multi-byte characters. Fortunately, character representation was done using C++ String class and was platform-independent. Another advantage is that over 80% of the source code is system-independent. Hence, about 7 million lines of C++ code was transitioned to Unicode in less than four weeks using the following approach:

Renaming of the old class String
Renaming of class `String` to class `ByteString` and `#define String ByteString`.

Create a new class UniString
Implementation of a class `UniString` including conversions from and to class `ByteString` for several character encodings. The class `UniString` was designed to have the same methods and functionality as the class `ByteString`. Stream methods were added to write `ByteString` from `UniString` and to read `ByteString` into `UniString`.

Use macro switch String to UniString
By `#define String UniString` the entire StarSuite code uses UniString. Since UniString class has all the methods of ByteString, the string class is replaced seamlessly. All file reading/writing code had to be changed to explicitly read and write `ByteString`. Stream operators << and >> were implemented for `ByteString` but had not been implemented for `UniString`, in order to force every developer to use the right methods.

Miscellaneous
The Resource system was enabled to read UTF-8 strings and display Unicode. Additional Unicode file and clipboard I/O had to be implemented.

The macro conversion does not apply to all the modules. For example a token parser's (for example a RTF or HTML) tokens must not change to Unicode because they always contain only ASCII characters. Only specific content or names must change to Unicode.

StarSuite introduced new code converters to convert from Unicode to the legacy code set and vice versa in order to load and save data in files like configuration files, database files or other file formats, such as a user-defined binary format or a third party format (text, RTF, WinWord, etc.).

Old I18n Framework under Unicode

The old i18n framework was based on tables that supported only single-byte locales. It is necessary to have the new i18n framework based on the UNO component model and able to support Unicode. Until the new i18n framework was ready, special wrapper classes were used to interface with older i18n classes. The special wrapper classes made it possible to move the source base to OpenOffice.org with fully-enabled Unicode and allowed third party vendors to provide additional locale support to StarSuite.

After successful completion of the first phase of using the old i18n framework with the Unicode framework, the next phase of implementation of the full-featured, comprehensive new i18n framework commenced. The following chapters discuss the requirements, design architecture and flexibility of the framework.

Unicode-based I18n Framework Requirements

The requirements for this new i18n framework are based on extensibility, pluggablity and simplicity (easy to use API and easy to modify locale data).

I18n Framework as UNO Component

StarSuite follows a layered architecture that allows the development of other custom desktop applications. Even though the modules in the framework layer like SVX are developed using C++, they are all UNO components and can potentially be used by any component developed in any language. Hence, the I18n framework should be developed as one or more UNO components, in such a way that the custom application should be able to access the i18n framework APIs too.

Unicode 3.0 Support

StarSuite represents characters using Unicode in order to process multi-lingual documents as well as process documents of any language irrespective of platform locale. The I18n framework should provide correct character classification mechanism to support the latest Unicode 3.0. Most of Unicode 2.0 implementations tend to assume that characters can be represented by two bytes, but that does not hold true any more. Each character may be represented using more than one code point. The API design of the character classification should handle the situation of multiple code points per character case correctly.

Encapsulation

All of the locale-sensitive behavior should be encapsulated in the i18n framework APIs. For example, the user may want to search a document for a given string. The search may include an option to perform case-insensitive search. Case-insensitive searches make no sense for Japanese documents. For a Japanese document, searching without distinguishing katakana-hiragana character differences makes more sense. Such options are locale-sensitive and hence the i18n framework APIs should encapsulate the locale-sensitive behavior to support additional locales in StarSuite without changing the binary. StarSuite has near-future plans to support up to 76 locales and it is becoming impossible to change the binary to support each and every locale.

Pluggable Locale Support

Since StarSuite supports many locales, the locale support is prone to error. The i18n framework should make the addition or modification of locale behavior easier. If a customer finds a bug in the behavior of a specific locale, the i18n framework should allow the removal of the buggy module and replace it with a new one without affecting the StarSuite binary. This is made easier by developing the framework using UNO. The UNO tools can modify the UNO repository to change the behavior of the locale.

Collation

A user can choose more than one collation algorithm for sorting data. This means that the collation API must provide interface to query the collation algorithms applicable for the locale and select one of them to sort. Collation options can be locale-sensitive. For example in Japanese locale, the font names can be sorted ignoring the difference between half-width and full width character. These options are very locale-specific and cannot be specified in the application. The collation API must provide some abstract options that map into locale-sensitive options.

