

**DESIGN AND IMPLEMENTATION OF A HIGHLY MODIFIABLE
RETAIL E-COMMERCE WEBSITE**

BY

Mark Soenen

Submitted to the graduate degree program in Electrical Engineering and Computer Science and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

Arvin Agah, Ph.D.
Associate Professor
Chairperson

Committee Members*

James Miller, Ph.D. *
Associate Professor

Prasad Kulkarni, Ph.D. *
Assistant Professor

Date defended: July 18, 2008

The Thesis Committee for Mark Soenen certifies that this is the approved version of the following thesis:

**DESIGN AND IMPLEMENTATION OF A HIGHLY MODIFIABLE
RETAIL E-COMMERCE WEBSITE**

Committee:

Arvin Agah, Ph.D.
Associate Professor
Chairperson*

Jim Miller, Ph.D.
Associate Professor

Prasad Kulkarni, Ph.D.
Assistant Professor

Date Approved: _____

Abstract

The availability, modifiability, and performance of retail e-commerce websites (RECWEB) is greatly impacted by seasonal constraints. For many RECWEB, half of the calendar year is comprised of holidays and seasons. Spikes in website traffic and transactions can lower availability, modifiability, and performance of a RECWEB. This can result in downtime, customer abandonment, and ultimately lost revenue.

This research focuses the modifiability aspects of the problem. During holiday and seasonal periods, enhancements to a RECWEB are generally not feasible. Enhancements put availability and performance at risk. In addition, most human resources are dedicated managing content changes. RECWEB are less modifiable than other systems because enhancements are only feasible for half of the calendar year. Furthermore, the scope of an enhancement must fit within a six month time box.

This research provides pilot project for testing, designing, and implementing a highly modifiable RECWEB. The approach is to automate seasonal content changes. The cost savings on human resources can be reallocated to enhancements work. In addition, enhancements can simulate holiday seasons further in advance. The result is enhancement deployment is more feasible throughout the calendar year.

Contents

1	Introduction	1
1.1	Justification	2
1.2	Significance and Expected Contributions	4
1.3	Research Methodology	5
1.4	Evaluation Criteria	7
1.5	Thesis Organization	8
2	Previous Work	11
2.1	Software Architecture (SA)	11
2.1.1	Definition	11
2.1.2	Functionality	12
2.1.3	Qualities	14
2.1.4	Trade Offs	22
2.2	Typical RECWEB Modifiability	25
2.3	Intelligent Agents	34
2.4	Semantic Web	36
2.4.1	Example #1: Buying a House	39
2.4.2	Example #2: Buying a Digital Camera	39
2.4.3	Resource Description Framework (RDF)	40
2.4.4	Ontology Web Language (OWL)	44
2.4.5	SPARQL Query Language for RDF	46
2.4.6	Justification	47
3	A Highly Modifiable RECWEB	49
3.1	Test Cases	49
3.2	Design	55
3.2.1	Model	56
3.2.2	View	59
3.2.3	Controller	60
3.3	Implementation	61
3.3.1	Model	61
3.3.2	View	65

3.3.3 Controller	65
4 Evaluation	67
4.1 People	68
4.2 COTS Integration	77
4.3 Quality Tradeoffs	77
5 Conclusion	79
Bibliography	83

List of Figures

1.1	Yearly :: Seasonal System	2
1.2	Retail E-Commerce Website :: Seasonal System	2
1.3	RECWEB Traffic [50]	3
1.4	Changing content during P	5
1.5	Deploying enhancements during P	6
2.1	Functionality Example: Sorting Algorithms	13
2.2	Tactics	24
2.3	Google™ Holiday Transition	25
2.4	RECWEB Holiday Transition	26
2.5	Valentine’s Day Scenario	27
2.6	MVC Structure [1]	28
2.7	Model Updates	30
2.8	View Updates	31
2.9	RECWEB Model	32
2.10	Semantic Web Layercake [23]	38
2.11	RDF Triple	41
2.12	Reasoning Example	42
2.13	Continous Merging	44
2.14	Expressiveness of Ontology Description Languages	45
3.1	RECWEB Intelligence Dataflow	55
3.2	RECWEB Seasonal Model	56

