An Authoring Tool for Temporal Intensional Web Pages

By
Yatang Hoke
B.Sc, Nankai University 1998

A Thesis Submitted in Partial Fulfillment of the Requirements for the
Degree of
MASTER OF SCIENCE
in the Department of Computer Science

Supervisor: Dr. W. W. Wadge

Abstract

ITPerl is a web authoring tool that enables users to add temporal features to intensional web pages. ITPerl adds temporal macros to IPerl, a Perl interface to the C++ intense library. ITPerl combines temporal and default logic with intensional programming. ITPerl is an ideal solution for many time-sensitive and rapidly changing web applications.

When developing a site, a user can add a temporal section to a webpage by providing different versions of it and associating each version with a time constraint. When a webpage is requested by a browser, a time point will be sent, as part of the request, to the ITPerl module. This time point is by default the current server time, but can be any time specified by the viewer. ITPerl will then compare the time point with each time constraint. If the time point satisfies more than one time constraint, then ITPerl will choose the best-fit version of the section by finding the most refined constraint. A html page that contains this best-fit version of this section will then be generated and sent back to the browser.

ITPerl is easy to use. It provides the user with high level macros that have a simple syntax and that generate temporal codes as cgi files. This frees the user from having to write the complex temporal code himself.

ACKNOWLEDGMENTS

I would like to thank Dr. W. W. Wadge, for his continuous support throughout the whole process of producing this thesis. Thank you, Dr. Wadge, for your guidance and direction.

I am grateful to Paul Swoboda for his excellent work on IPerl, and Lina Liu's THTML, which formed the foundation for my research.

Finally, I am indebted to my husband Justin Hoke for his diligent proof-reading of my work. I would also like to thank my mom and dad, Ying and Xiaoyuan Liao, for their love and dedication to me and my education.

CONTENTS

LIST OF FIGURES

Chapter 1 Introduction

1.1 Overview of the problem, the motivation and the approach

The World Wide Web is a hypertext system that operates over the Internet. It is used for serving web pages and transferring files. Since its birth in the late twentieth century, The World Wide Web has so dramatically changed people's lives that it could easily be called a technological revolution.

The Web is made up of three standards: the Uniform Resource Locator (URL), which specifies how each page of information is given a unique "address" at which it can be found; the Hyper Text Transfer Protocol (HTTP), which specifies how a browser and a server send information to each other, and the Hyper Text Markup Language (HTML), a method of encoding information so it can be displayed through a variety of devices.

With web technology becoming widely used in more and more application areas, one of the shortcomings of traditional HTML, which is becoming more and more obvious, is its lack of interactivity. Specifically, it provides no temporal support. Most time-sensitive applications are currently updated manually by webmasters. Much of this work is redundant and time consuming. Let us take a look at a few scenarios:

Scenario One: A course webpage that needs to be regularly updated. As the semester proceeds, the instructor needs to update the assignments and the reading lists, and post the answers for each assignment and for the exams after they are marked. This schedule can be planned in advance by the instructor, if we assume that he or she has been teaching the courses long enough. Using traditional HTML, we can only update the content of the web pages manually.

Scenario Two: A website called Victoria Dining Guide. In this website, we want to introduce people to different specials from local restaurants in the Victoria area. For example, we know that KFC has a special every Tuesday called “Townie Tuesday”, and we know that Subway has specials on their sub sandwiches every Tuesday and Sunday. Further we know that A&W has a “BOGO” deal, which is valid only from January 1 2005 to February 28, 2005. Our goal is simply to tell people which deals are in effect on any particular day. We can do it by checking all the specials in person and updating the information by hand daily. While some may be motivated enough for such a task, it would be a great deal of work.

Suffice it to say, we can easily find many more scenarios that match these in our daily lives, which require regular updates. In these cases, the updates have a fixed pattern, or are predictable in advance. We want to get away from manually updating these types of pages.

Traditional HTML can obviously not accomplish this task for us. Currently there are only a few ways to make dynamic WebPages, in which the content changes over time. One of them is HTML+TIME technology, also called Timed Interactive Multimedia Extensions[3]. This technology is implemented as the default behavior in DHTML. HTML+TIME defines a schedule, or timeline, for all the affected elements to follow. The main purpose for HTML+TIME technology is to enable synchronization between a wide range of element types. The timing capability is achieved by HTML+TIME's ability to sequence events according to the programmed schedule. HTML+TIME provides attributes you can use to specify an element's timing behavior, such as **begin**, **dur**,

repeatCount, **repeatDur**, and **end** attributes. These attributes are mainly used to add animation and multimedia application to web pages.

Let us take a look at the following piece of DHTML code:

```
<HTML>
<HEAD>
<STYLE>
        .time    {behavior: url(#default#time2);}
</STYLE>
</HEAD>
<BODY>
<DIV CLASS="time" TIMECONTAINER="seq">
<DIV CLASS="time" DUR="2" TIMEACTION="display">First line of
text.</DIV>
<DIV CLASS="time" DUR="2" TIMEACTION="display">Second line of
text.</DIV>
<DIV CLASS="time" DUR="2" TIMEACTION="display">Third line of
text.</DIV>
<DIV CLASS="time" TIMEACTION="display">Fourth line of
text.</DIV>
</DIV>
</BODY>
</HTML>
```

As soon as this page is loaded through a web browser, the first line, which is "First line of the text", will appear. After two seconds, the second line will appear replacing the first line. Similarly, after four seconds, the third line will replace the second one. From then on, the page reminds unchanged unless refreshed.

From this example, we can see that although HTML+TIME technology can be very useful, it is very limited, hence it is not the solution to the problems that we are looking for. This is due to the following factors:

1. DHTML does not provide flexible time representation. The default time unit in DHTML is a second. This makes it very hard to implement applications that use year or even larger time units.
2. DTHML does not support the expression of absolute time. The users are allowed to specify a time point that is relative to the time point that the

webpage loads up, such as “5 seconds after the page loads up, show this picture”. However, there is not an easy way in DHTML to specify an absolute time point or time period, such as “show this picture on March. 18, 2005”. This is due to the fact that in DHTML, the dynamic content of a webpage is interpreted and run by the client’s browser, at the moment when a web page is loaded.

3. DHTML does not support overlapping time constraints. This means, for any elements in a page, the user can only specify one time pattern for how this element will change over time.

Due to the above reasons, the HTML + Time technology is not a satisfactory solution to the problem that we are concerned about.

Another technology, Flash, uses similar methodology as DHTML. Flash allows the users to defines a schedule, or timeline, for all the dynamic elements to follow. However, Flash has similar problem as HTML + TIME. Since it is a client side technology, all the time defined in Flash are relative to the moment when a page is loaded.

Another option is that the user can also choose to directly use a scripting language, such as JavaScript. For example, we can add the following piece of code on top of a page[11]:

```
<SCRIPT language="JavaScript"><!--
theDay = new Date().getDay();
theHour = new Date().getHours();
//--></SCRIPT>
```

The Date() object in JavaScript gives the user access to date, month, and year information, while the Time() object returns current local time including hours, minutes, and seconds. Using these variables, we can do many things to a web page, such as to

show some content of a page only on a certain date and time.

However, this approach also has many limitations that make it not very practical. Firstly, a user has to write code from scratch for every temporal component on the web page, not mentioning the complicated time comparing algorithms. For many applications, the amount of work writing the JavaScript code would not be less than manual updates. Secondly, the individually written, non-generic code is not reusable.

In "Temporal HTML", Lina Liu proposed a much better and more efficient way of solving this problem, THTML, which incorporates discrete temporal logic and default logic with intensional programming. Inspired by the theory presented by Liu, this author has implemented a temporal web authoring tool named ITPerl, using the latest Intensional programming tool, IPerl and intenselib, by Paul Swoboda. This thesis will discuss in detail the design and implementation aspects of this new temporal web authoring tool, and how we can use it to implement complicated temporal web pages.

1.2 Overview of Chapters

Chapter 2 provides background information on the ITPerl. This includes intensional logic and programming, several generations of intensional web authoring tools, from IHTML to ISE and IML, as well as the first proposal of Temporal web authoring, THTML.

Chapter 3 discusses in depth the design approach of ITPerl, mainly how ITPerl implements and uses time in web pages.

Chapter 4 describes the implementation of ITPerl, the temporal web authoring tool.

Chapter 5 covers the actual use of ITPerl. It explains how to set up and run ITPerl, the syntax of ITPerl, and then demonstrates what ITPerl can do.

Chapter 6 is the conclusion of this thesis. It also presents some ideas for future work.

Chapter 2 Background

2.1 Intensional Logic

Intensional (or indexical) logic is the study of assertions and other expressions whose meaning depends on an implicit context or index, such as time or spatial position. The essential aspect of intensional logic is *possible world semantics*, or "*context*". The word "*intension*" is opposite to "*extension*". Extension, according to Dana Scott's definition, is "the truth value of an expression *e* at a particular world." "The intension of φ, on the other hand, he takes to be function that maps each world ω to the extension of φ at ω [1]." For example, Expression *e*

It is raining.

does not have a truth value without giving a particular world. However, if we specify a world ω - with time March 14, 2005 and location Victoria BC Canada, then *e* has a truth value, which is false.

$f_e(\omega)$ = false.

The function f_e is the intension of *e*.

This model is not restricted to propositional logics only. It can be used for numbers, expressions and much more.

In temporal web pages, an expression can be a webpage with multiple versions. Each version is associated with a time constraint. The possible worlds correspond to different requesting time points. Only by giving a specific time point can this temporal webpage have a meaning, which is one specific version of it for a given time point.

2.2 The evolution of Intensional Programming and Web Authoring

Intensional programming is programming using intensional logic. An intensional programming language retains two aspects of intensional logic. First, at the syntactic level, are context-switching operators, called intensional operators. Second, at the semantic level, possible world semantics must be provided for each intensional operator [4].

The first intensional programming language was called Lucid, invented by Bill Wadge and Ed Ashcroft in 1974. Its collection of possible worlds is a programmer-defined multidimensional space, where in each dimension a coordinate is a non-negative integer. It has several intensional operators, allowing access to the value at the next index in a dimension, the previous index, or all indices meeting some criteria[12].

As Wadge points out in [2], "The World-Wide Web is the first large-scale experiment in Intensional Programming.…The most basic semantic concept of IP is the intension – an entity which varies over a space of indices (also called possible worlds)….It should be clear then, that the Web is exactly an intension. The web is nothing more or less than an indexed family of pages, the indices being the URLs."

Since Multi-version web sites are very difficult to produce using conventional tools, intensional web authoring tools have been developed to meet this need.

2.2.1 IHTML

The first intensional web authoring language was Intensional HTML - an extension of HTML which allows a single piece of source to specify a whole family of pages or parts of pages. The first prototype implementation was developed by Taner

Yildirim. Gord Brown redesigned and re-implemented IHTML as a plug-in for the Apache server under Linux, called IHTML 2[13].

> IHTML allows the author to create generic source files for pages and parts of pages. This means that they are valid for a whole family of versions of the component in question. When a request for a particular version of a page is received, the source for that particular version is assembled from the results of specializing the relevant source files for the components. The assembling work is done by a plug-in for the Apache server; the output generated is standard HTML [6].

The different versions of a document are specified by the parameter settings, represented by expressions in a simple algebraic version language. A request for an IHTML page consists of two parts, a conventional URL indicating the name of the page requested, and a version expression indicating which version of the page is requested. The simplest version of an expression consists of a dimension (parameter) identifier and a corresponding value, separated by a colon[14]. For example, “language:English” is an expression.

Although IHTML is a practical approach, it has some serious shortcomings. One of them is lack of modularity. IHTML provides no facility for encapsulating complex markup. The IHTML author has no way of defining new modes for combining markup. Another shortcoming of IHTML is that it is not a programming language. It does not support evaluation of expressions [6].

2.2.2 ISE and IML

IHTML has now been superseded by ISE, a full-featured Perl-like scripting language, designed and implemented by Paul Swoboda [5]. ISE incorporates a run-time parametric versioning system, which means all entities in the language (variables, arrays, functions, etc.) can be versioned. The programs run in an implicit context, which determines which versions of the relevant entities are used. There are also explicit mechanisms for accessing and modifying the context [6]. Figure 1 shows part of the source code of an intensional web page written in ISE:

```
#!/usr/bin/ise

print("Content-type: text/html\n\n");
print(@[<html>
<title>Logic for Artificial Intelligence</title>

]@);
vmod([best({<> -> @[top:hom]@, <top:~> -> "" })]);
print(@[
<h2>Logic and Artificial Intelligence</h2>

<table border=1 bordercolor=white cellspacing=0>
 <tr>

]@);
vswitch{

<@[top:hom]@>{
print(@[
  <td width=140 align=center>
]@);
print("<a href=\"");
print($ENV{"SCRIPT_NAME"});
print("<",[sub:-+secd:-],">");
print("\">");
print(@[<font size=+2>Home</font>]@);
print(@[</a>]@);
print(@[]@);
}
```

```
<@[]@>{
print(@[   <td bgcolor=black width=140 align=center>
]@);
print("<a href=\"");
print($ENV{"SCRIPT_NAME"});
print("<",[top:hom+sub:-+secd:-],">");
print("\">");
print(@[    <font color=white>Home</font>]@);
print(@[</a>]@);
print(@[]@);
}

};
print(@[</td>

]@);
```

Figure 1 Source code of an example webpage in ISE

ISE is a powerful language, and provides many superior features that IHTML never could. However, as we can see from Figure 1, ISE's syntax makes it difficult to program and understand. Hence, IML was created as a front end for ISE to make it easier to use. An IML package is a collection of Groff macro definitions. A webpage written in IML has an extension name .n, and can be translated into ISE by Groff. Figure 2 shows the source code of the same webpage as Figure 1, using the IML macros:

```
.bdoc Logic for Artificial Intelligence
.de tpc

.biselect

.bicase top:\\$2

   <td width=140 align=center>

.balink sub:-+secd:-
<font size=+2>\\$1</font>
.ealink

.eicase

.bicase
   <td bgcolor=black width=140 align=center>

.balink top:\\$2+sub:-+secd:-
    <font color=white>\\$1</font>
.ealink

.eicase

.eiselect
</td>
..
.edoc
```

Figure 2 Source code of an example webpage in IML(.n file)

As we can see, IML significantly reduces the amount of code an author needs to write to produce an intensional webpage. It also greatly improves the readability of the source code. However, this method still has its limitations. The standard convention to make a piece of code with several layers of nested code more readable is to use indentation to identify the beginning and end of the nested code. Due to the syntax of Groff macros, we can not indent the code when we use IML macros.

In order to fix this problem, another set of macros were developed. These macros use UNIX SED to translate a source code, which has an extension name .m, to IML source code, .n files. Using .m macros, a user is free to indent the code whenever it is

necessary. Figure 3 shows the .m version of the same webpage that is shown in Figure 1 and 2. It is very obvious that this version is the shortest, and most readable.

```
{doc Logic for Artificial Intelligence}
.de tpc
 {iselect}
    {icase top:\\$2}
       <td width=140 align=center>
       {alink sub:-+secd:-}<font size=+2>\\$1</font>{/alink}
    {/icase}
    {icase}
        <td bgcolor=black width=140 align=center>
       {alink top:\\$2+sub:-+secd:-}
               <font color=white>\\$1</font>
       {/alink}
    {/icase}
 {/iselect}
</td>
..
{/doc}
```

Figure 3 Source code of an example webpage in IML(.m file)

IML macros dramatically reduced the complexity of the authoring procedure, and made it simple and practical for even unskilled authors to produce multi-version web pages. IML is also extensible. It allows users to define their own macros to meet different needs.

The authoring procedure using IML can be demonstrated with Figures 4 and 5. At compile time, the author can write the source code in .m format. Then this author can use a SED post file to translate this .m file into .n file. The .n file can then be translated in .ise file using Groff, a system for typesetting documents The .ise file is the file that needs to be uploaded to the web server. Note this procedure of translation only needs to be done once for each web page. At run time, when a user requests a .ise file through a web browser, a html file will be generated by the ise compiler on the fly, based on the parameters received.