Number Formatter

Much work had to be done on the number formatter since it makes extensive use of locale data and number format codes provided by the i18n framework. New keyword symbols, parsing methods and string output methods had to be developed to enable it to make use of the new calendar API and to use different calendars in the same format code. A special goal regarding calendar formats was not only the ability to behave the same way as, for example, Japanese Microsoft Excel does but also to extend the capabilities to be able to display any given combination of calendar systems for a given locale as long as the locale data provides information about them.

Calendar

The Calendar API provides an interface for performing date arithmetic based on various calendars. Even though most of the locales by default support the Gregorian calendar only, many locales support the Gregorian as well as other calendars. For example, the Japanese locale supports the Emperor era calendar as well as the Gregorian calendar. Hence, the calendar API should have the interface to query the available calendars for any locale.

Break Iterator

The break iterator provides APIs to iterate a string by character, word, line and sentence. Iterating character by character is essential for two reasons:

Cursor movement – UniString class has an array of code points. Since a character can take more than one code point, cursor movement cannot be done by incrementing/decrementing the index.

Complex text layout languages like Arabic, Thai, Indic scripts – In these scripts, multiple characters combine to form a display cell. Cursor movement should be calculated to jump a display cell instead of single character.

Line breaking should be highly configurable in desktop publishing applications. The line-breaking algorithm should be able to find a line break with or without hyphenator. The line breaking API should support for some special characters that are forbidden to at the end of the line or beginning of the line. The character/word/line iteration algorithms are locale-sensitive and should be pluggable.

StarSuite I18n Framework Architecture

I18n framework consists of several components namely *Locale Data*, *Character Classification*, *Collation*, *Calendar*, *Break Iterator and Transliteration* and *Search*. Each component of the framework is implemented as a UNO component. The following figure shows the interaction between various components. The following key architecture decisions have enabled this framework to meet all the requirements.

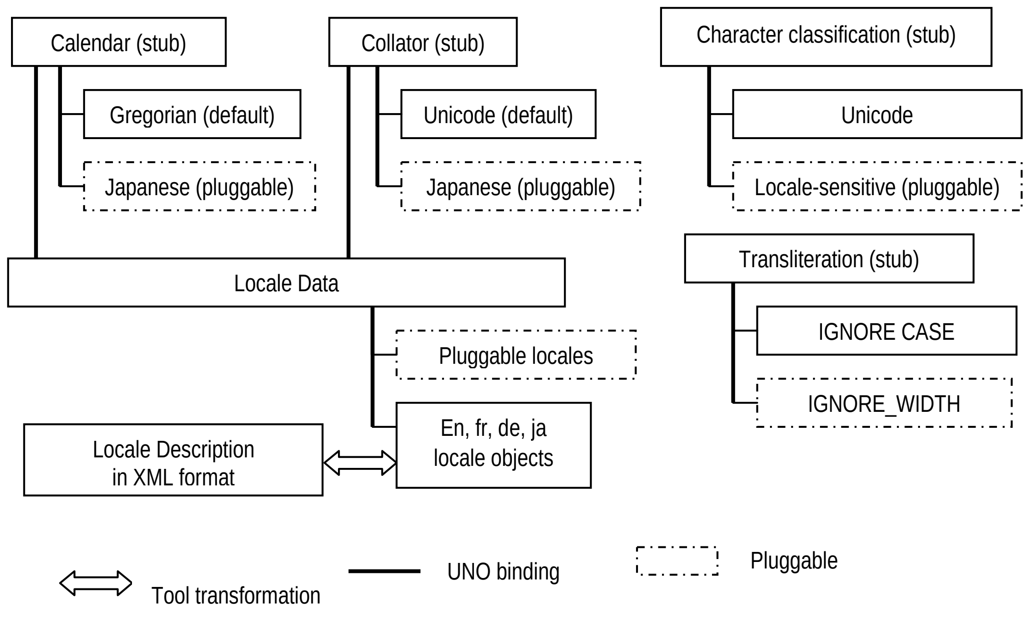

Fig 2: StarSuite I18N Framework Architecture

Standard naming convention for UNO Service names

StarSuite defines the UNO service naming convention for each locale-sensitive component. For example the service name convention for break iterator object is `com.sun.star.i18n.impl.<locale_name>.break iterator`. Each locale-sensitive component should be registered under the UNO service naming convention. If StarSuite is run in the Thai locale, it looks for the UNO service `com.sun.staroffice.i18n.imp.th_TH.break iterator`. Developers can register their Thai break iterator module against the service name and have it automatically loaded by StarSuite. By following the naming convention, any locale-sensitive component can be plugged dynamically into the StarSuite binary repository. Hence, without recompiling StarSuite, locale behavior can be enhanced as well as new locales plugged in.