List of Tables

1.1	Quality Scenario Parts	5
2.1	RECWEB Sample Tuples	33
2.2	Modifiability Tactics in RECWEB	35
3.1	Selecting a Greeting	50
3.2	Selecting an Image	51
3.3	Selecting Holiday Products	52
3.4	Selecting Products and Categories	53
3.5	Selecting a Seasonal Products	54
4.1	Roles	70
4.2	Cost Estimate - Current Seasonal Transition	71
4.3	Cost Estimate - Seasonal Transition in Highly Modifiable RECWEB	71
4.4	Cost Estimate - New Enhancement Project	73
4.5	Cost Estimate - New Enhancement Project with Shorter Cycles	74
4.6	Cost Estimate - New Enhancement Project with Higher Skills	75
4.7	Cost Estimate - Initial Highly Modifiable RECWEB	76

Chapter 1

Introduction

Enhancements are changes made to a software system during the software maintenance phase. The goal is to add functionality or improve some quality of a system (i.e. performance, availability). Enhancements are vital to a software system [59,60]. In addition, a majority of the cost of software maintenance is planning, designing, implementing, and deploying enhancements [18].

Modifiability is about cost of changing a software system [10]. Since an enhancement is a change to a software system, its cost depends on a system's modifiability. Therefore, low modifiability increases maintenance costs and cripples a system's existence.

This research aims to create a pilot project for highly modifiable retail E-commerce websites (RECWEB). A RECWEB can be described as a *seasonal system*(SS). SS consist of two composite states: off season(O) and peak season(P). Cycle time t begins with O, transitions to P, and ends with transition back to O. Figure 1.1 shows a SS with $t = 365$ days and $t/2$ days between O and P. In other words, half of a year is O and the other half P. For example, a RECWEB shown in Figure 1.2 has a small standard deviation of transition time from one cultural holiday to another. Therefore, P is considered the period between Halloween and Mother's Day. Modifiability of RECWEB is low because

Figure 1.1: Yearly :: Seasonal System

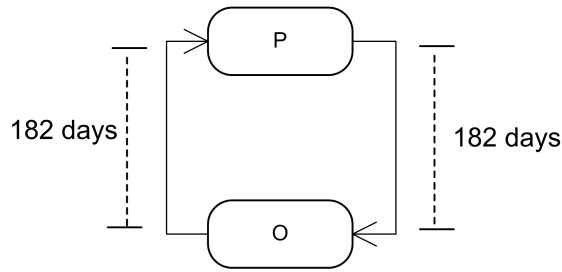
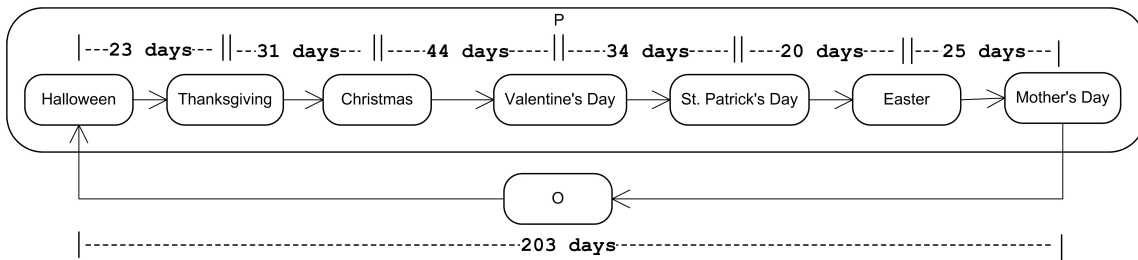


Figure 1.2: Retail E-Commerce Website :: Seasonal System



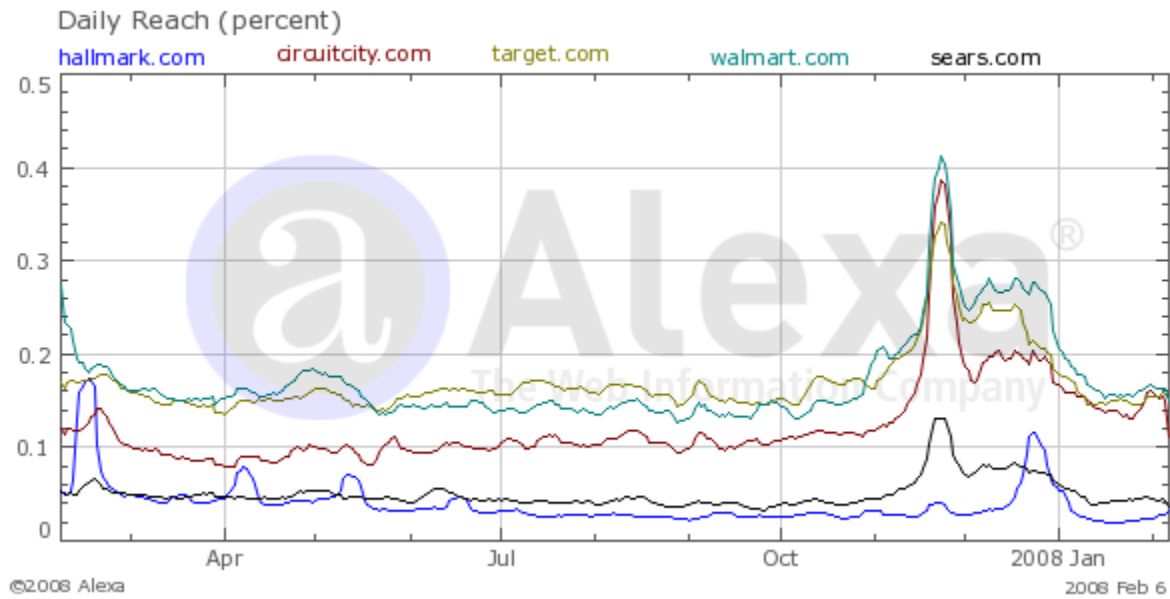
enhancements are generally not feasible during P.

