The *Content Manager*: A tool to develop multilingual and multi-preference web sites

WPI-CS-TR-01-11

Alok Mehta, Piedad Rodriguez, Ricardo Rodriguez
{amehta,prodriguez,rrodriguez}@afs-link.com
American Financial Systems, Inc.
9 Riverside Office Park
Weston, MA USA

George T. Heineman
heineman@cs.wpi.edu
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA USA

1. ABSTRACT

In this experience paper we outline our experience in developing a multilingual and multiple preference web site. In particular we describe *Content Manager*, a tool we developed to support the implementation of a business to consumer (B2C) international web site. We also describe the business requirements and challenges that we encountered. There are many commercial tools for managing a web site's content, but these tools are unable to manage the complexity of diverse languages taxation frameworks, and cultural systems. We found that by separating the content from the source code (as supposed to embedding text within the web page), we were able to focus on the development of the web site instead of worrying about the differences between target countries. *Content Manager* allowed us to maintain this separation and eased the development process.

2. INTRODUCTION

The Internet makes it possible to reach potential customers from around the world by breaking down communication barriers. This opportunity creates immense challenges for e-business developers who want to benefit form this situation. We need tools to help manage not just the content and structure of a web site but also enable a web site to be developed, deployed, and maintained for multiple diverse audiences with different languages and culture.

We describe our experience in building a web site for selling term life insurance products over the Internet to an initial audience located in seven countries and three languages. We designed and developed a tool to manage the content of this web site. In Section 2 we describe the main site, its mission, requirements and challenges. In Section 3 we describe the features of the *Content Manager*. In Section 4, we present the architecture of the *Content Manager*. In section 5, we discuss the related work and close with concluding remarks.

2.1. Context of the project

Our main business objective was to create a new distribution channel for term life insurance products in Latin America that would be more effective than the traditional agent structure. We expected to achieve this objective by developing a web site with the following characteristics:

- **Highly educational**: the website is directed to users that are often not familiar with life insurance or have misconceptions about it. Since it is very rich in content, the site should be well structured, be comfortable to read, and use an appropriate tone and wording to reach the target audience.

- **Transactional**: the user should be able to perform a needs analysis, get online quotes, have comparison criteria, apply for a product, complete underwriting procedures, and buy a policy.

- **Trustworthy**: to sell life insurance policies on-line, the web site should engender trust by showing affiliation with the customer. This is achieved by providing accurate and clear content, full disclosure on the terms and conditions of the products that are offered, simple transactional processes, and security.

Aside from the design of the transactional processes, which are the core of the business and support the revenue model, a great effort was put in the development of the content, which is critical to create an educational site, to show reliability and affiliation, and to reach potential customers.

Figure 1.0 describes the process flow of the main site which is divided into a secured and unsecured portion. The *Content Manager* manages the content for both sections. The unsecured portion of the site contains informational pages regarding the business. The site's secured portion is managed under the umbrella of SSL (128-bit encryption). The heart of the secured site is the automated underwriting subsystem which queries potential consumers about their health history in order to approve (or disapprove) an online coverage of life insurance. The main site also contains an *identity verification* component that verifies consumer information and the *ePayment* component that accepts online payments when the consumer qualifies for automated coverage. In this paper, we do not discuss the life insurance domain, identity verification, or the ePayment components We are concerned with deploying