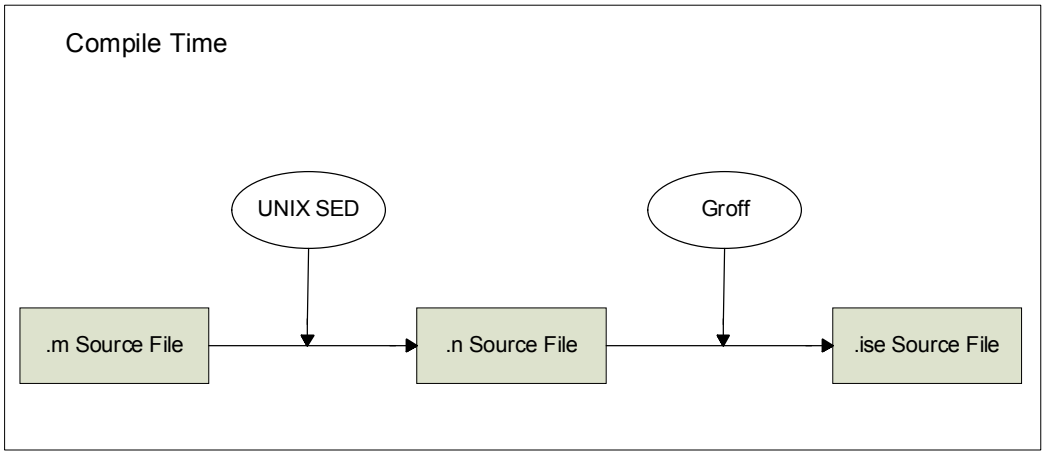

Figure 4 IML authoring procedure at compile time

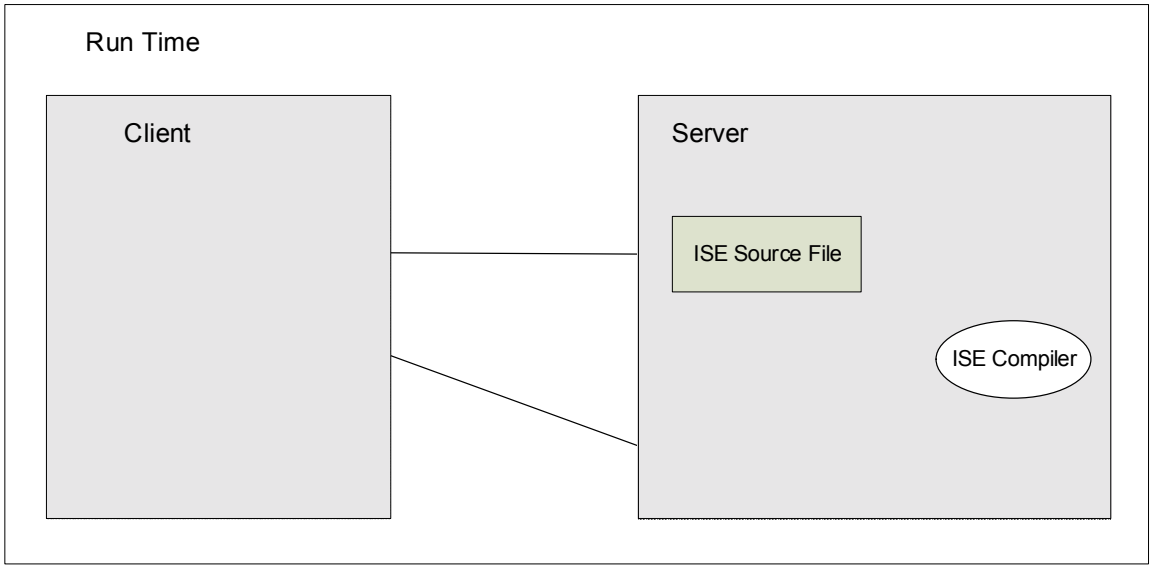

Figure 5 IML authoring procedure at run time

2.2.3 libintense and IPerl

In 2003, Paul Swoboda in his Ph.D. studies, developed a much better and more generic purpose intension library than ISE, called libintense. Libintense is implemented in C++.

In order to take advantage of this much better library and use it for web authoring, IPerl was developed. Paul said in [7] that "The original language combining versions and contexts, ISE, was a toned-down Perl, where everything had been re-written. Here, we take a different approach: we add intensionality to Perl, not by changing its interpreter, but by providing a Perl interface to the C++ libintense code base. By taking this approach, we anticipate attracting a larger user base than ISE ever could."

IPerl not only provides access to the powerful libintense library, it also allows all that Perl has to offer as a bonus. The macros for IPerl are very similar to those for IML, except that the web pages written in IPerl are now translated into cgi files instead of ISE files. The authoring procedure using IPerl is almost identical to IML. Please refer to Figure 6 and 7.

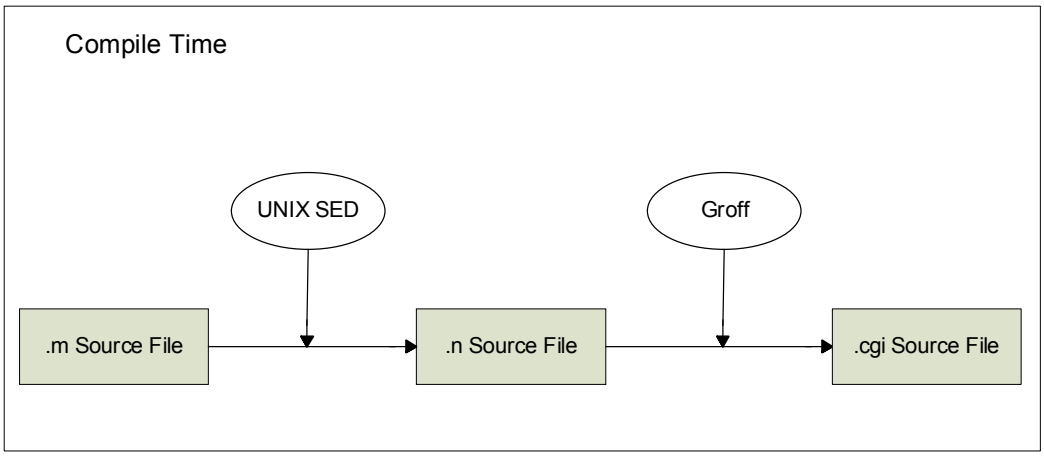

Figure 6 IPerl authoring procedure at compile time

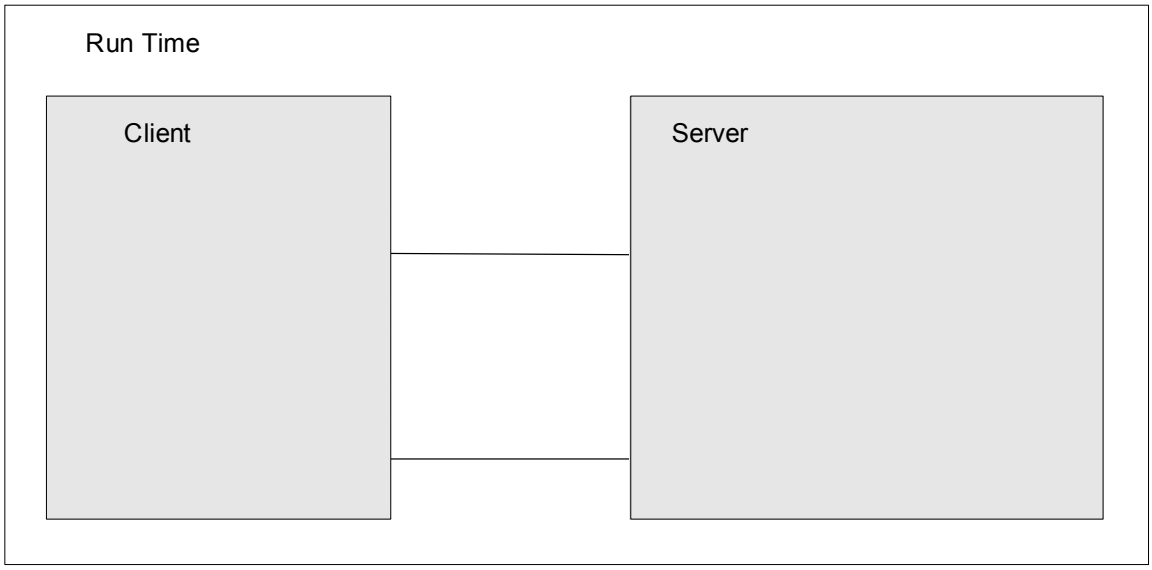

Figure 7 IPerl authoring procedure at run time

ITPerl, developed by the author of this thesis, is built on top of IPerl. It added a temporal module, called Temporal.pm to IPerl. This is the module that handles all the temporal operations. Since ITPerl is an add-on for IPerl, it can work well by itself, as well as with intensional web authoring.

2.3 Temporal web authoring and THTML

Temporal web authoring is one instance of intensional web authoring, which means to add a time dimension to the web page context.

The idea of Temporal HTML was proposed by Lina Liu in 2000. THTML was modeled on the basis of IHTML 2.0. In [8], Liu proposed a very effective way of representing time in temporal web authoring, as well as temporal versioning algorithms. However, since it is based on ITHML, it has all the limitations that IHTML has, as discussed before. More importantly, THTML was not implemented.

The author adopted Liu's idea of adding a time dimension into intensional web authoring, and designed and implemented ITPerl, a temporal web authoring tool, using the newest Intension web authoring technology.

Chapter 3 ITPerl design approach

3. 1 Overview of the design of ITPerl

There are numerous websites that are time-sensitive and that require timely updates. For those that have predictable content-changing patterns, we are looking for a way to free the users from unnecessary manual updates, by allowing them to specify the different versions of the same page in advance. The web page should be able to change its contents automatically over time. Based on these requirements, we need a system that can:

1. Provide mechanisms for users to define multi-versioned web pages.
2. Provide mechanisms for users to define any time point as well as simple or complex time constraints.
3. Provide temporal versioning algorithms to compare time points with time constraints, and to compare between time constraints, and find the most refined one, which is also called the best-fit.

ITPerl is designed to meet these requirements.

3. 2 ITPerl Introduction

ITPerl is designed to allow users to insert temporal sections anywhere in a webpage. It is designed to work in this way:

1. ITPerl provides the users with mechanisms to define different time-constraints. That is, the time intervals when a specific version of webpage is valid.

2. ITPerl allows a user to insert a temporal section into a webpage by pre-defining

the multiple versions of this temporal section. It also requires the user to associate each of these versions with a time constraint, to specify the valid period of each particular version. This is the development process for a temporal webpage. The source code of a temporal webpage will look like this:

```
Normal or intensional code
…
//Start of temporal section
Time Constraint 1:
    Version 1;
Time Constraint 2:
    Version 2;
…
Time Constraint n:
    Version n;
Default: (optional)
    Version n+1;
//End of temporal section
…
```

Figure 8 A demonstration of a temporal web page

3. When a webpage is viewed through a browser, the viewer can specify a time point, either in the past or in the future, as the requesting time point. The viewer can also choose to omit it, in which case the current server time will be taken as the requesting time point. This requesting time point then will be sent to the server together with the URL request. For each temporal section in the requested web page, ITPerl will verify the requesting time point with each time constraint of this section, and try to find a valid one. At this point, there are 4 possibilities:

i. If none of the time constraints is valid under this requesting time point,

then nothing will be returned.

ii. If there is only one valid time constraint, the version that is associated with this time constraint will be returned.

iii. If there happens to be more than one valid time constraint, ITPerl will use its best-bit algorithm to try to find the most refined time constraint. If found, it will then return the version associated.

iv. If ITPerl fails to find the most refined among multiple time constraints, then it will return nothing.

4. After iterating through all the temporal sections of the webpage, ITPerl will assemble the best fit versions of these temporal sections together, and produce the best-fit page in plain HTML. This page will be sent it back to the browser.

The whole procedure is demonstrated in Figure 2:

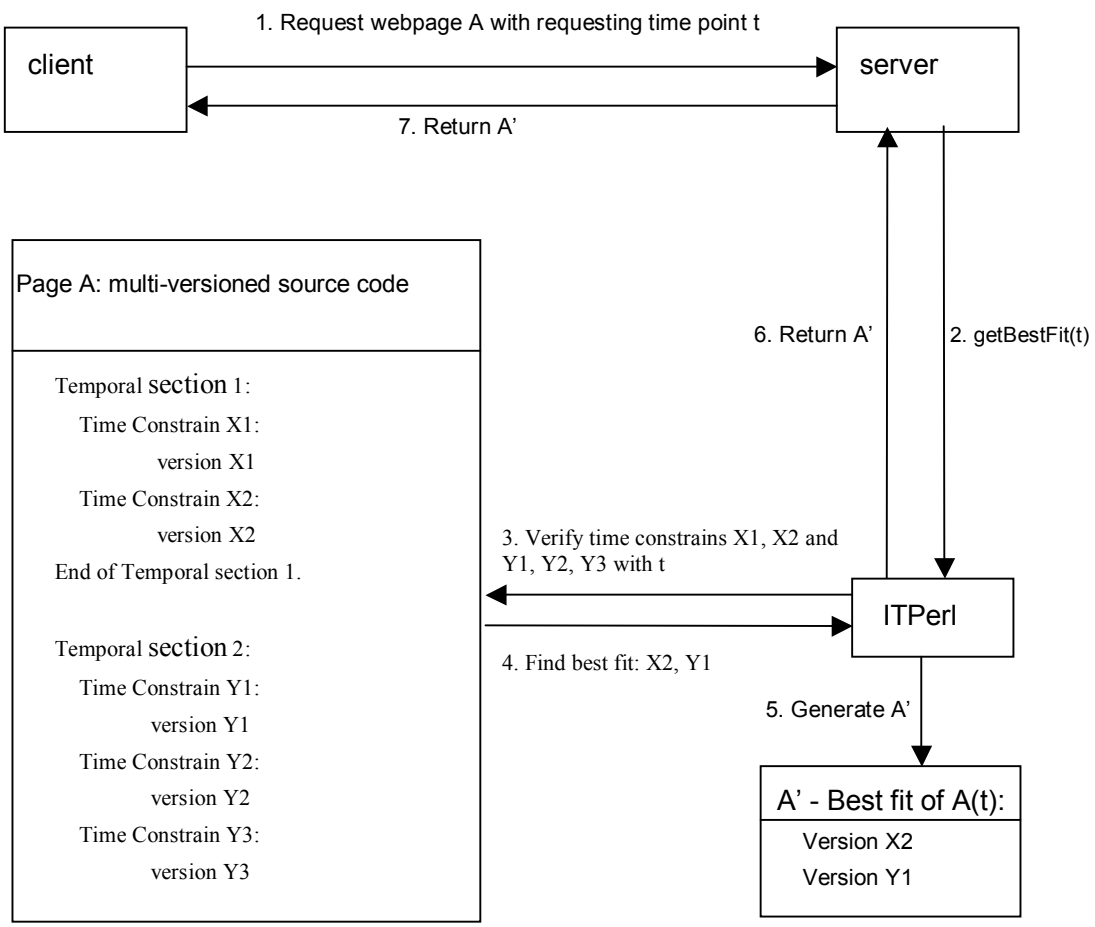

Figure 9 A sample procedure of loading a temporal web page

3. 3 Time representation in ITPerl

The essential aspect of a temporal web page is time. Choosing a simple and efficient way of implementing time and time-related concepts directly affects the usability of any temporal web authoring tool.

Based on the design requirement we discussed earlier in the introduction of this chapter, there are two important time-related concepts: "time point" and "time constraint".

In section 3.3.1 we will discuss the time point representation of ITPerl, and then we will discuss the time constraint representation of ITPerl in section 3.3.2.

3.3.1 Time point representation

When a temporal web page is browsed, there can be one and only one requesting time point. As stated earlier in this chapter, in ITPerl, the default requesting time point is the current server time, which is the date and time that a webpage is viewed. However, ITPerl does provide the users the ability to program the web page in such a way, so that the viewers can change the requesting time point to be any time in the past or in the future to fit their needs. This function of ITPerl will be introduced in more detail in Chapter 4.