1.1 Justification

E-Commerce is emerging as a key part of the global economy. It continues to grow at a very fast pace. There is lots of competition to deploy highly functional RECWEB that perform well. Also, consumers are only a click away from visiting a competitor site. This puts pressure on organizations to ensure customers find what they are looking for fast. Products and services must be in the right place at the right time. Seasonal patterns previously described just add additional pressure. Depending on time of year, a delicate balance of availability, performance, and modifiability must be addressed.

During P, traffic and transactions are high (see Figure 1.3. Consumers are just a

Figure 1.3: RECWEB Traffic [50]



mouse click away from visiting a competitor. Consequently, a RECWEB must be 100% available and perform as well as possible. Top retailers experience major problems (i.e. server crashes) [16, 29], despite over a decade of P experience coupled with significant research efforts [17, 19, 25, 26, 55, 56, 79, 96] to address performance and availability. Furthermore, modifiability is low during P due to risk of decreasing performance or availability.

Marketing requirements drive changes during P [20]. For example, products such as Christmas ornaments and red roses are more in demand on specific holidays; Christmas and Valentine's Day respectively. This requires website content to change within the short intervals between holidays. In the author's ten years experience working on several large RECWEB, content changes are usually done manually (i.e. data entry or file manipulation). The changes are error prone. Content becomes inaccurate or stale resulting in a plethora of customer service problems and lost sales. There is simply

not enough time between holidays to handle what can be many thousands of content changes. In addition, content changes consume resources normally dedicated to working on enhancements [36, 70].

During O, enhancements are more feasible, but still challenging. First, staff burnout can occur from finishing up work for P. Second, concurrent development (base-lining, testing, integrating, etc.) has its own set of challenges [22, 31–33, 38, 42, 89]. Finally, there is a risk of no return on investment (ROI) if an enhancement project fails to deploy in the six month period before P. In fact, one project the author worked on failed to launch in two consecutive years costing over ten times the original budget. Furthermore, the enhancements were built on a platform that was several versions obsolete.

1.2 Significance and Expected Contributions

The author has extensive experience working on high profile RECWEB projects. Organizations sponsoring such projects all have the same issues caused by seasonal constraints. The problem has costs organizations millions of dollars in both lost revenue and inability to deploy enhancements that can give them a competitive edge. Solving the seasonal problem for these organizations could save millions of dollars.

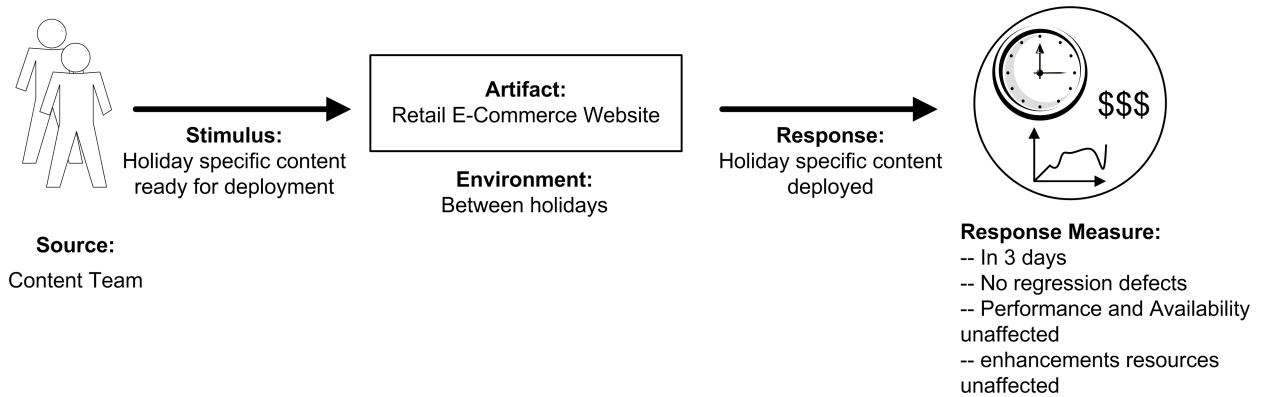
In addition, organizations with similar seasonal constraints but in different domains can also benefit. There is no real body of research surrounding the concept of *seasonal systems* as presented here. This research supports further investigation into the concept.