Fallback mechanism using stubs
Since each component in the framework is a UNO component, enabling a locale requires having all locale-sensitive components (character classification, collator) defined for the locale. It will be very difficult to develop all the components for each locale as most of the locales have overlapping behavior. For example collation in the en_US locale and en_CA locale are the same. This would result in a lot of duplicated effort. The worst case would be if StarSuite is run in a locale that is not supported and it will not run because of the unavailability of the locale-sensitive modules. In order to overcome this limitation, each component has a stub object as marked in the architecture diagram to provide this fallback functionality. Stub modules provide a UNO service that is guaranteed to be available. All StarSuite modules request the UNO service for the stub module and pass the locale information. Stub modules attempt to locate the locale-sensitive module using the UNO service naming convention. The fallback order is shown in the following table.

Order	Locale
1	<language>_<country>.<varient>
2	<language>_<country>
3	<language>
4	Default component – the name of the component is directly coded in the stub

In case of the break iterator, for the fr_CA locale, it attempts to locate a break iterator service for fr_CA. If it is unavailable, it attempts to locate a break iterator for fr. If it is not available, it falls back on the default break iterator. This fallback mechanism makes locale development easier. For example, the Spanish language has about 35 locales but they all share one collator.

The naming convention and fallback mechanism simplifies locale development and maintainence and hence achieves truly pluggable locales. Since the stub module is always available and the stub module always falls back on known default implementation, StarSuite does not fail to run in unsupported locales.

Character representation
StarSuite represents strings with an array if int16 (16 bit integer). All of the i18n framework APIs are designed to avoid the assumption that one code point, i.e. int16,

is a character. The i18n framework APIs do not accept int16 as a character parameter. Instead a character is represented as a separate data type; it is represented as a string with starting index. This representation allows multiple code points per character and the i18n framework APIs can process the characters accordingly. In the case of a Unicode specification upgrade, the i18n framework code has to be changed but the application layer requires no changes.

Character Classification

This module provides interface for providing information about a character, like isAlpha(), as well as converting characters into different categories, like toUpper(), toLower() etc. These APIs accept UniString and startIndex as parameters to represent a character like `isAlpha(UniString str, int startIdx)`. The stub module attempts to locate locale-sensitive UNO objects and uses the Unicode character classification object as a fallback object. In StarSuite, character classification modules are extensively used in parsing strings into tokens. In order to avoid misuse of these functionalities, parsing functions are also added as part of the interface. The parsing functions can tell basic tokens such as name or number from a string.

Locale Data

This module provides language, country and cultural specific data, like currency symbol, associated with any locale. It is cumbersome to write a UNO object for each locale, i.e. writing C++ modules for locale data is not the preferred way of providing locale data. In order to simplify the locale data UNO object development, several tools are provided. The locale data can be defined in XML format. The XML data file is then passed to an XML parser to generate C++ code which in turn is compiled into a UNO object. It is not mandatory to define locale data in an XML file but it is the recommended way of developing new locales in StarSuite.

The Locale Data module also provides additional information specifically required for modules like StarSuite Writer and StarSuite Calc. One example could be that StarSuite Calc allows number formatting in different ways. The number can be formatted using different format codes. StarSuite Calc shows a list of formats the user can use for any given locale and it retrieves the format code from this module. Hence, the special format code for the Japanese calendar is shown only to Japanese users and not to English users.

Break Iterator

StarSuite does not make an assumption that one character is one code point and that means that edit control components cannot just increment the index by one to calculate the next cursor position. This module provides character or word iteration APIs to calculate the next/previous cursor position. Word break algorithms for locales such as Japanese require a dictionary lookup to identify a meaningful word. Line break for edit components of StarSuite can be configured with any one or more of the following options:

- Hyphenate.
- Forbid some set of characters to begin or end the line.
- Allow some punctuation marks to render even outside the margin.

Since StarSuite uses Unicode, it has capabilities to handle multi-lingual documents. If a Japanese document is viewed with StarSuite Writer in an English locale, it does not make any sense to apply English dictionary rules to the Japanese document. In order to associate correct dictionary rules, this module also provides APIs to auto-detect the script of the string. StarSuite currently has a break iterator that works on Unicode as a default object loaded by stub. It also comes with a dictionary based break iterator for Japanese, Chinese locales also.