In ITPerl, this author adapted the basic time representation form from THTML, which is using seven units:

Second minute hour day month year day-of-the-week (**dow** in short)

In THTML, the scope of the time unit is the same as in Unix:

Second [00-59]

Minute [00-59]

Hour [00-23]

Day [00-30/31/29/28]

Month [1-12]

Year [0-](2-3 digits, after 1900)

Dow [0-6]

Figure 10 The time unit scope of THTML

In order to make ITPerl more user-friendly, this author has made a minor change to the scope of the units. Here is the time unit scope for ITPerl:

Second [00-59]

Minute [00-59]

Hour [00-23]

Day [1-30/31/29/28]

Month [1-12]

Year [0-](4 digits, the actual year)

Dow [1-7]

Figure 11 The time unit scope for ITPerl

Using these 7 basic units, we can specify any time point down to the accuracy of a

second. In ITPerl, we use ":" to separate the time units. For example, we can write

"**1:25:11:25:1:2005:2"**

to represent Tuesday, Feb 25, 2005, 1 sec after 11:25 am.

Not in every case is it needed to be as accurate as a second. In many cases, ***dow*** information is not necessary either. In these cases, we can use a "*" to omit the information that is unnecessary. For example, we can express Feb 2005 as

"***:*:*:*:2:2005:***"

Although by adding the ***dow*** information at the end of a time point expression gives the users a lot of freedom in customizing their time constraints, it can potentially cause conflict and error. For example, if a user specifies both a date value and a ***dow*** value, and these two values do not match, then we have a problem. In order to prevent this type of error from crashing the system, this author chose to give the date information higher priority than ***dow*** information. This means that if a conflict happens, the program will assume that the date is accurate, and calculate the ***dow*** value based on it.

3.3.2 Time Constraint representation

Besides time point, time constraint is also a very important concept that makes it possible for ITPerl to support temporal versioning. A time constraint is a time interval or a sum of many intervals. In ITPerl, we can think of a time constraint as a proposition that has a truth value when given a time point. The time constraints are used to specify when a specific version of a temporal section is valid and when it is not. For example, a user can specify that a webpage should use one style sheet every Sunday, and another style

sheet for the rest of the week.

3.3.2.1 Single time constraints

There are five types of time constraints that are supported in temporal logic. They are: ***point, during, before, after*** and ***recurring.***

Point represents a time point on a single time line.

During is a bound interval, which can be defined by providing the start and end points. In ITPerl, we use "-" to denote during. For example, we can use

"*:*:*:*:1-2:2005:*"

to represent "***during Jan and Feb 2005***".

Before /***after*** are unbound intervals. We can denote a before/after constraint by using "<" or ">" preceding a time point, which stands for the start/end point of the constraint. For example, we can specify before 12:00 am Dec 1, 2006 like this:

"<*:0:12:1:12:2006:*"

Recurring is the sum of several time points or time intervals that appear periodically on the time line. We use "," to denote recurring. For example,

"*:*:*:1,2,3:1,2,3:2005:*"

represents "***The first three days of Jan, Feb, and March of 2005***".

3.3.2.2 Composite time constraints

Using the four time constraints defined in section 3.3.2.1, one can also construct a

composite time constraint. If you combine ***during*** with ***recurring*** time constraints, this will construct a composite time constraint like this:

"*:*:*:*:1-3:2005:1,2,3"

which represents "each ***Monday, Tuesday and Wednesday from January to March 2005***". We can also use

"*:*:*:1-5,10-13:*:2004:*"

to specify "***from the first to the fifth and from the tenth to the thirteenth in each month of 2004***".

3. 4 Temporal versioning algorithms

After discussing the definition of time point and time constraint in ITPerl, we now need to take a look at how these two time components function.

3.4.1 Comparing time points with time constraints

A time constraint is a time interval that has a valid scope. When given a time constraint, a time point can either be in or outside of the scope. When a time point is in the scope, the time constraint is satisfied. If a time point is outside the scope, the time constraint is not satisfied. The process of verifying a time point with a time constraint is a process of checking whether this time point falls into the scope of the time constraint or not.

To verify a ***before*** time constraint, all we need to check is whether the time point is before the end point of the before constraints or not.

Similarly, to verify an ***after*** constraint, we can check and see whether the time point is after the start point of the ***after*** constraint.

To verify a ***during*** constraint, we need to check both the start point and the end point. A time point satisfies a during time constraint if and only if this time point is after the start point and before the end point of this constraint.

It is a little more complicated to verify a ***recurring*** time constraint. We need to view the recurring time constraint as the union of a finite number of ***during*** constraints. We need to iterate through these ***during*** constraints until we find one that is satisfied by this time point. If after we iterate through, we can find none, then we conclude that this time point does not satisfy this recurring time constraint.

3.4.2 Comparing time constraints - Best-fit algorithms

One of the best features that ITPerl provides to the user is multi-versioning, and multi-levels of refinement. A user can define as many versions of a temporal section of a webpage as they want, by providing each version with a time constraint. When given a time point, if there is more than one time constraint that is satisfied, the system needs to decide which one is more refined. The version that is associated with this time constraint will be returned as the ***best-fit*** version. This means that we need algorithms that can choose between constraints, which are called ***best-fit*** algorithms.

There are two rules used for comparing time constraints, which will be discussed in sections 3.4.2.1 and 3.4.2.2.

3.4.2.1 Extensional Rule

In general, when we compare two time constraints, the ideal scenario would be to find one constraint that is a subset of another. In this case we can conclude easily that the subset constraint is more refined. We can use notation "B⊆A" to denote "A refines B".

The extensional rule is this:

For time constraints A and B, if A is a subset of B, which means that for any time point within A, it is also within B, we can conclude that A extensionally refines B.

For ***before, after*** and ***point*** constraints, we can always find one that refines another, when given two constraints that are of the same type.

When comparing two ***before*** constraint, we need to compare the end time point of each constraint. The one with an earlier "end time point" is more refined. See Figure 5 for example.

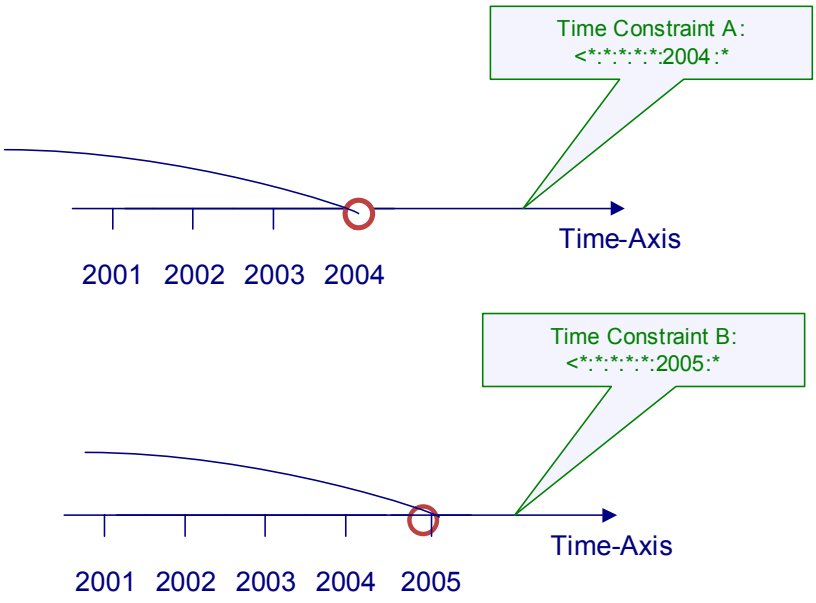

Figure 12 Comparison between two *before* constraints: B⊆A

Similarly, when we need to compare two after constraints, we need to find out which one has a later "start time point", and that one is the more refined constraint. See Figure 3.6.

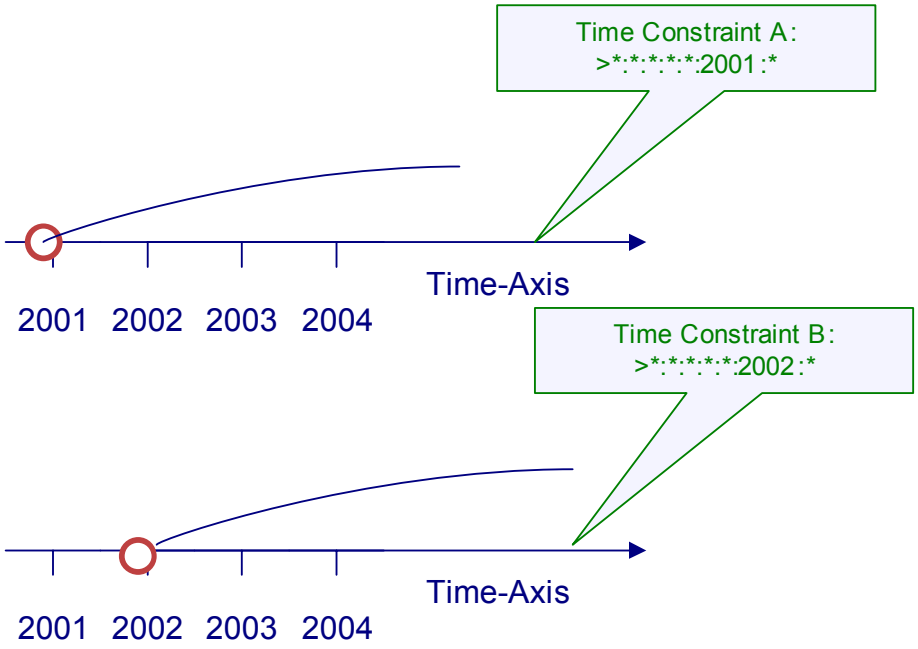

Figure 13 Comparison between two *after* constraints: A⊆B

It becomes more complicated when it comes to comparing ***during*** constraints. For two ***during*** constraints, there may be one that is a subset of another, but there also may not be. If we can find a ***during*** time constraint with a start point that is later than another's start point and with the end point being earlier, then this ***during*** constraint extensionally refines the other. See Figure 7.

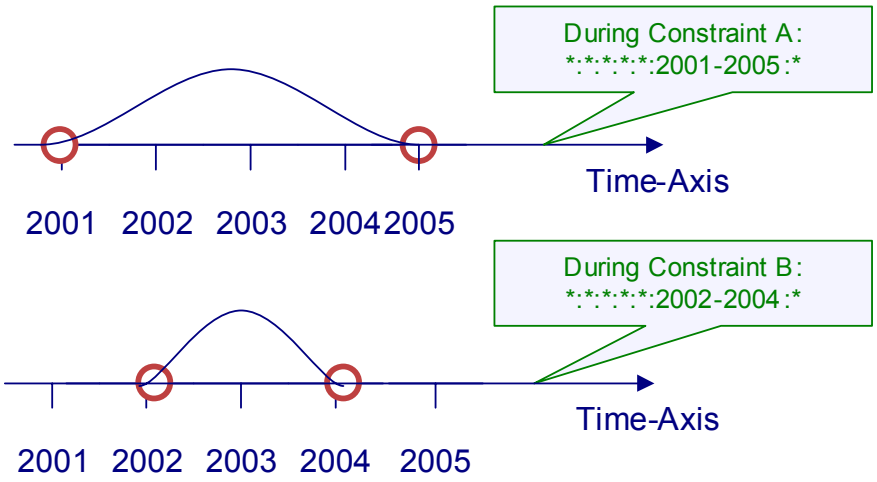

Figure 14 An example of comparison between two during constraints A⊆B

However, if neither constraint is a subset of the other, as seen in Figure 8, then the situation is ambiguous, and the problem will not be solved by applying the extensional rule.

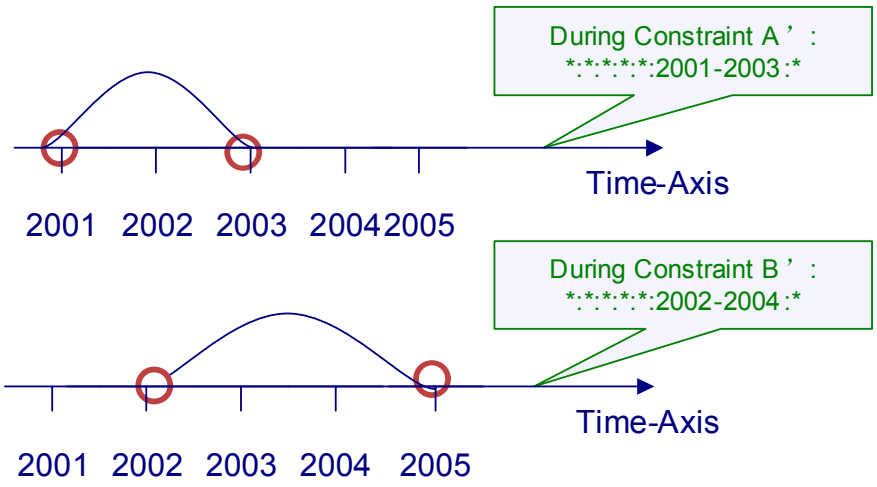

Figure 15 Another example of comparison between two during constraints

The same problem exists when we need to compare two recurring constraints. It is easy to solve if we can find one that is an absolute subset of the other. However, for most of the cases, we will find two that do not encapsulate each other.

For all the rest of the combination of comparisons, ambiguity occurs and is not solvable by applying the extensional rule. In these cases, neither of the versions will be retrieved. Not all the ambiguities are avoidable. However, in many cases, we can reduce them by adding the intensional rule to the system.

3.4.2.2 Intensional Rule

For the extensional rule, we need to have two time constraints where one is a subset of another. For all the other cases, we need intensional rules to decide whether there is a ***best-fit***.

The intensional rule is this:

For two time constraints A and B, if A has smaller intervals than B, then we can conclude that A refines B.

The reason is that a time constraint with smaller time intervals is likely to be more specific. For example, an announcement during a whole term is likely to be less specific than an announcement during an exam week, hence it will be overwritten by it.

The intensional rule can be applied to resolve the ambiguity between two time constraints of the same type, or different type.

3.4.2.2.1 Intensional Rule applied to comparison between time constraints of the same type

As discussed earlier, ambiguity occur when we try to compare two ***during*** and two ***recurring*** constraints. Using the intensional rule, the problem is easy to solve.

- Comparison between two ***during*** constraints: Take the comparison in Figure 8 for example. Since the ***during*** constraint A' has a smaller interval (2 years) than B' (3 years), A' is more refined than B'.
- Comparison between two ***recurring*** constraints: For example, if we have two constraints A and B as follows:

  Constraint A: "*:*:*:1,2,3:1,2005:*"

  Constraint B: "*:*:*:*:1,2,3:2005:*"

  The time interval for A is 1 day, and the time interval for B is 1 month. According to the intensional rule, A is more refined than B.

3.4.2.2.2 Intensional Rule applied to comparison between time constraints of different type

When comparing two time constraints, the intensional rule defines the order of priority of refinement in the following way:

point > during > before/after > recurring

It is easy to see that a ***point*** constraint has the smallest time interval; hence it is the most refined among all the time constraints.

A ***during*** constraint is bound by a start point and an end point, hence is the second most refined.

It is hard to decide whether a ***before/after*** constraint is more refined than a ***recurring*** time constraint. It is mainly a design decision to give the former a higher priority. In special cases where a ***recurring*** constraint does need to have a higher priority

than a ***before/after*** constraint, we can achieve this by expressing the ***recurring*** constraint as the sum of several ***during*** constraints. For example, we can replace the ***recurring*** constraint

"*:*:*:*:1,3,5:2005:*"

with

"*:*:*:*:1:2005:*" + "*:*:*:*:3:2005:*" + "*:*:*:*:5:2005:*"

3.5 Conclusion

In this chapter, we have covered several aspects of the design of ITPerl, including overall design, time representation, and temporal versioning algorithms. There are two time related concepts in ITPerl: time point and time constraint. The time point is used to represent the current requesting time point of a web page. The time constraint is used to specify the valid period of a particular version.

The temporal versioning algorithms include the algorithm of validating a time constraint by using a time point, and the algorithm of choosing the best-fit among multiple valid time constraints. Extensional rules specified that the smallest subset among all the valid time constraints are the most refined. This rule however still leaves some ambiguities. Most of these ambiguities can be resolved by applying the intentional rule, which defines a refinement order for different time constraints. Using these data types and algorithms allows a server to produce a best-fit version of the source file for the requested web page.

Chapter 4 ITPerl Implementation

4. 1 Introduction

As mentioned in Chapter 2, ITPerl is implemented on the basis of IPerl, an intensional web authoring tool developed by Paul Swoboda.