Finally, emerging technologies such as Semantic Web are used in this work. A foundation for a use case in Semantic Web technology can result. Such a use case does not yet exist.

Table 1.1: Quality Scenario Parts

<i>Part</i>	<i>Description</i>
Source	Entity (human, computer system, actuator)
Stimulus	Condition arriving to system
Environment	Context in which stimulus occurs
Artifact	Part of the system stimulated
Response	Activity undertaken
Response Measure	Metric for testing level of quality

Figure 1.4: Changing content during P



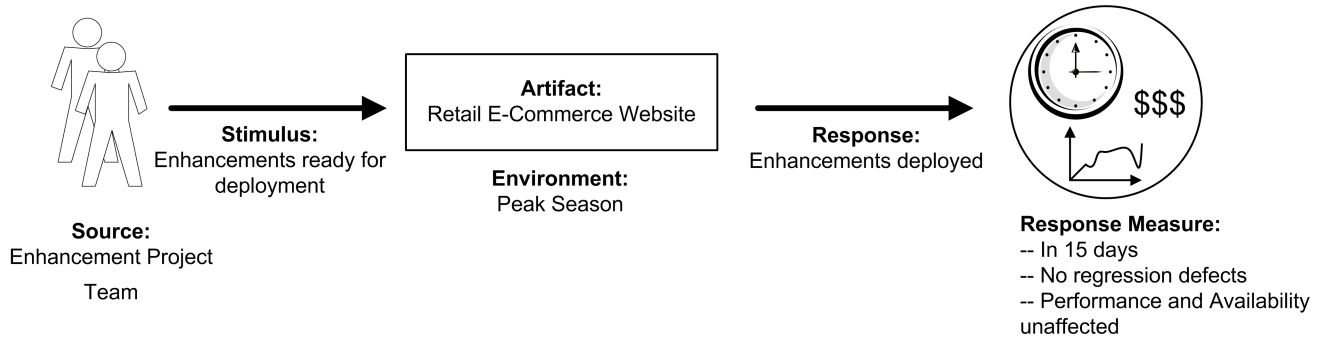
1.3 Research Methodology

Quality attribute scenarios [10] to characterize the modifiability requirements for a RECWEB. Table 1.1 shows the basic parts of a quality scenario. The purpose of using this technique is to define a set of changes and a context in which certain changes will take place.

The first scenario (Figure 1.4) covers content changes during P. In practice, this scenario is usually only partially satisfied. A large number of content changes must be made in a short period of time. This overloads content personnel with work during P.

The second scenario (Figure 1.5) covers deployment of enhancements during P. This

Figure 1.5: Deploying enhancements during P



scenario is very seldom an option unless a high ranking executive demands it. Most resources are dedicated to making content changes and preserving performance and availability.

In order to achieve these modifiability requirements, design decisions or *tactics* must be made that ‘influence the control of a quality attribute response’ [10]. For example, localizing changes, preventing ripple effect, and deferring binding time are three categories containing several modifiability tactics. However, these tactics are either already applied to a RECWEB or just do not satisfy the requirements.

The solution is to automate content changes. This will reduce the human touch points required during P. Freed resources can be used for enhancements work. The automation is not just deploying content changes, but the system actually making the content change. Intelligence can be built into the system to determine, for example, what navigation, products, and promotions should be deployed in a given seasonal context.

The seasonal calendar is fairly predictable. In other words, Thanksgiving is always the last Thursday in November and Christmas on December 25. A system can deduce this knowledge by comparing system time to a seasonal calendar. With this information, the system can intelligently find content to match the particular season or holiday. In order

to accomplish the content match, the system must be able to reason with or understand the data representing the content.