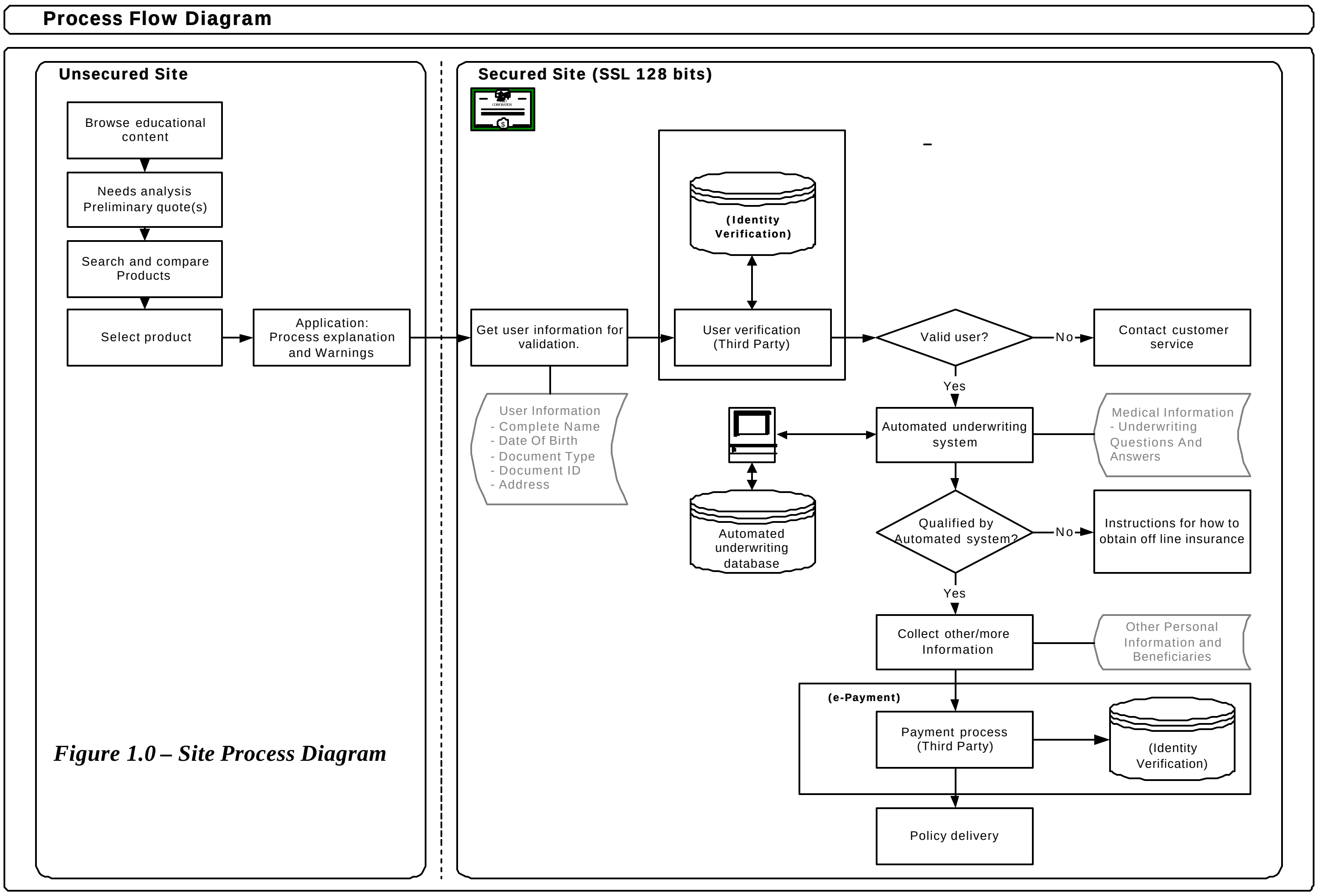

Process Flow Diagram
Unsecured Site
Browse educational content
Needs analysis Preliminary quote(s)
Search and compare Products
Select product
Application: Process explanation and Warnings
Secured Site (SSL 128 bits)
(Identity Verification)
Get user information for validation.
User verification (Third Party)
Valid user?
No
Contact customer service
Yes
User Information - Complete Name - Date Of Birth - Document Type - Document ID - Address
Automated underwriting system
Medical Information - Underwriting Questions And Answers
Automated underwriting database
Qualified by Automated system?
No
Instructions for how to obtain off line insurance
Yes
Collect other/more Information
Other Personal Information and Beneficiaries
(e-Payment)
Payment process (Third Party)
(Identity Verification)
Policy delivery

Figure 1.0 – Site Process Diagram

and maintaining .a web site that will support multiple languages and be easily extensible.

2.2. Requirements and challenges

We faced numerous design challenges because of the international nature of the project and specific complexities of the insurance industry. The following key points formed the basis for our design:

- We needed to capture each country's specific legislation and business environment concerning the product's application and purchase process. The site should be able to adapt to these particularities without losing commonalities among countries.

- While many of the countries targeted by the site share Spanish as a common language, there are many regional linguistic differences. The site must include local dialects and characteristics to reach each country's audience.
- Cultural non-language differences must be considered in the design of transactional processes. Some examples include address and telephone format, currency and currency formats, hour and date formats, and height and weight units.
- The site should be able to adapt to other user preferences. In this particular case, users should have the option to select between a summarized version and a detailed version of the educational content.

- The site should enable the purchase of life insurance coverage in either the local currency or in US Dollars; therefore the system should handle multiple currencies and exchange rates.