ITPerl consists of three main packages: The C++ library, the Perl interface and the macros, as demonstrated in Figure 9. The intense library is the basis and the core module that performs all the intensional operations. We will introduce this library in more detail in 4.3.1. ITPerl uses the same intense library as IPerl, with no additional code added. The Perl interface package consists of the Perl wrapper classes for the C++ intense library. It contains two Perl modules, one is Intense.pm, and the other is Temporal.pm. Temporal.pm is the center piece of temporal web authoring. All the temporal operations are handled independently in this module. In the rest of this chapter, we will put our focus on this module. Last, but not least, is the macro package. These macros dramatically decrease the complexity of intensional and temporal web authoring. This makes it possible for even non-experienced user to produce intensional and temporal web pages.

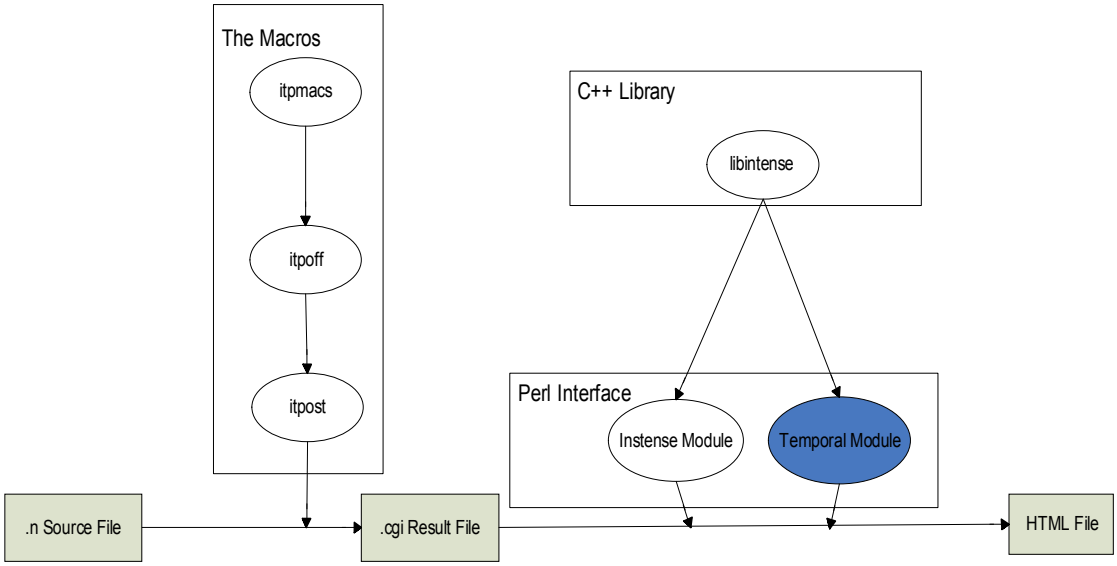

Figure 16 The Overall Structure of ITPerl

4. 2 Chapter overview

In Section 4.3, we will discuss the three major components of IPerl, since most of them are shared by ITPerl. In this section, we will introduce the Intense library, the Perl interface called Intense.pm, and the macro definition file called ipmacs.

In Section 4.4, we will introduce the new modules and functions added in ITPerl. They are, the temporal Perl interface called Temporal.pm, and the new macro definition file called itpmacs.

4. 3 Introduction to IPerl – the basis of ITPerl

ITPerl is implemented on the basis of IPerl. In order to make it easier for the reader to understand how ITPerl works, it is very necessary for us take a brief look at IPerl.

IPerl consists of three components: the intense.pm Perl module, the ipmacs macro definitions and the intense C++ library called libintense. The intense library is the C++ source code which implements all the intensional context switching algorithms. The intense.pm is the Perl wrapper which allows access to the C++ intense library using Perl. The ipmacs is the macro definitions that allow the users to write simple macro code in .n files, instead of complicated .cgi code. By running groff, the .n file will be translated into .cgi files.

4.3.1 Intense library

"Intense is a library of routines created to support intensional programming. Intense supports the creation of tree-structured contexts, and their manipulation using similar, tree-structured context operations [9]."

Below is the list of the classes in this library with brief descriptions.

AEther.cpp: Contexts with participants.

BaseValue.cpp: Abstract base class for version base values.

BinaryBaseValue.cpp: A basic binary-string-valued BaseValue subclass.

BoundBaseValue.cpp: BaseValues to represent bound entities.

Context.cpp: Intensional contexts. Extends a right-threaded AVL tree implementation for optimal space and time efficiency and scalability.

ContextDomain.cpp: Defines a set of Contexts, which can be used to perform intensional best-fits.

ContextManager.cpp: C++ global context manager for libintense.

ContextOp.cpp: Intensional Context Operators.

ContextOpLexer.cpp:

Dimension.cpp: Abstract Dimension.

NumberBaseValue.cpp: Number-valued BaseValues.

SetContextDomain.cpp: ContextDomain implemented as an STL set of Contexts.

StringBaseValue.cpp: Byte-array BaseValues.

StringUtil.cpp: Static utility methods for string handling and conversion in Intense.

4.3.2 Intense.pm Module

Intense.pm is the Perl interface to the C++ intense library. It defines some intensional classes as well as functions. Here are some of the functions it provides to the user defined cgi file to use:

cget: Takes no parameter. It is used to return the current context.

cset: Take 0 or 1 parameter. It is used to set the current context. Set the current context to the value of the parameter, if there is any; or create a new context if no parameter is given.

crefset: It is used to set the current context using a context reference object.

cmod: Use to apply an operation to the current context. It takes a ContextOp object as the parameter.

cswitch: This is one of the most important functions. It takes a block of code as parameter, separate the input as context-code pairs, and store them in a ContextDomain object generated. After all the pairs has been stored, it then calls the best-fit on the ContextDomain object and return the best-fit, or return undef if there is no "best-fit". This function is replaced by a function called "tswitch" in ITPerl, using similar algorithm.

4.3.3 ipmacs – the macro definitions

As ismacs is to IML, ipmacs is the macro definition file of IPerl. It provides the user with easy to use, high-level macros, which reduces the amount and complexity of code a user need to write. The macros in ipmacs are groff macros [15].

For a complete list of macros defined in ipmacs, please refer to Appendix A.

4. 4 Temporal.pm Module

The Temporal.pm module is a Perl interface that works similarly to Intense.pm, except that it handles the temporal requests. Temporal.pm implements the time elements as well as the best-fit algorithms. In the rest of this section, we will discuss this Perl module in detail.

4.4.1 Implementation of time

As mentioned in Chapter 3, the implementation of time in ITPerl consists of two parts: time point and time constraint. We implemented these two concepts as two object

class: TST and TIS.

4.4.1.1 Implementation of time point - TST class

The word "TST" comes from Temporal HTML, standing for "Time Sensitive Tag". In ITPerl, this author adopted it as the name of the class that implements "time point".

The TST class has seven attributes. They are second, minute, hour, day, month, year and dow.

Besides the normal constructor and toString() function, the TST class also provides the following functions:

toArray() – This function returns the value of the seven attributes as an array for easy comparison with another TST object.

toEpoch() – This function converts a time point into Epoch times, which is the number of non-leap seconds since whatever time the system considers to be the epoch. The exact definition of the epoch varies depending on the system. MacOS uses 00:00:00, January 1, 1904, and most other systems use 00:00:00 UTC, January 1, 1970[10]. The return value is used for easy comparison between two time points.

_epochToTST() – This private function takes a epoch value and converts it back to TST format.

compare(TST) – This method takes another TST as a parameter, and compares it

with the TST object it belongs to.

Apply(TimeOp) – This function takes a time operation object TimeOp as a parameter. The definition of TimeOp class will be discussed in Section 4.2.2.2. This function applies the time operation to the current TST, which is either advance it to the future, or set it back to the past.

In ITPerl, we use the variable "$context" to get and set the requesting time point. Same as IPerl and all previous intensional authoring tools, the value of this variable is passed around by the URL of the page. For example, if a page "test.cgi" is loaded with the URL

http://www.abc.com/test.cgi?context=<a:1>,

then the current context is "a=1". Here "a" is a dimension with the value 1. When a temporal page is first loaded, the system will check to see whether there is any value for the time dimension. If not, a TST object with the current server time will be appended to the context. This time value will be passed to the next page that is loaded, and it will be used to calculate the best-fit version. For some applications, the viewer can be allowed to change the requesting time, currently only through a temporal auto link. After a temporal auto link is clicked, the value of the time dimension in context will be changed to the new value, and the webpage will be computed accordingly.

4.4.1.2 Implementation of time constraints –TIS class

The TIS class is the class for time constraint. The name "TIS" comes from

Temporal HTML as well. It originally represents “Time interval stamp”.

The TIS class has ten attributes. The first seven have the same names as those in TST class. However, the values of these attributes are strings. They are simply the raw data from a time constraint expression. For example, for the constraint:

“>*:*:*:1-2:3,4,5:2005:*”

the string “1-2” is stored in the “***day***” attribute, and the string “3,4,5” is stored in the ***month*** attribute.

Besides these seven attributes, the TIS class also has ***$pri, @timePoints,*** and ***$toString*** attributes. The $***pri*** attribute is an integer that is used to determine which kind of time constraint this TIS is. We use 0 for ***point***, 1 for ***during***, 2 for ***before*** and -2 for ***after***, and 3 for ***recurring***. The ***@timePoints*** is an array of TST, which stores a list of time points that are involved in this time constraints, including the start points and end points. A ***before***/***after*** constraint has one element in the ***@timePoints***. A ***during*** constraint has two. A ***recurring*** constraint has twice as many elements as it has intervals, since every interval has a start and end time point. The $***string*** attribute is simply the string form of this TIS.

The TIS class contains many functions that are used for temporal versioning. The first function is the ***constructor***. It takes a string as a parameter. It constructs a TIS object following the steps below:

Step 1. The constructor extracts the date and time information from the parameter, by splitting the string by “:”. After this step, the first seven attributes receive their values.

Step 2. It determines what kind of constraint it is using Perl pattern matching.

Step 3. It extracts the time points based on the type of constraint. As mentioned earlier in this section, a ***before/after*** constraints expects one time point, a ***during*** constraint expects two. For ***recurring***, the constructor iterates through the intervals and extracts twice as many time points as the number of intervals.

The next function of TIS is ***check(TST).*** It takes a time point object TST as its parameter, and returns a Boolean value, representing whether this time point is within the scope of this time constraint or not. When this function is called, it will call another check function based on the type of the constraint.

1. If the time constraint is a ***before/ after*** constraint, it calls ***checkBefore(TST)*** or ***checkAfter(TST)*** function, which compares the time point with the start or end time point of the constraint.
2. If the time constraint is a ***during*** constraint, it calls ***checkDuring(TST)***. This function compares the TST with the start and end point in ***@timePoints*** of the current TIS. If the TST is in between the start and end point, it will return true; or else, it will return false.
3. If the time constraint is a recurring constraint, it calls ***checkRecurring(TST).*** This function iterates through each pair of time points in ***@timePoints*** of the current TIS, and compares the TST with them, until it finds one pair that the TST falls between, then it returns true. If none is found, the function returns false.

4. If the time constraint is a ***point*** constraint, it calls ***checkPoint(TST)***. This function compares the TST with the time point of the TIS. If they are the same, it returns true, otherwise it returns false.

The TIS class can not only be compared with a TST object, it can also be compared with another TIS object. This is done by the function ***refines(TIS).*** This function takes a TIS as its parameter, and returns true if the current TIS refines the parameter TIS. Otherwise, it will return false. The algorithm is like this:

1. It checks the absolute value of ***$pri*** attribute of both TISs. It returns true if the current TIS' absolute value of ***$pri*** value is greater than the absolute value of ***$pri*** of the parameter. It returns false otherwise. Based on the intensional rule we introduced in Chapter 3, time constraints have priorities in following order: ***point > during> before/after > recurring***. Hence when two different types of constraints are compared, the program simply compares their priority level.
2. If the absolute value of ***$pri*** of both TISs are equal, but their values are not, the program returns false. This scenario only happens when comparing a ***before*** constraint with an ***after*** constraint. Since they are of the same priority level, and they can not encapsulate one another, neither one refines the other.
3. If the values of ***$pri*** of both TISs are equal, which means the two constraints are of the same type, the program calls the private refine functions according to the constraint type. There are five private refine functions defined in TIS class. They are: _refinesPoints(TIS), _refinesDuring(TIS), _refinesBefore(TIS), _refinesAfter(TIS), _refinesRecurring(TIS).

4.4.2 Utility classes and functions

The TIS and TST classes are insufficient for the Temporal.pm module to accomplish all the temporal versioning tasks. It needs other helper classes and functions. In this section, we will discuss the TimeDomain and TimeOp classes, as well as some utility functions such as dateAdd(), tswitch() and changeTime().

4.4.2.1 TimeDomain class

The TimeDomain class is the class that stores all of the time constraints and their associated versions of a temporal section for a temporal webpage. It works like a container. It only has one attribute, %timeList, which is a hash array that takes the time constraints as keys and the versions as values. Figure 10 demonstrates how a TimeDomain object stores information.

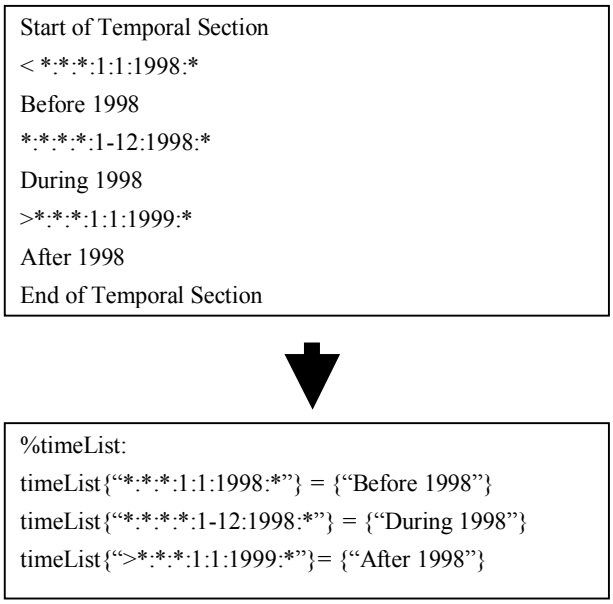

Figure 17 An example of %timeList in a TimeDomain object

The TimeDomain class provides an insert(String, String) function, which adds a pair of time constraint and a version of code into the list. It also provides a best(TST) method, which returns the best-fit version based on the TST given.

4.4.2.2 TimeOp class

The TimeOp class defines time operation. A time operation can be apply to any time point, which will either move it forward or backward a certain amount of time. A time operation expression consists of three parts: a type, a number, and a time unit. The type can be either forward(denote as ">") or backward(denote as "<"). The number can be any positive integer. The time unit can be any one of the following six values: "sec, min, hour, day, mon, year". For example, we can use

> 3 mon

to express "forward the current time point for 3 month."

A TimeOp object is used as a parameter to the apply() function in TST class, as discussed a little earlier in this chapter.

4.4.2.3 Utility functions

Besides the classes discussed earlier, the Temporal.pm package also contains some utility functions that are necessary to help perform the temporal versioning. They are: dateAdd, tswitch and changeTime. Among these, tswitch and changeTime are exported and available for the users to use in the macros or the cgi files.

4.4.2.3.1 dateAdd

DateAdd is the function that handles date adding and subtraction. In order to take advantage of the date validation rules and functions provided by Perl, this author chose to convert all the date information into Epoch format, before any calculations were done. This conversion was achieved by using the built-in function of Perl "localTime()".

The reverse process, which converts the Epoch time back to conventional date format, is also easy. We can use the built-in function timelocal().

One concern of the function dateAdd is the addition of months. The length of a month varies from month to month. For example, the command, "forward the current date one month ahead", can be ambiguous, if the current date is January 31. Should the result of our command be "Feb 28" or "Feb 29"? We can answer this question by checking the calendar. However, this requires complicated computing algorithms. In ITPerl, since the dateAdd function will only be used by the user to change the current requesting time point, and one change can be done in multiple ways, this author believes that to include such a complex algorithm is unnecessary. Currently, a 30 day length is chosen as the value of one month. For example, if the current request time is Jan 31, 2005, and the user requests the time to be one month later, then the system will change the time to be March 2, 2005. If the user needs to get the exact date Feb 28, 2005, s/he can simply subtract 2 days.

4.4.2.3.2 tswitch

The tswitch function is the core function to perform the best-fit algorithm.