Removing human involvement from the process is not an overnight endeavor. As previously mentioned, most RECWEB are constructed by customizing a COTS package. Automating content changes must be achieved within the bounds of the COTS package. Again, customization of COTS can be challenging. In the previous section, major patterns and tactics a typical COTS-based RECWEB implements are presented. This work references those modifiability points and presents a solution within those bounds. Realistically, any solution would require a phased approach.

1.4 Evaluation Criteria

The first scenario (see Figure 1.4) is essentially in the critical path of the second scenario(see Figure 1.5). Content changes are a necessary task for doing business during P. Adding enhancements to the mix would require taking resources away from content changes and vice versa. Again, content changes are already constrained. Therefore, any solution should address the content change scenario first. Accomplishing this goal alone will increase modifiability because enhancements will be open for deployment all the time.

The goal is to show an increase in modifiability without drastically increasing costs. People, COTS integration, and quality tradeoffs are all factors that constrain costs. The following sections outline specific criteria for each of these constraints.

People The assumption is adding more people is not a desirable solution. First, the author's experience has been RECWEB organizations are already short on people. Second, research shows that adding more people can make a situation worse. [15] In fact,

reducing resources required would be a welcome result.

COTS Integration The leading commercial off the shelf (COTS) packages over 1000 production deployments [49, 83]. Section 2.2 describes how this particular package and many like it are unable to overcome the lack of modifiability. Unfortunately, too much time, money, and resources have been invested in developing and deploying these packages. Enough that replacing them with new packages that would accommodate seasonal constraints is far too costly. Therefore, any solution to the problem has to lay within the extensibility offered by the packages.

Inevitably, any new tactics to increase modifiability should be phased in. Most RECWEB have also been running in production for several years. It is unrealistic to develop a complete solution that will be deployed all at once.

Quality Tradeoff Finally, performance or availability can only be minimally affected. Ideally, the required downtime for rolling content changes can be eliminated actually increasing availability. Any effects on performance should not be directly observable by users. For example, page load times should remain close to the same as they are today.

1.5 Thesis Organization

The thesis is organized into the following sections:

Chapter 1: Introduction – The background and justification for the work contained in this thesis.

Chapter 2: Previous Work – A detailed account of previous work in Software Architecture, Typical RECWEB Modifiability, Intelligent Agents, and Semantic Web.

Chapter 3: A Highly Modifiable RECWEB – Test cases, design, and implementation of a highly modifiable RECWEB.

Chapter 4: Evaluation – Evaluation and analysis.

Chapter 5: Conclusion – A conclusion regarding the results of this work.

Chapter 2

Previous Work

2.1 Software Architecture (SA)

The previous section captures some high level functions, qualities, and characteristics of a large class of systems – RECWEB. The problems identified with RECWEB involve managing the complexity of relationships between functionality and qualities such as availability, modifiability, and performance. Section 2.1.1 provides a basic description of SA and its role in the life cycle of a software system. Section 2.1.2 describes how functionality and qualities are orthogonal. Section 2.1.3 describes tactics used to achieve individual qualities. Section 2.1.4 describes trade offs that must be made to achieve different sets of qualities.

2.1.1 Definition

[10] provides the following definition for SA:

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally

visible properties of those elements, and the relationships among them.

The role SA plays in the life cycle of a software system changes over time. At the outset, SA consists of design decisions made to achieve a set of functional and quality requirements driven by business stakeholders. Decisions are documented and serve as a vehicle of communication among all stakeholders. For example, implementers of the system can refer to quality requirements to understand rationale behind design decisions. The same is true for maintainers who also face requirements change over time. The SA serves as a transferable abstraction of the system which can be used to facilitate any changes.

2.1.2 Functionality

Functionality is the initial driver for building a system. A business problem exists that must be solved by functionality provided by a software system. Increased revenue and/or cost savings are the goal.

There are many ways to achieve any particular function. For example, three snippets of code in Figure 2.1 all perform sorting functions. Each program uses a different algorithm, but all are functionally equivalent. So which is 'better'? Well that depends on the criteria for 'better'. It depends on the metrics that are provided by a business stakeholder.

A quality such as performance might be the criteria. One system might use algorithm that computes the same function faster. Another algorithm might do more to handle faults making it more available. Yet another system might not compile code until runtime allowing the code to be changed without rebuilding the entire system. Qualities are used to determine what is 'better'.