- Assumptions for economic variables such as interest rate, inflation rate, salary increase rate, and exchange rate vary form country to country. Default values should be considered carefully and the user should be able to modify them.

Various phases of the design and implementation of the project were developed by a physically disperse team, which created a communication and coordination challenge. Team members were multilingual, had different professional and cultural backgrounds, and worked in different countries. The technical infrastructure of the project had to ensure that proper communication channels among the countries existed, and tools to share and transfer all sorts of data were available.

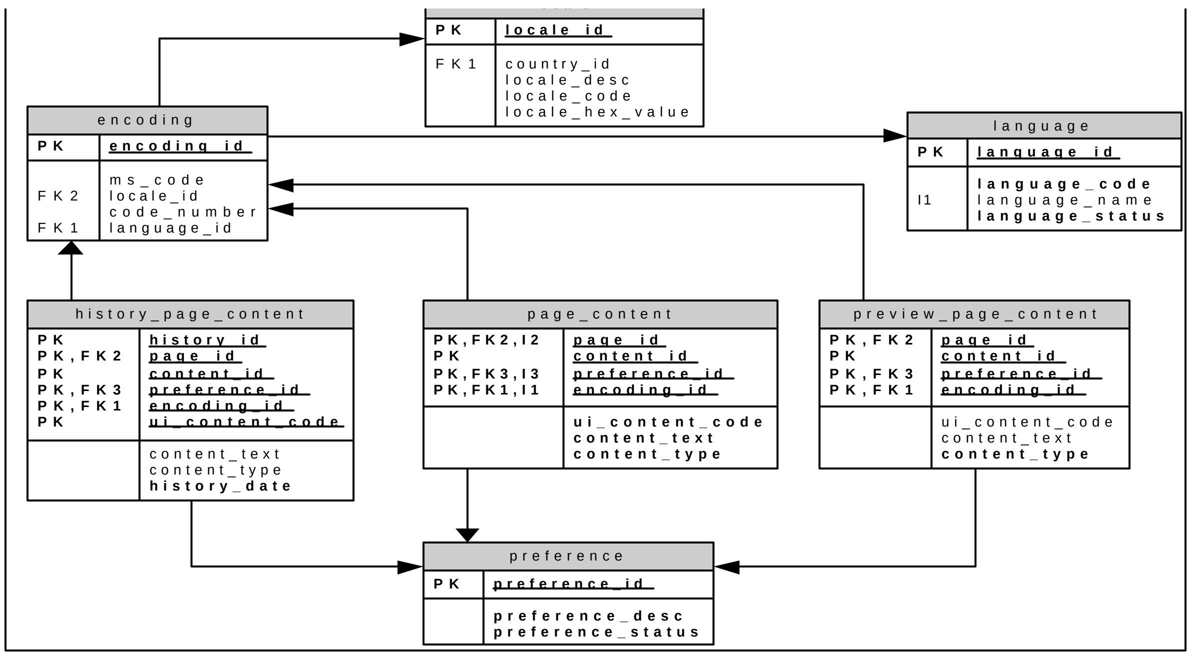


Figure 2.0 – The Data Model

By operating in multiple countries, we had to integrate with local providers for specific transactions such as the identity verification of the users and credit card payment authorization and processing.

3. The *Content Manager*

The tool developed to address most of these challenges is called *Content Manager*. As its name suggests, *Content Manager* makes it possible to administer the entire site's content independent from the source code. Content developers can input, edit, organize, format, translate and customize every single word that appears in the web site.

3.1. English Content Manager:

We selected the English *Content Manager* to be the primary interface for creating the site's content. In this module, all the pages of the site are listed, and when one of the pages is selected its content is displayed. The content is divided in small subsections identified by a unique ID that is referenced from the main code. In this way, one subsection can have the page's title, another can have a sub-title, another the first paragraph, and so on; from the code each content ID is formatted and placed in the right place.

The user can modify sections of the content in each page and, after saving the changes, use the Preview option to verify how the screen will look. Then, using the *Publish* tool, changes are made permanent in the database. In this way, the content can be easily modified without the programmer intervention.

The appearance of the content can also be defined using HTML tags, and basic navigation can be implemented though using the LINK instruction, which converts a given text into a link to a specific page, using a particular style from the site's CSS. The English *Content Manager* also allows the content developer to identify searchable content accessed using the keyword search engine from the site.