The syntax of tswitch is:

tswitch [context][block]

The context is the current context of the program. The block consists of constraint-version pairs, which are time constraints with associated versions of a section.

The algorithm of tswitch function is like this:

1. It retrieves the current context from the parameter list. It then checks to see whether there is a current request time that exists. If there is, it retrieves it. If not, it gets the current server time, and appends it to the current context.
2. It creates a TimeDomain object, then it retrieves the constraint-version pairs from the block of the parameter list.
3. It loops through the constraint-version pairs, and inserts them into the TimeDomain object one by one.
4. It calls TimeDomain->best() to find the best-fit version. If there is none, it returns “undefined”; Otherwise, it returns the best-fit version.

4.4.2.3.3 changeTime

The changeTime function is used to change the current request time point in the context.

It is expected to take two parameters: the current context, and a context operation. The context operation is a ContextOp object. Please refer to section 4.4.2.2 for more information concerning ContextOp objects.

The algorithm of this function is trivial.

1. It gets the current request time, and checks the current context. If the time dimension of the current context has no value, it will give it the value of the current server time. If there is a value, it will retrieve that value.
2. It applies the time operation to the time dimension.

4. 5 itpmacs

Itpmacs stands for "intensional and temporal Perl macros". It is the macro definition file for ITPerl. It has added a few temporal macros to ipmacs. In this section, we will discuss these newly added macro definitions. The syntax of these macros and how to use them will be covered in Chapter 5.

4.5.1 tselect and tcase

The macros tselect and tcase are the two macros for creating a temporal section in a web page. The source code of tselect and tcase are as follows:

```
.rem begin temporal switch statement
.de btselect
.ehtml
tswitch($context, [
..
.
.rem begin the temporal case statement
.de btcase
\'<\\$1>\', sub {
.bhtml
..
.
.de etcase
.ehtml
},
..
.
.de etselect
]);
.bhtml
..
.
```

Figure 18 The macro definitions of tselect and tcase

4.5.2 talink

The macro talink is used to generate a temporal auto link. Through the link, a viewer can change the current requesting time point to view the version of a webpage they need, by clicking a link. The source code of this macro is as follows:

```
.rem begin temporal auto link
.de btalink
.ehtml
$opalink1 ="\\$1";
$opalink2 ="\\$2";
$opalink3 ="\\$3";
$timeOp = new TimeOp($opalink1,$opalink2,$opalink3);
$tempcon = changeTime($context,$timeOp);
$_ = $tempcon->canonical();
s/</%3C/g;
s/>/%3E/g;
s/"/%22/g;
s/\\\\+/%2B/g;
print("<a href=\e"");
print($ENV{"SCRIPT_NAME"});
print("?context=".$_);
print("\e">");
.bhtml
..
.
.rem end temporal automatic link
.de etalink
.ehtml
.bhtml
</a>
.ehtml
.bhtml
..
.
```

Figure 19 The macro definition of talink

4.5.3 .bdoc

The .bdoc macro in itpmacs is a modified version of the one in ipmacs. A function call, checkTime($context), is added to it. What it does is check whether the current context has any information about the requesting time point. If yes, then nothing extra will be done at this point. If no, which means that the user did not specify a time point, the current server time will be assigned as the value of the time dimension.

The source code of .bdoc is listed below:

```
.rem begin an ITPerl document
.de bdoc
$vardom = new ContextDomain();
$query = new CGI;
$domain = new ContextDomain;
print $query->header;
$context = new Context();
$tempcon = new Context();
$context->parse($query->param('context'));
checkTime($context);
.bhtml
<html>
<title>\\$*</title>
.context
<p>
..
```

Figure 20 The macro definition of bdoc

4.6 Conclusion

The implementation and structure of ITPerl were covered in this chapter. ITPerl

consists of three main components: the intense library, the Perl interfaces and the macros. This author's main contribution is the temporal Perl interface called Temporal.pm. In this module, the author implemented two classes to represent time concepts: the time point class called TST, and the time constraint class called TIS. The author also implemented the multi-versioning best-fit algorithm in an exported function, tswitch. This function is available for users in cgi level, and is also used in the macro definition. Besides the Temporal.pm module, this author also added a few macros to the macro definitions. Those macros can be used to easily insert temporal sections anywhere in a webpage.

Chapter 5 ITPerl Application

5. 1 Overview

In this chapter we will cover the topic of application. In section 5.2, we will describe the steps needed to set up ITPerl on a web server. In 5.3, we will answer the question of how to compile and run ITPerl. After that we will introduce the syntax of ITPerl, mainly the syntax of the built-in macros. At last, we will demonstrate an application of ITPerl, showing what we can use this tool for.

5.2 How to set up ITPerl

In order to use ITPerl, one must already have IPerl installed on the web server. This includes the C++ intense library, Intense Perl module and ipmacs installed on the web server.

Please refer to Appendix C for instructions on how to install IPerl.

After IPerl is installed, we can now set up ITPerl. using the following two steps:

Step 1: Install the Temporal.pm module.

There are several ways to install the Temporal.pm module. As the root, one can easily put the file in one of the Perl directories, or install the file under a local directory and append that directory to the @INC array. If a user does not have the root permit, or does not want to install this module to everyone's account, there are a few other easy ways to install the module as well.

This author saved the Temporal.pm module under /home/yatang/lib directory, and

choose to use Perl's ***use lib pragma***. This means, for every Perl/CGI file that uses Temporal.pm, we need to add two lines at the top of the script, like this:

```
use lib "/home/Yatang/lib";
use Temporal;
```

Fortunately, since we use groff macros to auto generate the cgi files, we can just include these two lines in the itpost file. Please refer to step 2 for more information concerning itpost.

Step 2 Install the macro file and auto post file.

With the Temporal.pm module set up correctly, one can now develop temporal web pages in CGI files. However, writing CGI code is very complicated and time-consuming. In order to make it possible for even unskilled users to use this tool, we predefined a groff macro file, called itpmacs. Using the macros defined in itpmacs, all a user needs to know is how to use a few macros with simple syntax.

Besides the macro file, we also need to have two simple auto post files, called itpost and itpoff. The source code for these is as follows in figures 14 and 15:

```
//itpost
echo "#!/usr/bin/perl" >$1.cgi
echo "use Intense;" >>$1.cgi
echo "use CGI;" >>$1.cgi
echo "use lib \"/home/yatang/lib\";" >>$1.cgi
echo "use Temporal;" >>$1.cgi
./itpoff $1.n >>$1.cgi; chmod o+rx $1.cgi
```

Figure 21 Source code for itpost

It should be easy to understand that the purpose of itpost is to add the necessary heading to each cgi script, and redirect the source file to itpoff for processing.

```
//itpoff
: 'converts it arg to itperl - html'
for a
do
 if test -r $a
 then
    cat $a |
    cat $HOME/bin/src/ipmacs - | tee $a.ip |
    groff  -Tascii -
 else
    echo there is no file  $a
 fi
done
```

Figure 22 Source code for itpoff

Itpoff takes a .n file written using macros defined in ipmacs, and processes it using groff. The final result file will be a .cgi file.

After we have setup these files, and set up the path variable, we are ready to use ITPerl to write temporal web pages.

5.3 How to run ITPerl

After we finish setting up ITPerl as instructed, and we have a .n file, we then can run ITPerl and generate a cgi file. As shown in Figure 16, we have a source file named ilai.n. We can simple run the itpost command and give the file name as a parameter. As

shown, two files were generated. The ilai.cgi file is the final result that we can post to a web server. The other file, ilai.n.ip is a middle product that is useful for the purpose of debugging.

```
[yatang@i test]$
[yatang@i test]$ ls -l
total 44
-rwxr-xr-x    1 yatang   yatang      37811 Apr  4 21:24 ilai.n
-rwxr-xr-x    1 yatang   yatang        219 Apr  4 21:24 itpost
[yatang@i test]$
[yatang@i test]$ ./itpost ilai
[yatang@i test]$
[yatang@i test]$
[yatang@i test]$ ls -l
total 168
-rw-rw-r-x    1 yatang   yatang      77474 Apr  4 21:25 ilai.cgi
-rwxr-xr-x    1 yatang   yatang      37811 Apr  4 21:24 ilai.n
-rw-rw-r--    1 yatang   yatang      43911 Apr  4 21:25 ilai.n.ip
-rwxr-xr-x    1 yatang   yatang        219 Apr  4 21:24 itpost
[yatang@i test]$
```

Figure 23 A demo on how to run ITPerl

5.4 Syntax of ITPerl

As mentioned before, the user can write a temporal code directly in CGI script using temporal.pm, or they can choose to use the macros. Here we will only discuss the syntax of the macros that were added by this author in itpmacs.

5.4.1 Syntax of tselect and tcase

In a similar fashion to HTML tags which have open tags and close tags, macros in ITPerl also have open and close tags. A macro starts with ".b*" and ends with ".e*".

Tselect is the macro that is used to add a temporal section in webpage. It starts with ".btselect" and ends with ".etselect", both of which take no parameters. In between ".btselect" and ".etselect", we use tcase statements to add different versions with their correspondent time constraints. We use ".btcase" to start a version and end it with ".etcase". The macro ".btcase" can either take one parameter or no parameters. When given no parameters, it is interpreted by tcase macro as the default version, when no better version can be found. ".btcase" can also take a time constraint as a parameter, surrounded by double quotes. A sample piece of temporal code using btselect and btcase macros can be found in figure 17 below:

```
.btselect
.btcase
Default
.etcase
.btcase "> *:*:*:1:1:1998:*"
After 1998
.etcase
.btcase "*:*:*:*:*:1999-2003:*"
Between 1999 and 2003
.etcase
.btcase "<*:*:*:1:1:2005:*"
Before 2005
.etcase
.etselect
```

Figure 24 Sample code using tselect and tcase macros

5.4.2 syntax of talink

The macro that is used to add a temporal auto link in a webpage is talink. These auto links give the viewers the ability to change the current request time, in order to retrieve information either in the past, or future. The macro talink starts with ".btalink"

and ends with ".etalink". ".btalink" is expected to take three parameters, separated by one or more blank spaces. These three parameters are to be used to generate TimeOp objects. The first parameter denotes the type of time operation, which can either be ">" representing "forward", or "<" representing backwards. The second parameter is a non-negative integer. The third parameter denotes a time unit, which can be "day", "month" or "year". In between ".btalink" and ".etalink", we can put whatever words that needs to be shown for this link. A typical piece of code that uses the talink macro to add a row of temporal auto links is shown in Figure 18:

```
<table>
 <tr><td>
.btalink < 1 day
1day before<br>
.etalink
</td>
<td>
.btalink > 1 day
1day after<br>
.etalink
</td>
<td>
.btalink < 1 mon
1 month before<br>
.etalink
</td>
<td>
.btalink > 1 mon
1 month after<br>
.etalink
</td>
<td>
.btalink < 1 year
1 year before<br>
.etalink
</td>
<td>
.btalink > 1 year
1 year after<br>
.etalink
</td>
</tr>
</table>
```

Figure 25 Sample code using the talink macro

After being posted and published, the page should look like Figure 19.

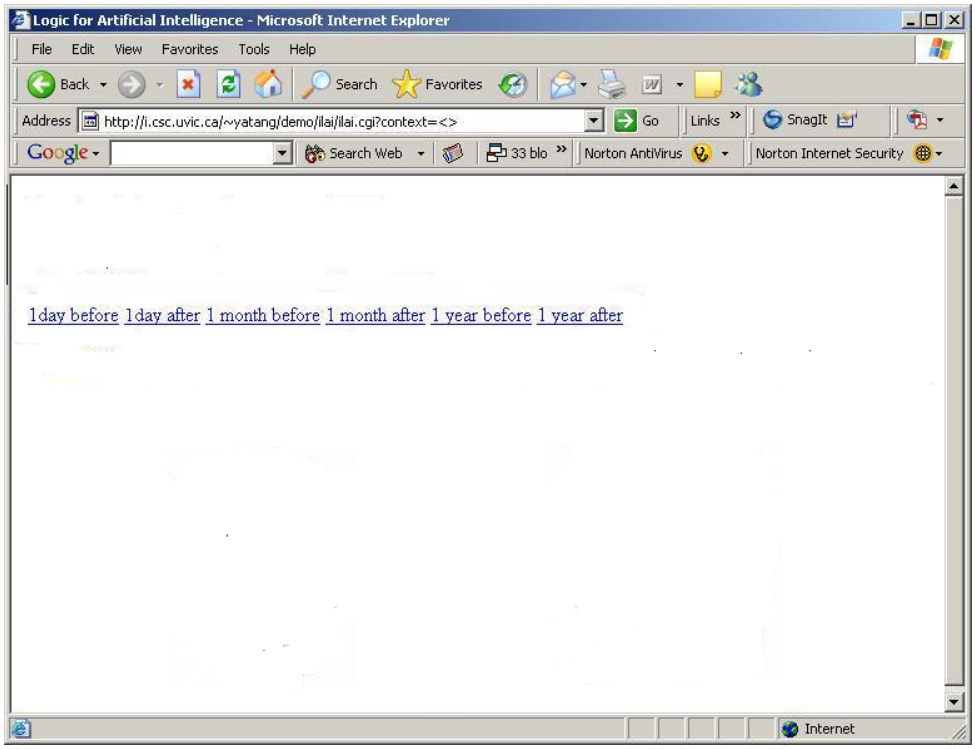

Figure 26 Demo page of temporal auto links

5. 5 An Example – Logic and AI course homepage

This intensional temporal web authoring tool can be used in many time-sensitive applications. In order to demonstrate how ITPerl works and what it can do, the author of this paper has added some temporal features to an existing intensional web site, the "Logic and AI" course web page.

The AI course homepage is a web site that was developed by Dr. Bill Wadge for the "Logic and Artificial Intelligence" course that he taught in the Fall term of 2003. The original course webpage was developed using IML. Using the intensional web authoring technology, the website has great advantages over conventional web sites. The whole site consists of only one source file, lai.ise. However, it can produce multi-versions of the

same page based on different dimensions and values. For example, as shown in Figure 20 and 21, the home version of the page is retrieved when given the context <top:hom>, and the version for final marks is loaded when the context is <top:mar>.

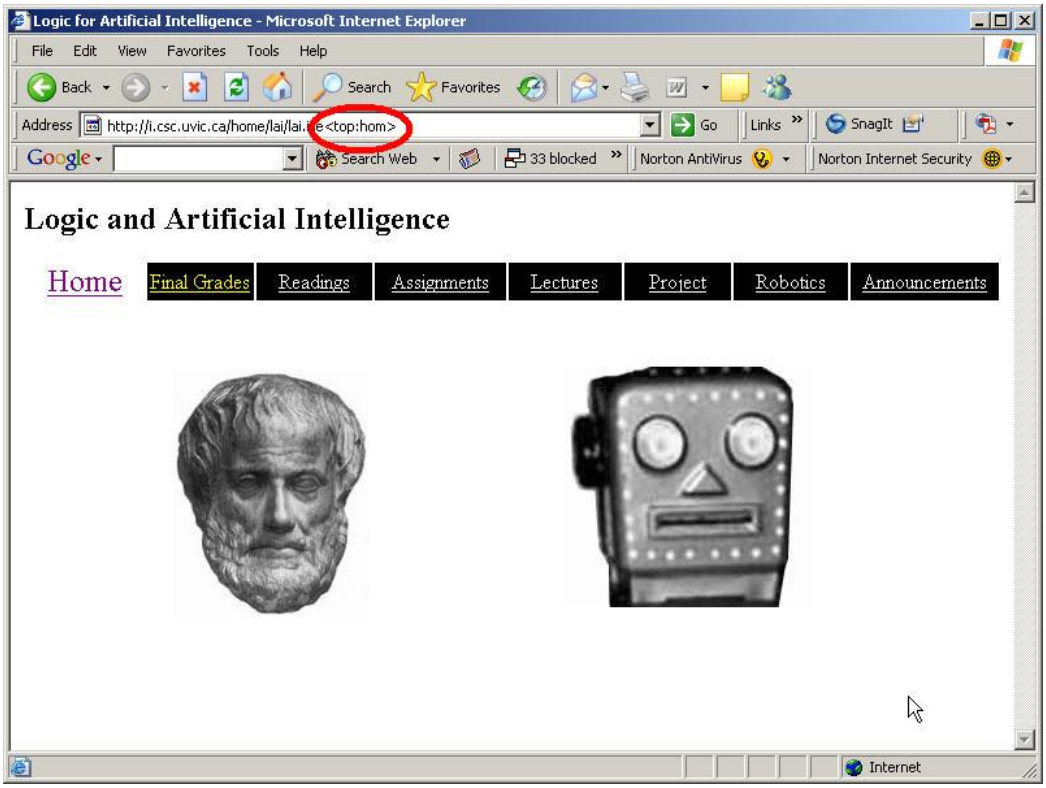

Figure 27 Logic and AI homepage(IML) – Home

As dynamic as it is, the original webpage is static in time. The instructor needs to manually update the information, such as assignments, lecture notes, projects and the marks as time goes on. With ITPerl, we can convert this intensional webpage into an intensional temporal webpage. For changes that are known in advance, no manual update is needed. Let us take a look at what ITPerl can do to this webpage.