Figure 2.1: Functionality Example: Sorting Algorithms

```
procedure bubbleSort( A : list of sortable items ) defined as:  
  do  
    swapped := false  
    for each i in 0 to length( A ) - 2 do:  
      if A[ i ] > A[ i + 1 ] then  
        swap( A[ i ], A[ i + 1 ] )  
        swapped := true  
      end if  
    end for  
  while swapped  
end procedure
```

(a) Bubble

```
function quicksort(array)  
  var list less, equal, greater  
  if length(array) = 1  
    return array  
  select a pivot value pivot from array  
  for each x in array  
    if x < pivot then append x to less  
    if x = pivot then append x to equal  
    if x > pivot then append x to greater  
  return concatenate(quicksort(less), equal, quicksort(greater))
```

(b) Quick

```
procedure cocktailSort( A : list of sortable items ) defined as:  
  do  
    swapped := false  
    for each i in 0 to length( A ) - 2 do:  
      if A[ i ] > A[ i + 1 ] then // test whether the two elements are in the wrong order  
        swap( A[ i ], A[ i + 1 ] ) // let the two elements change places  
        swapped := true  
      end if  
    end for  
    if swapped = false then  
      // we can exit the outer loop here if no swaps occurred.  
      break do-while loop  
    end if  
    swapped := false  
    for each i in length( A ) - 2 to 0 do:  
      if A[ i ] > A[ i + 1 ] then  
        swap( A[ i ], A[ i + 1 ] )  
        swapped := true  
      end if  
    end for  
  while swapped // if no elements have been swapped, then the list is sorted  
end procedure
```

(c) Cocktail

2.1.3 Qualities

Quality attribute driven design involves defining all of the quality attribute scenarios then applying well known *tactics* that achieve specific qualities [93,94]. Table 1.1 describes how quality requirements can be captured in scenarios. The technique was used to capture the core problem for this thesis in Figures 1.4 and 1.5. Like functionality, there are many options available to achieve the same quality. The purpose of this section is to describe the most common qualities and tactics for achieving them.

Availability

Consider a BBQ grill that runs on a propane tank. The propane provides the energy to make a flame that will cook food. Once the propane runs out, the grill is unusable. The best case scenario is the propane runs out after all the food for one meal is cooked. The worst case scenario is the propane runs out just minutes into cooking a meal. At this point, the propane tank must be refilled to cook food on the grill. Otherwise, the meal must be cooked using alternative means such as the oven or microwave.

If the grill is seen analogous to a software system, then propane tank is a component that has 'failed'. The consequence is the grill is unavailable for cooking. At some point, the amount of propane in the tank became too small for cooking another meal. This point would be considered a 'fault' in a software system. If not corrected, a failure ultimately results.

Some newer grills now have a detection system that identifies when the 'fault' occurs. An indication is given when the weight of the propane tank reaches a certain threshold. This allows the user to know when the propane tank needs to be refilled. A grill without such a detection mechanism might require the user to keep a second propane tank filled and ready when the first one runs out.

The grill being unavailable due to a failure is an example of downtime. An example of the former is when a DNS server goes down resulting in web browsers unable to find a website. The DNS server is responsible for resolving a domain name (i.e. `www.recweb.com`) into an IP address. Without this resolution, the web browser cannot download content from the website. To prevent this problem, DNS servers are redundant so when one is unavailable, another can provide the resolution service.

In the author's experience, failures also occur inside of a website's core software system. For example, all content might be generated dynamically from a database. Problems connecting with the database result in no content retrieved. Typically, the site displays a "We're sorry..site unavailable" message to the user. As with the grill and DNS, a spare database or redundancy is used.

Scheduled downtime is often required to perform maintenance or deploy new enhancements to a software system. A failure isn't the cause of the downtime, but the system is still unavailable. Our experience is scheduled downtime occurs every couple of weeks and during off peak hours (i.e. 3AM). In other instances where a RECWEB was deployed to an international audience, there was less of a window of off peak because of global time zone differences. This created an even greater challenge for availability.

In some cases, a user observes a failure long after it occurs. In other words, a system never is never really seen 'down', but it does not operate as expected. For example, an online shopper observes a large RECWEB offering television \$400. It is unusual in that similar televisions with the same specs were over twice as expensive in other places. Seeing a great deal, the online shopper notifies several friends and colleagues who also order the television. However, weeks later the online shoppers are notified that the offered price was a content error on the website. The individual responsible for entering prices on the website mistyped "\$400" instead of "\$1400". Prior to checkout, the website should verify with the order management system that all prices are correct. However, at the

particular moment the televisions are ordered the connection to the order management system could be down. The system is programmed to log the fault but proceed to take the order. This is based on optimism that the prices are usually correct. By the time the log is detected, many people might take advantage of the great deal. The point is the system is never really 'down' per se. It just does not handle a fault, thus causing a failure to deliver specified functions. This example is actually based on a real world experience of the author.