Transliteration

Transliteration maps one character into another without understanding the sematics of word or sentence, hence it should not be confused with translation. Character level transliteration can be applied to some scripts which have multiple writing systems like Japanese or Chinese. A typical example of transliteration would be to convert a half width character into a full width character in Japanese. In StarSuite, users of edit controls can select a block of text and convert it into another text using this module. All the transliteration modules are UNO objects and are registered in the UNO repository using a unique UNO service name. L10n developers list the service names of transliteration modules applicable to the locale in the Locale Data XML file. The transliteration stub module loads the list of available transliteraiton modules from locale data and passes to edit control. Thus, users get to see the transliteration module applicable to the default locale only. The transliteration module IGNORE_CASE implements the Unicode case folding algorithm, which is different from `toUpper()` or `toLower()` as these are locale-sensitive. Transliteration can also be used as an option in the find and replace operation. In applications like StarSuite Writer, users can perform a search with one or more options such as case insensitive search, ignore Katakana/Hiragana and so on. Each of these options can be mapped into a transliteration module. The search algorithm transliterates the string first and performs the search for an absolute match. The transliteration API returns the transliterated string as well as the mapping table between source character index and target index. If the search algorithm finds a match in the transliterated string, it uses the mapping table to find the source string that matches. The transliteration stub API allows the cascading of more than one transliteration module.

Collation

Collation is used by many modules in StarSuite. Since collation algorithms are locale-sensitive, each collation algorithm is registered under unique UNO service name. The collation stub needs to be more sophisticated than just loading localized modules because of its wide-spread use by various components. Collation usage can be classified in two broad categories:

User-Invoked
The user selects data from a spreadsheet and invokes sorting through the GUI. The users of the sorting dialog may select the sorting algorithm and the sort options like case insensitive. The sorting algorithms available as options to end-user are locale-sensitive, i.e. the German telephone number sorting algorithm is not applicable to the Chinese user. The UNO service name of all collation algorithms applicable to a given locale is listed in XML locale data. The collation stub provides a separate API (listCollatorAlgorithms(locale)) which retrieves the list of collator algorithms applicable to a locale from locale data.

Application-Invoked
StarSuite invokes collation modules for sorting font names, file names, auto completion, auto correction and so on. Sorting the different data items need not be strict. For example, the font names can be sorted insensitive to case in the en_US locale but it can be different for a Japanese locale. These collation options can be mapped into a transliteration module. Since collation options for sorting is locale-sensitive, they are listed in the locale data under an abstract option. StarSuite modules pass the abstract option to the collator stub and the collator stub looks in locale data to find out the actual transliteration modules defined for the abstract name and applies them before invoking the actual collator algorithm. For example, StarSuite defines an abstract option called NAME_SORT which is used for sorting font names and file names. The abstract option is mapped into CASE_IGNORE for the English locale and IGNORE_WIDTH for the Japanese locale.

Find / Replace

StarSuite currently supports three types of search algorithm, namely the absolute search, regular expression search and approximate search. The absolute search matches the string in a document. The absolute search provides options which can be mapped into transliteration modules. The list of search options applicable for a locale is listed in XML locale data.

Customizing Locale Specific Modules

All the components in the StarSuite I18n framework are UNO components. Hence the i18n components can be easily removed or added into UNO repository without modifying the StarSuite binary. Following the naming conventions for UNO service, names ensures that the locale-sensitive components are picked up correctly. The new i18n framework has been architected to make locale management simple. This i18n framework allows other vendors to add new locales to support their local market needs as well as enhance existing locale behavior with enhanced algorithms. For example, vendors can add a new Greek locale to meet the needs of Greek language users or add a better dictionary based word break algorithms for a Japanese locale, add a new calendar to the Chinese locale and so on without modifying the StarSuite binary.

Adding a New Locale

The effort needed to create new locale support varies with the complexity of the locale. Regional L10N developers need to develop new locales to enable StarSuite in the locales not supported by StarSuite. The following steps highlight the steps involved in creating a locale.

Step #1: Locale data, Calendar, Collator, Break Iterator, Character classification are mandatory to create a locale, transliteration modules are optional.

Step #2: Find out the number of calendars applicable to this locale. If it is just Gregorian, it is already available in StarSuite. If any other calendar is necessary, develop the module and register against unique UNO service name. Make note of UNO service names of calendars.

Step #3: Check if you need any special break iterator module and character classification module. StarSuite by default provides Unicode-based character classification and break iterator. If that is not enough, develop them and register them against the UNO service naming convention suggested by StarSuite.