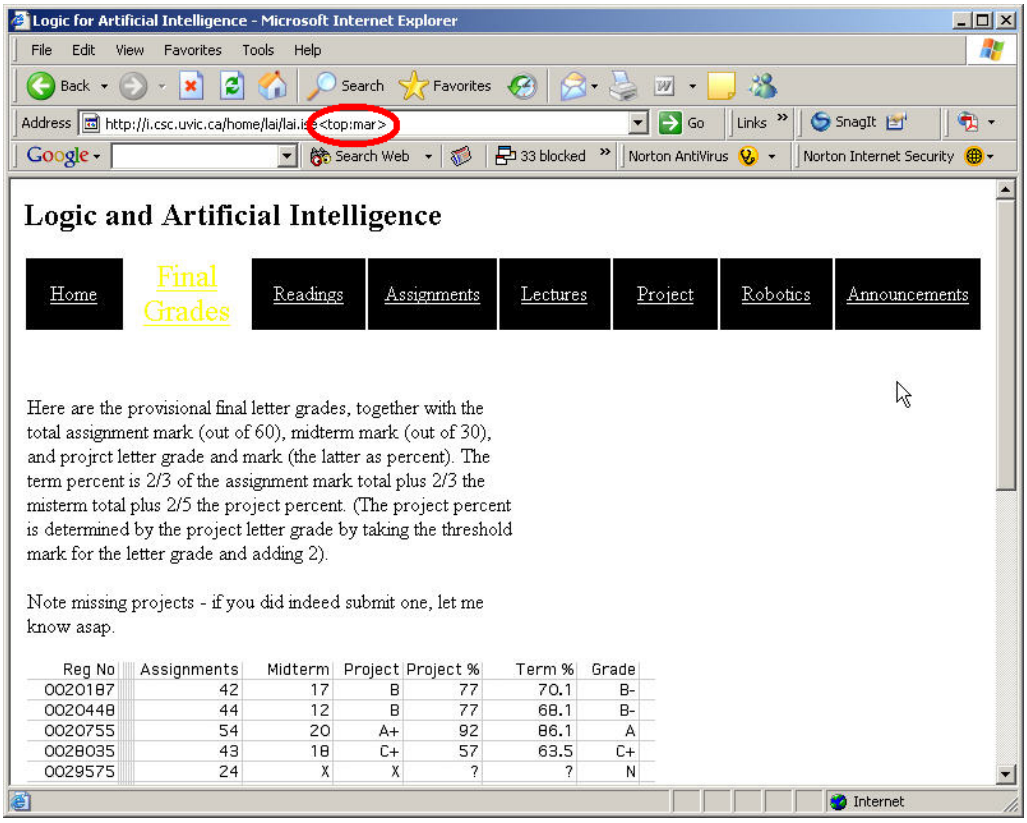

Figure 28 Logic and AI home page(IML) – Final Marks

5. 6. 1. Convert the .n source file to .cgi file

Since the ITPerl macro definitions are absolutely compatible with IML, it was not hard to convert the web page into cgi files using ITPerl.

We can save another copy of the IML source file lai.n, name it ilai.n. We then can run itpost command and convert it into ilai.cgi file.

Now we type in the URL :

http://i.csc.uvic.ca/~yatang/demo/ilai/ilai.cgi?context=%3C%3E

Note that %3C and %3E are the codes for < and >.

Here is what we see:

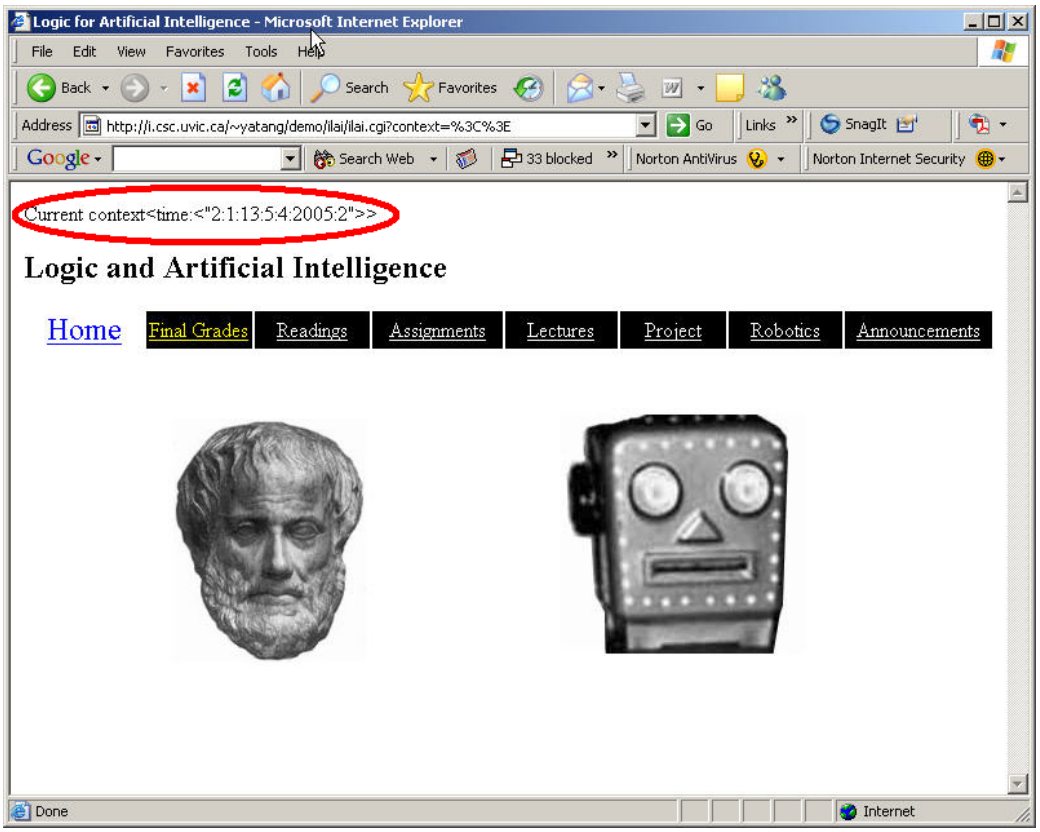

Figure 29 Logic and AI home page(ITPerl) – Home

We can see that there is not much difference between the ITPerl version and the IML version, except the current context value we print out on the top of the screen, for testing purposes. As we can see, we sent our request with context = <>, but the print out value of context is <time:< "2:1:13:5:4:2005:2">>. As mentioned before, when we use the new .bdoc macro to begin a file, it checks the context, and gives the current server time to it, if there is no time information in the context.

5. 6. 2 Using talink macro to add temporal auto links to the page

The next step we need to take is to use talink macros to create some temporal

alinks, so we can change the current requesting time, to see how the page will change with time. We added the following piece of code in the ilai.n source code.

```
<table>
 <tr><td>
.btalink < 1 day
1day before<br>
.etalink
</td>
<td>
.btalink > 1 day
1day after<br>
.etalink
</td>
<td>
.btalink < 1 mon
1 month before<br>
.etalink
</td>
<td>
.btalink > 1 mon
1 month after<br>
.etalink
</td>
<td>
.btalink < 1 year
1 year before<br>
.etalink
</td>
<td>
.btalink > 1 year
1 year after<br>
.etalink
</td>
</tr>
</table>
```

Figure 30 Source code for adding temporal auto link

After posting the source file, we can find a row of links that can allow us to change to requesting time, as show in Figure 24.

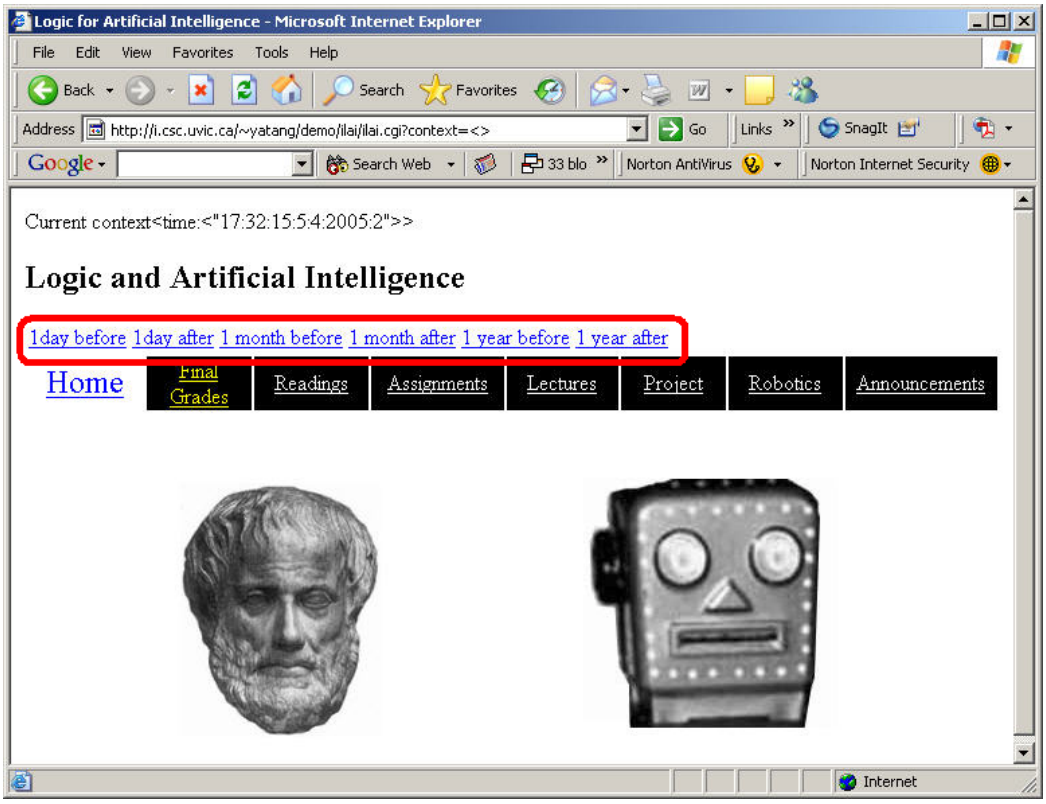

Figure 31 Logic and AI homepage(ITPerl) – talinks

By clicking the links, we can change the requesting time to be any time in the past or future. For example, after clicking on the “1 year after” and “1 month before”, we changed the current context to be “<time:<"19:40:14:6:3:2006:1">+top:<"hom">>”. The current time has been changed to Feb 6, 2006. Note that using ITPerl, we successfully preserved all the intensional features that the page originally had, like the dimension “top:<”hom”>”.

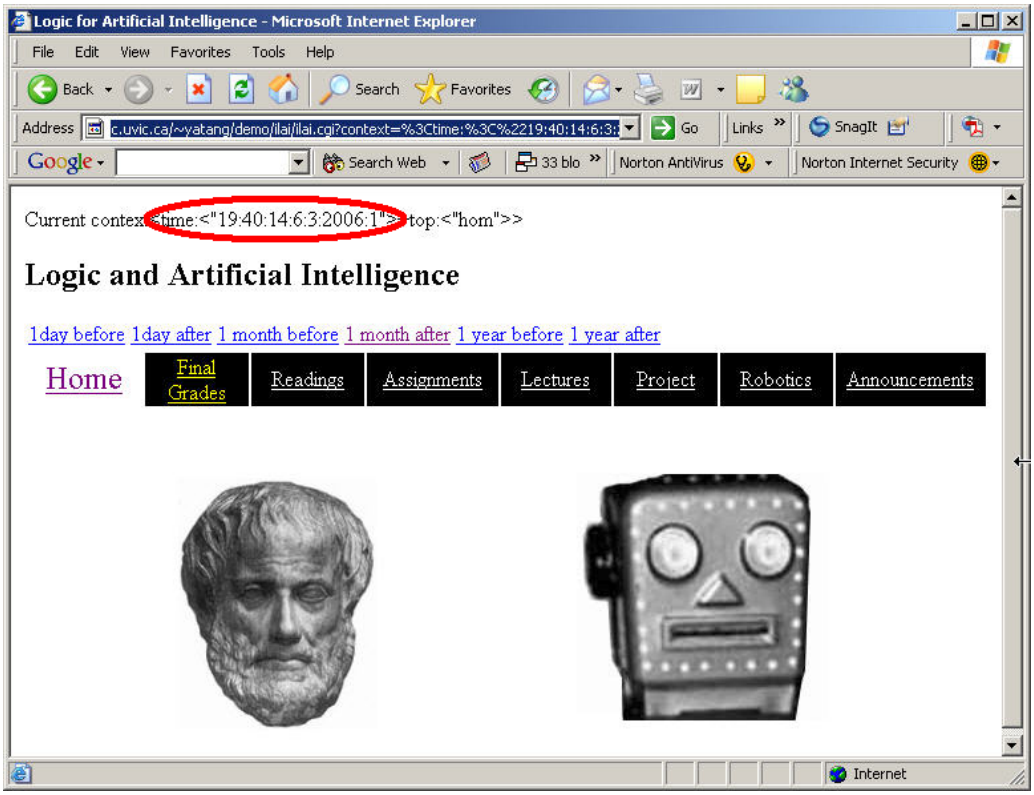

Figure 32 Changing the requesting time using the talinks

5. 6. 3 Adding temporal features to "Final Grades" page

We assume that the final grades will be available after Dec 10, 2005. We can let the page post the marks on Dec 10, using the code in Figure 26.

```
.bdef sn top:mar
.btselect
.btcase
The final grades will be posted here when they are ready.
.etcase
.btcase ">*:*:*:10:12:2005:*"
Here are the provisional final letter grades, together with
the total assignment mark (out of 60), midterm mark (out of 30),
and projrct letter grade and mark (the latter as percent). The
term percent is 2/3 of the assignment mark total plus 2/3 the
misterm total plus 2/5 the project percent. (The project percent
is determined by the project letter grade by taking the threshold mark
for the letter grade and adding 2).
<p>
Note missing projects - if you did indeed submit one, let me know asap.
<p>
<img src=img/grades.gif>
.etcase
.etselect
.edef
```

Figure 33 Code for posting marks on Dec 10, 2005

Now all that is left for the instructor to do is simply mark the exams, save the result file as "grades.gif", put it under the correct directory, and that is it. Now let us test how it works. With the current date, which is Apr 5, 2005, the page shows no final scores. When we change the time to be Dec 10, 2005, the final scores are shown. Please refer to Figures 27 and 28.