Formally, availability is the probability a system will be operational when needed [10]. Operational means it performs functions to specifications masking faults before they become failures. The aforementioned examples note tactics such as keeping a spare and using redundancy as viable for achieving higher availability. Following is a description of some other tactics.

Keeping a system available starts with detecting all faults that could lead to failure. One tactic is exception handling. Most modern programming languages provide a facility to identify blocks of code that might cause a fault. When a fault in that block of code occurs, the programmer specifies an alternative path in code to hopefully recover and prevent the fault from becoming a failure.

Other tactics are designed for when one component is critically dependent on another. The previous example describes the dynamic content generation component being dependent on a database connection. In this situation, the content component could issue a 'ping' to the database and wait for an 'echo' or response from the database indicating it is up and ready for service. If no echo is received the content component can send out alerts so any problems with database can be identified and fixed immediately. The actual 'ping' might even trigger a test transaction on a the database to assure not only is the database up, but it is functioning to a certain degree.

Similar to ping/echo is the heartbeat tactic. This would mean the database server

sends a message out on a given interval to all components interested in the database being 'alive'. When a component fails to receive the message, it can assume that something might be wrong with the database server and act accordingly.

Redundancy is used to recover from faults. Basically it is when there is more than one instance of a component available should a fault occur. In BBQ grill example, a spare propane tank was used. The spare tank merely needs to be plugged in to restart cooking. However, software systems often require some synchronization to put the spare component in the last working state of the failed component. Synchronization is also required in cases where the extra instances of the component are online and might even be taking turns servicing requests—a concept called load balancing. Ultimately, redundancy is about keeping all parallel instances of a component in a consistent and working state. This requires rolling back and synchronizing all components as needed.

In some cases where greater availability might be required, the same request for service might be sent to several instances of the same components. The requester might just use the first response it gets, or it might compare the responses to ensure they are the same. This tactic of redundancy is called voting.

Preventing faults from occurring in the first place is another goal of availability. Any component that generates one or many faults might have to be taken out of service. Similar to exceptions, transactions are another option. A transaction is a set of processing steps that are treated as one atomic operation. This means every step must succeed or the state of the system should be rolled back to what it was before the transaction started. Many modern programming languages also support this feature.

Process monitoring can also be used to increase availability. For example, if a certain component appears to contain a memory leak a monitor can alert engineers when the memory reaches a certain threshold. The engineer can then restart the process to clear some memory. Obviously, fixing the memory leak in the code would be ideal. However,

if resources are not available to do so immediately, a process monitor can insure the leak never occurs in operation.

Performance

Consider a trip to the grocery store. For most, it is a necessary task in life that consumes free time. A task that one might try to minimize the amount of time spent. The amount of time spent depends on a variety of aspects of the the store.

The experience starts upon pulling into parking lot. Some stores might have better parking access. Upon arriving to one store with a full lot, one might decide it is faster to drive to another store down the street.

Inside the store, several places might add or subtract time to the task. For example, getting meat from the butcher might take a while if only one butcher is working the counter. Or perhaps, the store offers other time saving functions like a pharmacy or dry cleaners. However, these functions might also add time. For example, multiple pharmacists concurrently filling prescriptions should lead to faster service. Similarly, more cashiers concurrently checking people out leads to faster service.

The grocery store is an example of concurrency that leads to better performance. Performance is about time and efficiency of allowing people to complete the task of grocery shopping. The same methods of concurrency are used on a RECWEB. During peak shopping season, extra web servers are added to concurrently handle more requests from users for service.

Formally, performance is how much time a system takes to respond to events. Ultimately, event can arrive at different rates and require different resource usage. For example, some people visit a grocery store just to pick a prescription and not buy any groceries. By the same token, some people log on to a website just to update their account information, but do not actually add products to their cart and checkout. Controlling

performance comes down to how all resources are managed as a whole to provide the fastest service to every user.

The moment an event arrives at a system there is a demand for some resource in the system. In fact, over the course of processing the request several resources might come into play; each with a specific task to perform. This leads to some resources being blocked as multiple other resources might depend on it. Again, the database server a resource highly contended for by components.