When new pages are added to the site, they can be included in the *Content Manager* through the Add Page option, and in a similar way pages can be removed from the site with the Delete Page option. Since content can be added and deleted for each of the pages, the *Content Manager* keeps a history of every page, allowing previous versions of the selected page to be viewed and restored.

User customization can be achieved through the Preferences option, where any amount of preferences can be added to the site. In this particular project, this option was used to target users who prefer short and summarized contents as well as users who prefer long and detailed contents. To achieve this customization, a detailed version of the content was entered in preference #1 and a summarized version in preference #2. The user can then select according to his/her preference, the detailed or the summarized version from the main site. In future developments this option will be vital to target specific needs and tastes of the consumers.

3.2. Language Manager

The Language Manager defines the different languages used within the site. Through this module we addressed the issue of customizing the language for the different countries by considering the local accent of each country as a separate language: for example, Mexican-Spanish was the language defined for Mexico while Argentine-Spanish was used in Argentina.

3.3. Content Translator

The Content Translator is the module used to translate the content to different languages. Its layout is very similar to that of the English *Content Manager*, except that when a page is selected, the English content for that page appears un-editable on the left side with a corresponding editable field to the right where the translation can be input. It also has an option to Preview the changes.

The Content Translator can simultaneously manage multiple languages using the Language control, where each of the languages defined in the Language Manager is displayed. When a language is selected, the English content is displayed with the corresponding fields for the language's translation. This multi-lingual capability made possible and easy to achieve the different "flavors" of Spanish required for each country.

The different preferences can also be translated in a similar way to the multiple languages. In this way a Mexican consumer can view a detailed explanation of life insurance written using a Mexican accent, (Language: Mexican-Spanish; Preference #1) while an Argentine consumer can view the summary of the same page with an Argentine accent (Language: Argentine-Spanish; Preference #2). The combination of these two features allows enormous possibilities for customization.

3.4. Control Manager, Error Manager, and Help Manager

Since the site is not only textual but also transactional, controls, help text, and errors are also handled in the *Content Manager*. The Control Manager allows the creation, edition, and elimination of controls, and through the Control Translator, the elements of each control can be tailored for each language. This functionality allowed us to solve many of the cultural differences among countries. For example, the elements for the "Time" control for Mexico were defined as 1:00PM, 2:00PM, and so forth, while the same elements for Argentina were defined as 13:00, 14:00, etc, with a simple switch of language. This was extended to many other controls, which permitted us to tailor the transactional process as easy as the rest of the content without having to create different screens and screen flow for each country.

Since errors and help text are also part of the content, the Error Manager and Site Help Manager modules enables developers to create and modify the help tips to be displayed when a user clicks on the help icon or makes a mistake while conducting an operation in the site. Each error and help text has a unique id that is used in the code to reference the proper text. Errors, as well as help text, can also be easily translated in a similar way as the general content, through the Error Translator and Help Translator respectively.

3.5. Site Navigation Manager

In addition to managing the content and making easier the customization of the transactional process, the *Content Manager* centralizes the images used in the site for navigation and aesthetical purposes through the Site Navigation Manager. In this module, the navigation bars and graphical menus of the site are constructed by including the images that will compose them, and identifying the page that will be displayed when the user clicks on each of the images.

This module also includes the Image Manager, which organizes all images included in the site by categorizing them in common images such as pictures or icons, and language specific images such as graphical titles.

4. SYSTEM FRAMEWORK

Our mission was to design an easily extensible, web-enabled, insurance purchasing system with multi-language support.

4.1. Technology Overview

We developed this site using Microsoft™ based technology. In essence, we used SQL Server 7.0 database on Windows 2000 Server platform using IIS (Internet Information Server) and ASP (Active Server Pages) and Visual Studio 6.0. While the data model can be supported by any relational database vendor, we have used MS SQL 7.0. Our tiered architecture is developed in various programming languages (Visual Basic, C, C++, Java, ASP) using component based software solutions. The majority of the system is composed of ASP code and VB DLLs running in the IIS environment with a SQL Server 7.0 database. MTS supports transactions across multiple databases.