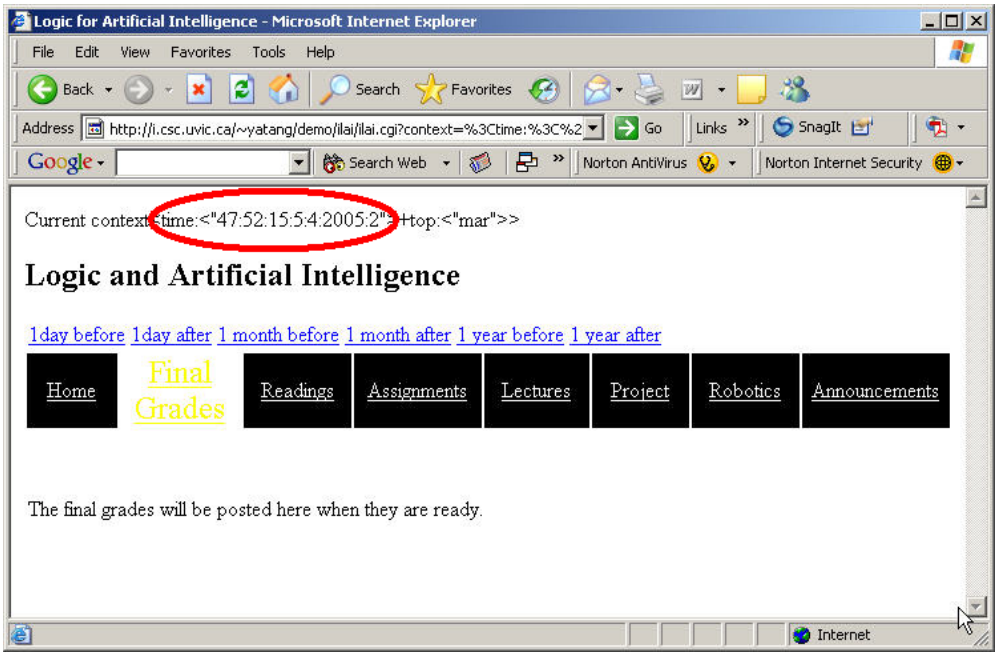

Figure 34 Requesting on Apr 5, 2005 – No final marks available

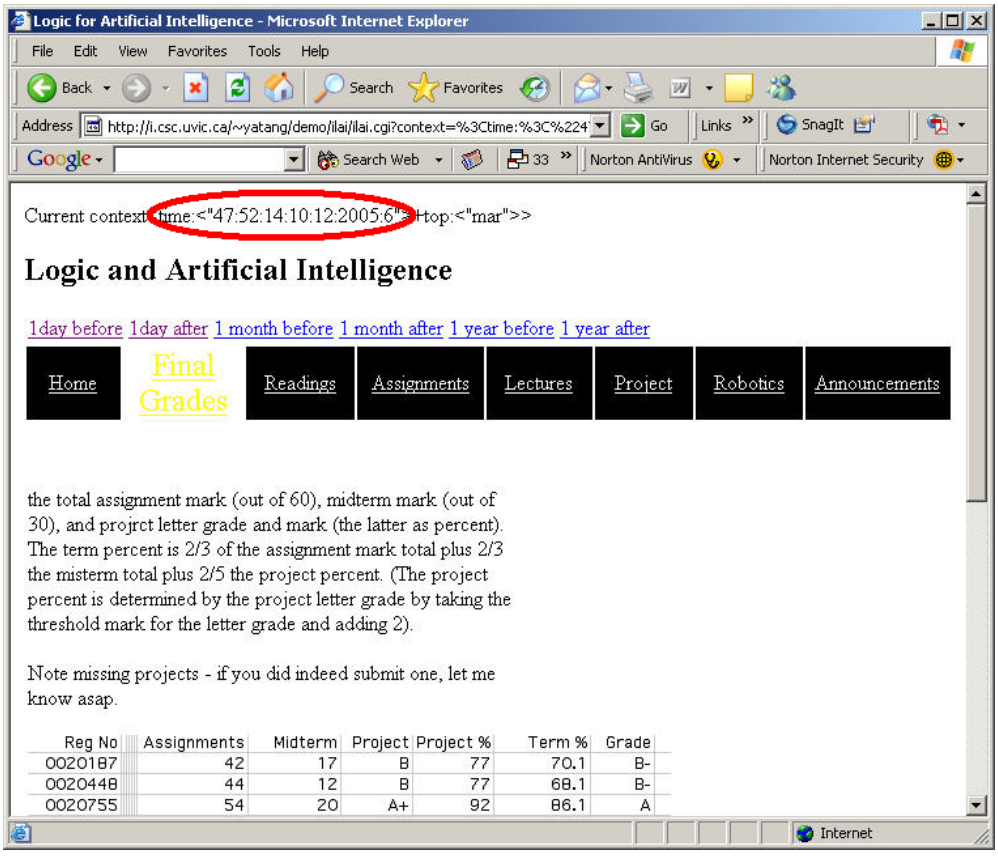

Figure 35 Requesting on Dec 10, 2005 – Final marks are shown

Currently this temporal webpage will only work for fall 2005. That means that any time after Dec 10, 2005, the final marks will be available. What if the course needs to be offered every year? ITPerl can also make this easy. By replacing the "2005" by a "*" in the code shown above, the page will work for every fall semester. The final marks will only be only from Dec 10 to Dec 31 every year. Figure 29 shows that after we made the changes, how the page will look like in Oct 2006.

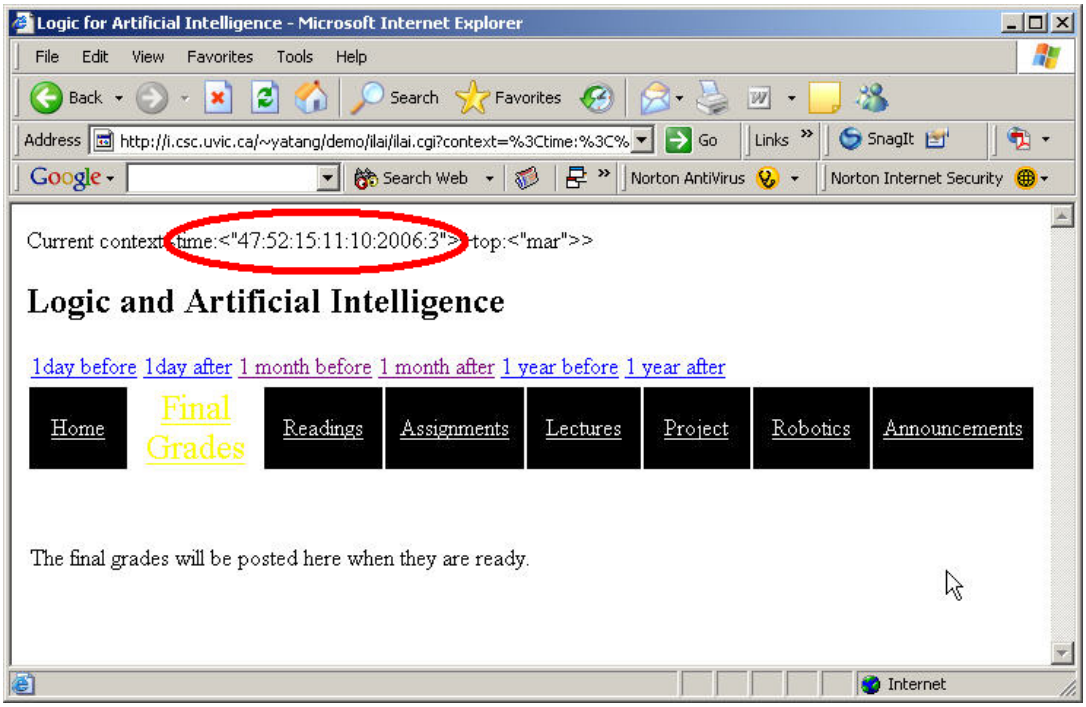

Figure 36 Requesting in Oct, 2006 – No final marks shown

Using similar method, we can turn the reading section, assignment section, lecture section and project section into temporal web pages as well.

5. 6. 4 Converting the Announcement page – an example of using the best-fit algorithm

The simple example in section 5.6.2 demonstrates how convenient a temporal

webpage can be. However, since we have only two versions of a section at any given time in that example, we have not yet used the most powerful feature of ITPerl – the best-fit algorithm. Here we will use a more complex example to demonstrate how ITPerl can pick out the best-fit when there are multiple valid versions.

For the testing purpose, we have made the following assumptions for the announcement section:

1. When no other announcement is valid, the default is "No announcement at this time. Please check back later."
2. From Sept 30 to Oct 5, post the answers for assignment 1 and 2.
3. From Oct 20 to Nov 5, post the reminder of the Midterm.
4. Post "MIDTERM TODAY - Good luck everyone!" on Nov 6, the midterm day.
5. From Nov 6 to Nov 8, post reminder for the due date of Assignment 3.
6. From Nov 30 to Dec 5, post reminder for the final exam.
7. On the final exam day, Dec 5, post "Final Exam – good luck."
8. Since Dec 10 on, post the reminder that the final marks are available.

These requirements can easily be handled by ITPerl, as can be seen in Figure 30.

```
.btselect
.btcase "*:*:*:30-5:9-10:2005:*"       //tcase 1
ASSIGNMENTS - Here are P. Kaminski's answers to assignments 1 and 2.
.etcase
.btcase "*:*:*:20-5:10-11:2005:*"      //tcase 2
MIDTERM - In class, Tuesday November 5, with a take-home part due at the beginning of Wednesday's class.
.etcase
.btcase "*:*:*:5:11:2005:*"            //tcase 3
MIDTERM TODAY - Good luck everyone!
.etcase
.btcase "*:*:*:6-8:11:2005:*"          //tcase 4
ASSIGNMENT 3 - Note the due date - Friday, November 8, in class.
.etcase
.btcase "*:*:*:30-5:11-12:2005:*"      //tcase 5
FINAL EXAM - Dec 6 2005. Close book.
.etcase
.btcase "*:*:*:5:12:2005:*"            //tcase 6
FINAL EXAM - Good Luck
.etcase
.btcase ">*:*:*:5:12:2005:*"       //tcase 7
The final is over! Time to enjoy the holiday seasons!!
.etcase
.btcase ">*:*:*:10:12:2005:*"          //tcase 8
Final Marks are posted. Please check the Final Scores section.
.etcase
.btcase                                //default case
No announcement at this time. Please check back later
.etcase
```

Figure 37 Temporal source code for the announcement page

After we post the file, we are ready to try a few test cases.

Test case 1: Testing for the default tcase.

Requesting Time Point: Sept 10, 2005.

Expectation: Since none of the tcases is valid, the default version should be retrieved and shown in the web browser.

Result: See Figure 31.

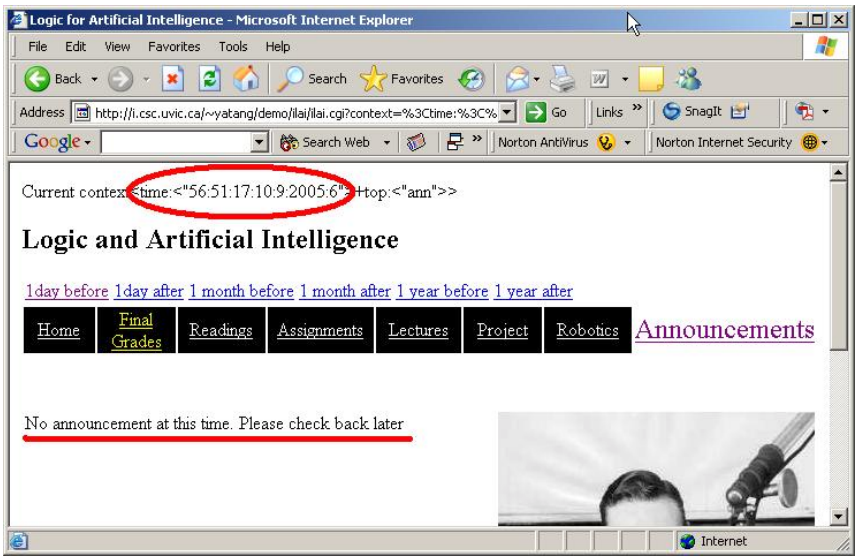

Figure 38 Result for testing case 1

Test case 2: Testing for ***During*** time constraint.

Requesting Time Point: Oct 2, 2005.

Expectation: It is in the scope of "Sept 30 to Oct 5", hence the current announce should be the tcase 2, which is the assignment answers.

Result: See Figure 32.

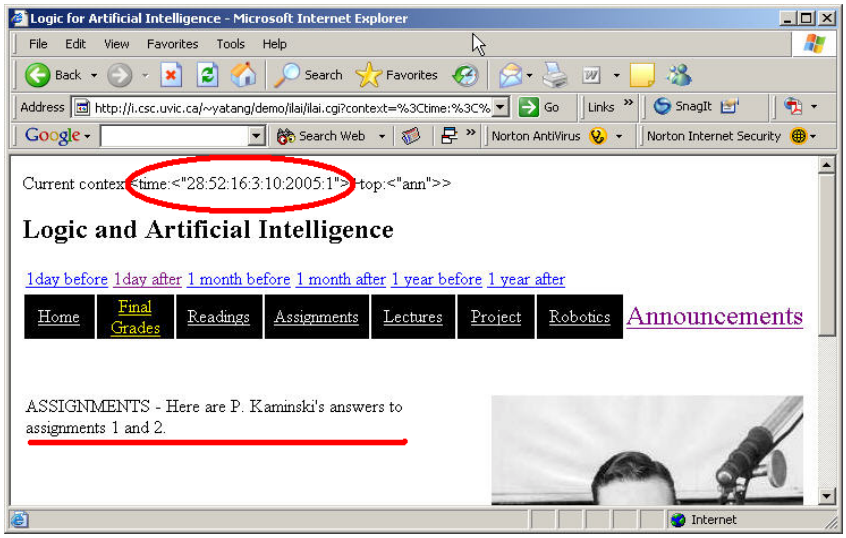

Figure 39 Result for testing case 2

Test case 3: Testing for comparison between a ***Point*** constraint and a ***During*** constraint.

Requesting Time Point: Oct 5, 2005.

Expectation: Both tcase 2 and tcase 3 are valid. Based on the rules we discussed in Chapter 3, tcase 2, which is a ***point*** constraint, has higher priority than tcase 3, a ***during*** constraint. Hence the expected result, should be the "Midterm today" announcement.

Result: See Figure 33.

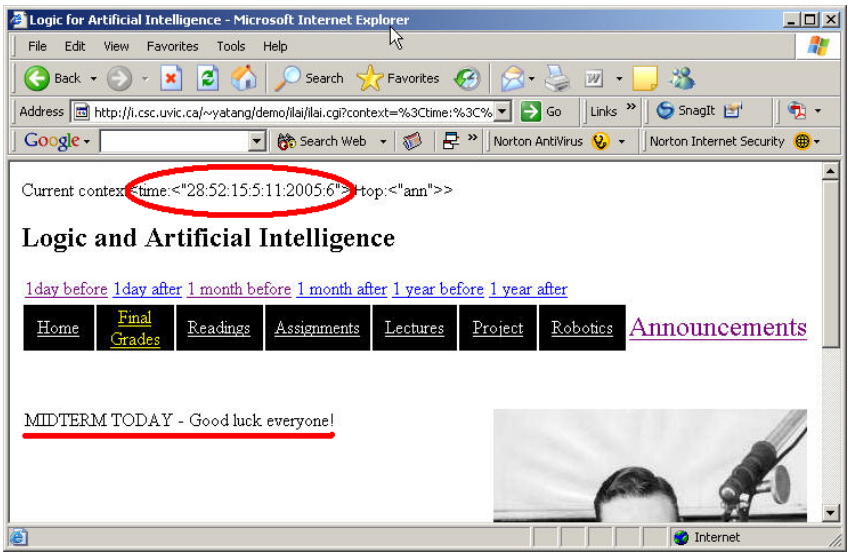

Figure 40 Result for testing case 3

Test case 4: Testing for the comparison of two ***after*** constraint.

Requesting Time Point: The requesting time point is Dec 10, 2005.

Expectation: Both tcase 7 and tcase 8 are valid, and both of the two are ***after*** time constraints. Based on the rules we discussed in Chapter 3, tcase 8 has a later start point, which is a more refined constraint. Hence the expected result should be the "Final marks are available" announcement.

Result: See Figure 34.