Step #4: Develop locale data in XML format. Locale data XML file requires information about locale like currency symbol, format codes to be used etc. In addition to this information, a list of collators to be used and a list of calendars applicable for this locale should also be mentioned in the XML file

Step #5: Run the XML parser to generate C++ files

Step #6: Provide C functions `component_writeinfo()` and `component_getfactory()` to register the locale data object, break iterator object and character classification object.

Step #7: Compile the C++ files to generate Shared Objects or DLLs

Step #8: Identify the binary repository of StarSuite (usually it is named `applicat.rdb`). Run the `regcomp` tool to register the DLL with `applicat.rdb`.

Modifying Locale Behavior

The following steps are useful for l10n developers who enhance StarSuite with value added modules for existing locales. The example shows how to add the new Japanese Emperor calendar in addition to an existing Gregorian calendar.

The interface for the calendar is same as the interface implemented for the calendar stub.

Use the `cppumaker` utility to generate an interface header file from the UNO repository of StarSuite (usually it is called applicat.rdb).

Implement the Japanese calendar overriding the virtual methods of the interface.

Write additional C functions `componenet_info()` and `component_getfactory()` to register the component against a unique UNO service name.

Calendar stub reads Locale Data to find available calendar for this locale and this needs to be changed. Obtain the locale data in XML format, add the new calendar entry listing the UNO service name of the Japanese calendar.

Use the XML parser to convert the XML file into a C++ file.

Compile the C++ file and generate the shared object for this locale.

I18n Framework is Universal and Extensible

The StarSuite I18n framework has been developed with desktop applications like StarSuite Writer and Calc in mind. This Unicode-based StarSuite i18n framework can be considered universal in that it can be used for any application for the following reasons:

Platform-independent because it can be ported to any platform without any modification.

C++ applications can use this framework.

Since the components are UNO-based, an application that is developed in any programming language can make use of these APIs as long as the programming language binding is supported by UNO.

Since URE comes with remote bridges for CORBA and OLE, the CORBA and COM components can access this framework using the bridge.

No assumption made about character or string representation. The framework works with various APIs of string.

This framework is extensible. L10n developers can add new locales or enhance existing locale components to meet market requirements of StarSuite. This framework uses minimal features of UNO and can be migrated with any other interfaced component model like CORBA. The i18n framework uses String APIs from StarSuite and does not make any assumption about the length of character or string representation. Hence it can be easily ported to any application that may use strings and characters different than StarSuite. Thus the framework is extensible for future desktop and office application development and ideally suited for cross platform C++ applications.

The source code of this framework is planned to be open-sourced without giving out third party modules. Openoffice.org L10n developers can add new locales or enhance existing locale behavior.

Conclusion

StarSuite is not just an office application suite, it is a completely object-oriented platform for developing any cross-platform desktop application (like mail client, scheduler). The new StarSuite i18n framework is Unicode based and offers a rich set of APIs and functionality. These APIs meet the requirements of existing applications and are also generic enough to be used for any application developed on this platform. It provides a rich set of APIs to encapsulate all localization behavior inside the framework to serve i18n requirements of office suite products. The i18n framework allows localization developers to add new locales or enhance existing locale behavior to meet regional market requirements without modifying the StarSuite binary. The framework API is accessible to CORBA/UNO components, which makes the StarSuite i18n framework universal.

Reference

[1] Sun Microsystems announces availability of StarOffice[tm] source code on OpenOffice.org
http://www.sun.com/smi/Press/sunflash/2000-10/sunflash.20001016.4.html
http://www.sun.com/staroffice/openoffice/

[2] OpenOffice home page
http://www.openoffice.org

[3] OpenOffice localization and internationalization project
http://l10n.openoffice.org/

[4] UNO home page
http://udk.openoffice.org

[5] Introduction to UNO
http://udk.openoffice.org/common/man/concept/unointro.html

[6] I18n API
http://api.openoffice.org/source/browse/api/offapi/com/sun/star/i18n/

Contacts

Dieter Loeschky
Dieter.Loeschky@germany.sun.com

Shanmugam Senthil
Shanmugam.Senthil@eng.sun.com

Acknowledgements

The i18n framework architecture, design and APIs are team work done by a number of developers. The authors of this presentation would like to thank Ralf Hofmann, Thomas Hosemann, Juergen Pingel and Eike Rathke for their invaluable suggestions in architecting and designing new APIs for this i18n framework.