4.2. The Data Model

Because of space reasons, Figure 2.0 shows a partial data model of the main site. The *Content Manager* is programmed to populate/manage this data model. To review the features of the Content Manager, see Section 3.0. The most important table in the entire data model is the *page_content* table. It contains the text that is to be displayed on a given web page in a given language for a given local and preference. Each page within the web site has a unique id and is identified by the *page_id*. A given country may have several different dialects of the same language as discussed in Section 2.0. For a given country and language such dialects may be identified by the locale table. *Encoding_id* is the combination of the language and the locale and is used across the entire data model. User preferences are coded in the preference table. The *Content Manager* provides a capability to keep track of historical content using the *history_page_content* table.

4.3. Programming Overview

The overall system consists of ASP files and ActiveX DLLs. The ASP files are divided into 2 main categories: display and process. The display files will consist of HTML/ASP code for creating a UI. Display files include forms, education pages, user messages, login pages, etc. The process files will consist of ASP code for processing the input from a display page. Each process page will be associated with a specific display page (this is true for every process page except *generic_proc.asp*). Every user request to the IIS server will always request a process page that will first initialize the system (ie. language, local, user-state, etc.). The server responds to each request through some specific processing and eventually a display page is returned to the user. The request is fulfilled once the display page is drawn.

4.3.1. Process Flow

Figure 3.0 illustrates the process flow of a web page within the main site. A web client request always launches a process (*_proc.asp*) page. The process page will do some level of processing and then call a display page (*.asp*) which will send HTML back to the user and end the response. A request always begins with a process page and ends with a display page. Every process will begin by including *include_main.asp* (includes the system) and calling *process_init()* (performs system initialization).

process_init() sets the following global variables:

- timestamp = time when received request YYYYMMDDHHMMSS

- encoding_id = language encoding id for the request/response

- preference_id = preference id for the request/response

- db_manager = DB management object initialized to NOTHING

- submit = submit value of the request Sets the submit value of the user request. The submit value is determined by finding the first variable starting with 'SUBMIT_' in the Request.Form or Request.Querystring object. The part of the name to the right of '_' (*submit_goto* results in GOTO, *submit_save* results in SAVE, etc.).

- Draws the appropriate display page if the user request does not require processing (ie. *submit_goto=submit&goto=display1.asp)* This request sent to any processing page will automatically display the page referenced by the *goto* variable, without performing any other processing.

4.3.2. Creating a Web Page

1. Determine the name of the page and create it with an .ASP extension ie.*page1.asp*. Make sure to add a reference for the new page into *include_paint.asp*. This page will consist of straight HTML/ASP code.
2. Determine if your web page will require variable declarations inside it. If yes then you will need an additional file with the same name as your display page with *_disp* extension ie. *page1_disp.asp*. Make sure to add a reference for this page into *include_display.asp*. This page will consist of HTML/ASP code wrapped in functions. The page will have at least one function, the main display function ie. *page1_disp()*, and possibly other functions relevant to that particular display.
3. Determine if your web page requires processing. If yes then you will need another file with the same name as your display page with *_proc* extension ie. *page1_proc.asp*.This file will contain ASP script for processing the results of the display page. Make sure to restart the IIS application (save *Global.asa*) because all *_proc.asp* files get cached in memory by the system.

4.3.3. Database Access

All Database (DB) access is done through a centralized VB DLL, *db_manager*. This DLL provides a simplified DB interface through ADO and ODBC. It also manages DB connections by opening new connections and reusing active connections. All active connections are automatically closed upon object destruction. In ASP, *db_manager* is called through *include_dbconn.asp* which contains functions to provide a simplified interface to *db_manager*. These functions should be used to access *db_manager* in ASP pages. DLLs that require DB connectivity should interface to *db_manager* directly.