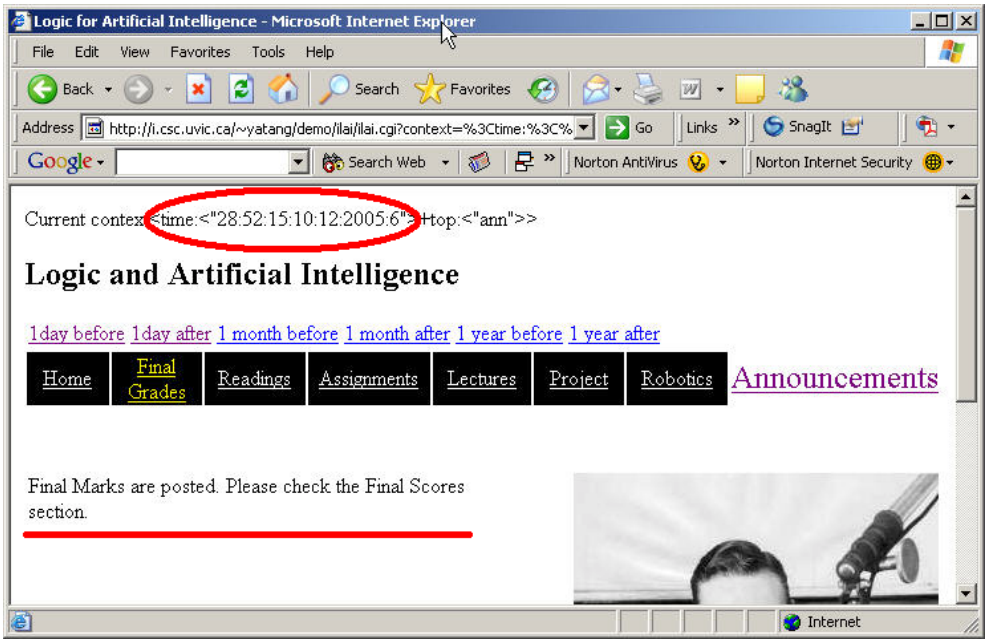

Figure 41 Result for testing case 4

Chapter 6 Conclusion and Future Work

6.1 Add more functions to ITPerl

More functions can be added to improve the functionality, flexibility and the accuracy of ITPerl in the future. Here is a list of the possible functions that might be added in the future:

1. Due to the complexity of implementation, the usage of dow in time constraints is not yet implemented. This is mainly due to the fact that when the use of "*" is allowed, using dow can make things complicated. In the future, algorithms can be developed to specifically deal with the dow calculation.

2. New macros can be developed to enable more flexible methods of changing the requesting time point. Currently, changing the requesting time point can only be done through a link. In the future, a new macro can be developed to allow the viewer to directly input a requesting time. Time validation rules and algorithms need to be developed at the same time as these new macros are developed.

6.2 Improve the run time efficiency

Since the ITPerl is an external Perl module of the C++ intense library, the run-time efficiency might not be the best. When tested in normal applications, the efficiency was not a problem. However, it potentially might not meet the requirement of an extremely large application, to which the run-time efficiency is vital. In those cases, we can consider converting the Perl code into C++ code, and merging it with the C++ intense library.

6.3 Conclusion

ITPerl is an intensional temporal web authoring tool that was designed with the goal of allowing users to make temporal web pages. Based on the research that this author has done, there is no other existing solution that can solve this problem adequately. ITPerl adapted the design approach of THTML, and was implemented using the latest intensional web authoring tool, IPerl. In this way, ITPerl successfully merged the intensional and temporal web authoring technologies, and provides users with the freedom to create dynamic web pages.

Using ITPerl, one can achieve:

- The ability to predefine multiple versions of one section of a web page, with time constraints. The content of the page will be changed by the system over time, according to these predefined versions and time constraints. This frees web authors from having to manually update the pages.
- The ability to enable the viewers of a page to change the requesting time, so that they can not only view the currently information, but also information in the past or future.

ITPerl is not only useful, it is also easy to use. By using the predefined groff macros, it hides the complexity of the multi-versioning algorithms from the end users, which makes it possible for even non-experienced users to make intensional temporal web pages.

Bibliography

[1] W. W. Wadge, "Intensional Logic in Context," Intensional Programming II, World Scientific Publishing, Singapore, 2000, pp1-13.

[2] W.W. Wadge and A. Yoder, "The possible-world wide web," in *Intensional Programming* I, M. A. Orgun and E. A. Ashcroft, Eds. World Scientific, 1995.

[3] Introduction to HTML+TIME, online available:
http://msdn.microsoft.com/library/default.asp?url=/workshop/author/behaviors/time.asp

[4] J. Plaice, "Introduction to intensional programming," in *Intensional Programming* I, M. A. Orgun and E. A. Ashcroft, Eds. World Scientific, 1995, pp1-14

[5] P. Swoboda and W.W. Wadge, "Vmake and ISE, general tools for the intensionalization of software systems," Intensional Programming II, World Scientific Publishing, Singapore, 2000, pp310-319.

[6] W. W. Wadge, "Intensional Markup Language," in Distributed Communities on the Web, Third International Workshop, DCW 2000, Quebec City, Canada, June 19-21, 2000, Proceedings, ser. Lecture Notes in Computer Science, P. G. Kropf, G. Babin, J. Plaice, and H. Unger, Eds., vol. 1830. Springer, 2000.

[7] P. Swoboda, " A Formalization and implementation of distributed intensional programming" Ph.D. thesis, The University of New South Wales, 2003

[8] L. Liu, "Temporal HTML," M. Sc. Thesis, University of Victoria, 2000.

[9] P. Swoboda, "libintense readme", online available at
http://cvs.gna.org/viewcvs/intense/intense/libintense/README?rev=HEAD&content-type=text/vnd.viewcvs-markup

[10] Date & Time in Perl, Online available at
http://www.infocopter.com/perl_corner/date-time.htm

[11] H.Cunningham, "Stupid Web Tricks: Automatic content refreshing", Online available at:
http://builder.com.com/5100-31-5074081.html

[12] H. Li， "An Intensional Tool Applied in French Language Educational Software", M. Sc. thesis, University of Victoria, 2003

[13] W. Wadge, “Intensional Markup”, online available at
http://i.csc.uvic.ca/home/hei/hei.ise%3Ctop:ise%3E

[14] J. A. Plaice and W. W. Wadge, “A new approach to version control”, IEEE Transactions on software engineering, March 1993, pp. 268-276.

[15] Home of Groff, online available at:
http://www.gnu.org/software/groff/

Appendix A The IPerl macro definition - ipmacs

```
Ipmacs
.rem definition of register variables
.ll 132
.nf
.na
.nh
.nr a 0 1
.nr b 0 1
.nr p 0 1
.nr c 0 1
.nr d 0 1
.nr s 0 1
.nr l 0 1

.rem begin an HTML section
.de bhtml
$pagecode .= sprintf <<EOHTML
..
.
.rem end an HTML section
.de ehtml
EOHTML
;
..
.de bperl
.ehtml
.eo
..
.
.de eperl
.ec \\
.bhtml
..
my @cstack = ();
```

```
my $pagecode = "";

sub pageout{
$pagecode .= @_[0];
}

sub randc{
 my $arr = shift;
 my @a = @$arr;
 my $i = rand @a;
 &{$a[$i]}();
}


sub dequote { $_ = @_[0]; s/^"//; s/"$//; return $_; }

 sub bval {
 my $dim = shift;
 return dequote($context->value($dim)->getBase())
}
sub isint
{my $x = shift; return ($x =~ m/[0-9]+/);}

sub nxcoord
{my $tdim = shift;
 $c = bval($tdim);
 if(isint($c)) {return 1+$c;} else {return $c;}
}

sub dimexp{
  $_ = @_[0];
while ( m/^(.+)##([0-9a-zA-Z:]+)##/ )
 {
  $dimcoord = dequote($context->value($2)->getBase());
  $_ = $1 . $dimcoord . $';}
  return $_;
```

```
}


.
.rem begin an IML document
.de bdoc
$vardom = new ContextDomain();
$query = new CGI;
$domain = new ContextDomain;
pageout( $query->header);
$context = new Context();
$tempcon = new Context();
$context->parse($query->param('context'));
.bhtml
<html>
<title>\\$*</title>
.context
<p>
..
.de bdocdb
$vardom = new ContextDomain();
$domain = new ContextDomain;
$context = new Context();
$tempcon = new Context();
$context->parse(('<>'));
.bhtml
<html>
<title>\\$*</title>
.ehtml
pageout('Current context ');pageout($context);
.bhtml
<p>
..
.
.rem end IML document
.de edoc
```

```
</html>
.ehtml
print $pagecode;
..
.rem intensional switch statement.
.de biselect
.ehtml
cswitch($context,[
..
.
.rem begin the case statement
.de bicase
dimexp(\'<\\$1>\'), sub {
.bhtml
..
.
.de eicase
.ehtml
},
..
.
.de eiselect
]);

.bhtml
..
.
.rem add comments to IML document
.de rem
..
.

.rem insert comment into IML document
.de hrem
<!--\\$*-->
..
```

```
.
.rem begin auto link
.de balink
.ehtml
$opalink =dimexp("[\\$1]");
$contextOp = new ContextOp;
if ($contextOp->parse($opalink)) {
    die "non-parsable ContextOp string";
 }
       $tempcon = new Context();
$tempcon->parse($context->canonical());
$tempcon->apply($contextOp);
$_ = $tempcon->canonical();
s/</%3C/g;
s/>/%3E/g;
s/"/%22/g;
s/\\\\+/%2B/g;
pageout("<a href=\e"");
pageout($ENV{"SCRIPT_NAME"});
pageout("?context=".$_);
pageout("\e">");
.bhtml
..
.
.rem end automatic link
.de ealink
.ehtml
.bhtml
</a>
.ehtml
.bhtml
..
.
.rem pageout the current context
.de context
.ehtml
```

```
pageout('Current context ');pageout($query->escapeHTML($context));
.bhtml
..
.
.rem begin a section of intensional perl
.de bipl
.ehtml
.eo
..
.
.de eipl
.ec \\
.bhtml
..
.rem the current url as a string
.de iaurl
.ehtml
$opalink ='[\\$1]';
$contextOp = new ContextOp;
if ($contextOp->parse($opalink)) {
    die "non-parsable ContextOp string";
 }
$tempcon->parse($context->canonical());
$tempcon->apply($contextOp);
$_ = $tempcon->canonical();
s/</%3C/g;
s/>/%3E/g;
s/"/%22/g;
s/\\\\+/%2B/g;
pageout("\e"http://i.csc.uvic.ca");
pageout($ENV{"SCRIPT_NAME"});
pageout("?context=".$_);
pageout("\e"");
.bhtml
..
.de bdt
```

```
<h\\$1 >
.SC
\\$2</h\\$1>
.SD
..
.de edt
.SE
..
.
.de SC
.biselect
.bicase \\n+a:
.balink \\na:on
<img border=0 src=sdarrow.gif alt=">">
.ealink
.eicase
.bicase \\na:on
.balink \\na:-
<img border=0 src=dnarrow.gif alt="V">
.ealink
.eicase
.eiselect
..
.
.de SD
.biselect
.bicase \\na:
\\$*
.eicase
.bicase \\na:on
..
.
.de SE
.eicase
.eiselect
..
```

```
.de bmenu
.ds md \\$1
<form>
<select name=imenu
onChange="window.location.replace(options[selectedIndex].value)">
..
.
.de option
<option value=
.iaurl \\*(md:\\$1
.ehtml
$tempcon->parse('<\\*(md:\\$1>');

cswitch($context,
 ['<>', sub { pageout("");},
 $tempcon,sub {pageout(" selected  ");}]);
.bhtml
>
..
.de emenu
</select>
..
.de initdim
.ehtml
$conop = new ContextOp();
$conop->parse('[\\$1:\\$2]');
cswitch($context,
  [ '<\\$1:~>',sub{$bleenno=2;},
    '<>',sub{$context->apply($conop);}]);
.bhtml
..
.rem begin the definition of an intensional variable
.de bdef
.ds defname \\$1
.ds defcon \\$2
```

```
.ehtml
$varbin = new ContextBinder( sub {
.bhtml
..

.de edef
.ehtml
});
$varbin->parse(\'<v:<\\*[defname]>+c:<\\*[defcon]>>\');
$vardom->insert($varbin);
.bhtml
..

.rem the value of an intensional value (the argument)
.de val
.ehtml
$tempcon->parse(\'<v:<\\$1>+c:\'.$context.\'>\');
$varres = $vardom->best($tempcon);
&{$varres}();
.bhtml
..
.rem begin using a temporarily vmodded context
.de bdo
.ehtml
push(@cstack, $context->canonical());
$conop = new ContextOp();
$conop->parse(\'[\\$1]\');
$context->apply($conop);
.bhtml
..
.rem end using temporary context
.de edo
.bperl
$context->parse(pop(@cstack));
.eperl
..
```

```
.rem the baseval of argument (a dimension)
.de baseval
.ehtml
pageout(dequote($context->value(\'\\$1\')->getBase()));
.bhtml
..
.rem expand the ##dim## expressions in the argument with base values
.de dimexp
.ehtml
pageout( dimexp("\\$*"));
.bhtml
..
.de bgmod
.ehtml
$conop = new ContextOp();
cswitch($context,[
..
.de egmod
]);
$_ = $con;
while ( m/^(.+)##([0-9a-zA-Z]+)##/ )
 {
  $dimcoord = substr($context->value($2)->getBase(),1,-1);
  $_ = $1 . $dimcoord . $';}
$conop->parse($_);;
$context->apply($conop);
.bhtml
..
.de gcase
 '<\\$1>', sub {$con = '[\\$2]';},
..
.de mso
.pso ~/bin/iprep1 \\$1
..
.de bcstr
```

```
.ehtml
\\$1([sub {
.bhtml
..
.
.de more
.ehtml
}, sub{
.bhtml
..
.
.de ecstr
.ehtml
}]);
.bhtml
..
.de vmod
.ehtml
$conop = new ContextOp();
$conop->parse(\'[\\$1]\');
$context->apply($conop);
.bhtml
..

.de bnext
.ehtml
$nxval = nxcoord(\\$1);
.bhtml
.balink \\$1:$nxval
..
.de anext
.ealink
..
```

Appendix B New macro definitions and modified macro definitions in itpmacs

1. tselect – the temporal switch statement

```
.rem begin temporal switch statement
.de btselect
.ehtml
tswitch($context, [
..
.

.de etselect
]);
.bhtml
..
.
```

2. tcase – the temporal case statement

```
.rem begin the temporal case statement
.de btcase
\'<\\$1>\', sub {
.bhtml
..
.
.de etcase
.ehtml
},
..
.
```

3. talink – the temporal auto link

```
.rem begin temporal auto link
.de btalink
.ehtml
$opalink1 ="\\$1";
$opalink2 ="\\$2";
$opalink3 ="\\$3";
$timeOp = new TimeOp($opalink1,$opalink2,$opalink3);
$tempcon = changeTime($context,$timeOp);
```

```
$_ = $tempcon->canonical();
s/</%3C/g;
s/>/%3E/g;
s/"/%22/g;
s/\\\\+/%2B/g;
print("<a href=\e"");
print($ENV{"SCRIPT_NAME"});
print("?context=".$_);
print("\e">");
.bhtml
..
.
.rem end temporal automatic link
.de etalink
.ehtml
.bhtml
</a>
.ehtml
.bhtml
..
.
```

4. bdoc – modified the .bdoc macro in ipmacs and added the checkTime to it

```
.rem begin an ITPerl document
.de bdoc
$vardom = new ContextDomain();
$query = new CGI;
$domain = new ContextDomain;
print $query->header;
$context = new Context();
$tempcon = new Context();
$context->parse($query->param('context'));
checkTime($context);
.bhtml
<html>
<title>\\$*</title>
```

.context
<p>
..
.

Appendix C Instructions for installing IPerl

1. Install the libintense C++ library
   1.1 Get the libintense package. It is available to download at:
   http://cvs.gna.org/viewcvs/intense/intense/libintense/
   1.2. Compile and install the package. The instruction of installation is available at:
   http://cvs.gna.org/viewcvs/intense/intense/libintense/INSTALL

2. Install the Perl module for the intense library
   2.1 Download the Perl module at:
   http://cvs.gna.org/viewcvs/intense/intense/libintense-perl/Intense/
   2.2 Compile and install the package. Simply run the following commands:

```
> perl Makefile.PL
> make
> su
> make install
```

3. Install the ipmacs, ipost and ipoff

   For the source code of ipmacs, please refer to Appendix A.

   The source code of ipost is as followed:

```
echo "#!/usr/bin/perl" >$1.cgi
echo "use Intense;" >>$1.cgi
echo "use CGI;" >>$1.cgi
ipoff $1.n >>$1.cgi; chmod o+rx $1.cgi
```

   The source code for ipoff is as followed:

```
: 'converts it arg to iperl - html'
for a
do
 if test -r $a
 then
    cat $a |
    cat /home/yatang/bin/itpmacs - | tee $a.ip |
    groff  -Tascii -
 else
    echo there is no file  $a
 fi
done
```

PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my thesis (or dissertation) to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain by the University of Victoria shall not be allowed without my written permission.

Title of Thesis/Dissertation:

An Intensional Temporal Web Authoring Tool

Author _____

Yatang Hoke

May 30, 2005