5. Conclusion

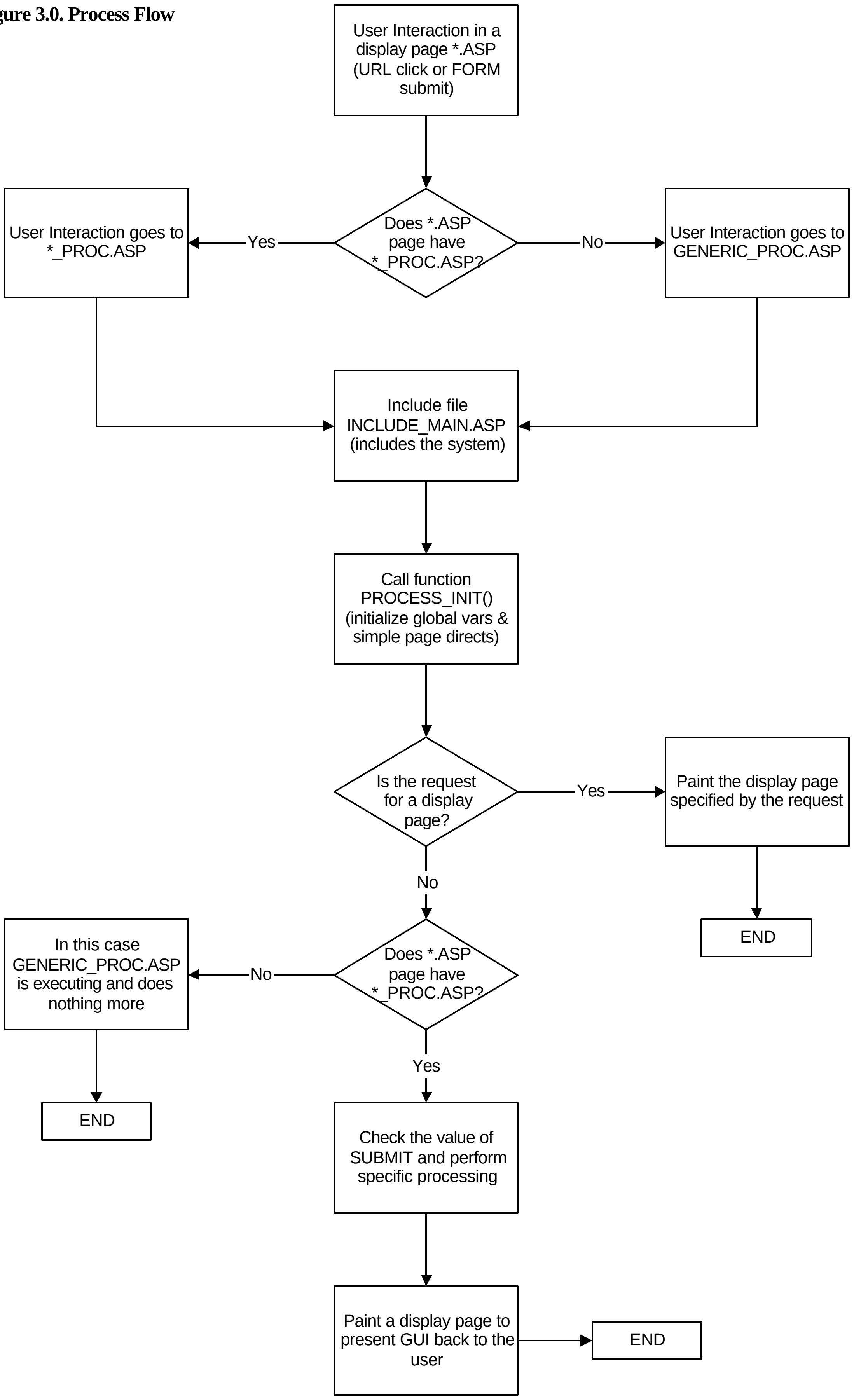
Using *Content Manager* made it possible to develop this multi-country, multi-language, and multi-preference website. In this project, many of the goals and challenges were related to the site's content and therefore the development and use of this tool was critical to the project's success.

Since all the content could easily be managed independently from the code, the development efforts could be directed towards the transaction processing and the complex calculations required of insurance products. Also, having access to the *Content Manager* through the Internet made possible the development, editing, and management of the content from multiple geographical locations. From the project management perspective, this resulted in a more efficient allocation of resources, and lower development times.

It is clear that the audience that can be reached through the Internet is not homogeneous and therefore it should not be treated as such. The *Content Manager* is a tool that can be adapted to suit the requirements of many website development projects that require to reach users at an individual level.

System Process Flow

Figure 3.0. Process Flow



User Interaction in a display page *.ASP (URL click or FORM submit)
Does *.ASP page have *_PROC.ASP?
Yes
No
User Interaction goes to *_PROC.ASP
User Interaction goes to GENERIC_PROC.ASP
Include file INCLUDE_MAIN.ASP (includes the system)
Call function PROCESS_INIT() (initialize global vars & simple page directs)
Is the request for a display page?
Yes
Paint the display page specified by the request
END
No
Does *.ASP page have *_PROC.ASP?
No
In this case GENERIC_PROC.ASP is executing and does nothing more
END
Yes
Check the value of SUBMIT and perform specific processing
Paint a display page to present GUI back to the user
END