Section 2.1.2 provides an example of how choice of algorithm can increase the speed of computing a given function. Thus, changing the algorithm is one tactic for increasing performance. In fact, refactoring code is one of the most popular and effective ways to increase performance.

The rate at which events are allowed to be directed to other resources can also be controlled. For example, a RECWEB might collect all the information from a user to process an order and provide the end user with an 'Order successful!' message. However, the order might still be waiting in a queue to be submitted to the back end fulfillment system. Performance is increased on the back end fulfillment system because orders can be fed in for processing at the most optimal rate. The front end application also performs better because it can respond with a success message faster.

Concurrency is a very widely used method for increasing performance. The pharmacist and cashier in the grocery store example provide a basic understanding. In a software system, each additional cashier or pharmacist would be accomplished as an additional thread. The thread processes events in parallel, thus handling multiple requests simultaneously. HTTP servers are a great example of how concurrency works.

Caching data is another common tactic used on RECWEB to increase performance. Multiple copies of content can be distributed across several servers. Demands for the resource can then be satisfied by one of copies. Furthermore, access to the copies can

be optimized. The assumption is when the content becomes stale or out of date, it must be updated by some mechanism. The tactic is well researched and implemented for several RECWEB.

Of course, one rudimentary tactic for increasing performance is adding hardware resources such as additional servers or memory. In some cases, the situation demands it where a system just out grows the initial hardware allocated. In other cases, the tactic is really just a band aid on poorly designed software. In either case, hardware resources come cost money so another tactic might be preferable.

Modifiability

Consider when it comes time to spend money on entertainment. For example, a person would like to go see a baseball game but do not have a ticket. You show up at the ticket booth to find that all 40,000 seats are sold. However, some concourses throughout the stadium can be sectioned off to accommodate an additional 2000 people. Individuals can pay to stand in this space and watch the game.

The stadium is analogous to a software system. At the moment the last of the 40K tickets were sold the stadium was modified to accommodate more people. It is an example of *late binding*. The concourse is available to function as either a walkway or a standing place for viewing the game. But the time it is bound to that function isn't until game time when the number of tickets sold is determined.

Formally, modifiability is about cost of changing a software system [10]. Tactics fall into 3 categories: localize changes, prevent ripple effect, and defer binding time. Because the focus of this thesis is on modifiability, Section 2.2 discusses many modifiability tactics already applied to RECWEB. Figure 2.2 provides specific examples.

Other

Availability, performance, and modifiability are most relevant to this work. However, many other quality attributes exist. We briefly describe a few here, but make no further mention unless a specific part of the solution significantly decreases one of these qualities. In this case, it is noted as a necessary trade off. The idea of trading off one quality for another is covered in the next section.

Security has always been a big concern for RECWEB because lots of private personal information is exchanged (i.e. address, credit card numbers, etc.). There have been attacks that compromised this information for many users. However, most of the attacks come through a hole in the corporate firewall network. Even when careless password policies are in place, we've seen high traffic sites remain fairly secure. For example, one RECWEB the author worked on kept the same root password for the database and all application servers for over 2 years! Furthermore, many developers even had access to this password. Even so, there was never an attack. In general, the front end web application architectures handle security well.

Testability is somewhat an issue but low priority. Testing web applications can be hard with hyperlinks constantly changing. But in the author's experience, most RECWEB organizations have access to the latest testing tools and built in monitors often come with the off the shelf packages they are built upon. There are always a certain amount of code defects to address, but this is the case with any software system. Lack of time for testing is an issue especially during P. Any increase in testability would be welcome, but status quo is accepted by most organizations the author has dealt with.

Usability is a moving target on most of the projects the author has worked on. Organizations are constantly trying to figure out a better user interface. Some of this is driven by web user interfaces being limited by web browser technology as compared to

traditional desktop applications which have a richer set of components to build upon. But there is a whole movement of research trying to solve the problem of making web applications richer user interfaces. In addition, dozens if no hundreds of web application frameworks have been produced to streamline the most common parts of web user interfaces. For example, form validation used to be rewritten for every new project. Web frameworks now provide this in a reusable component that is easily configurable.

2.1.4 Trade Offs

If only one quality were desired, then the previous sections provide ample examples of tactics to achieve that quality. However, multiple qualities are often desired. It is impossible to have them all because they impact each other. For example, one might use the tactic *insert and intermediary* to increase modifiability. The intermediary is a new resource that might require other resources. This can affect both availability and performance. The core problem addressed by this research is another example. The RECWEB scenarios described Section 1.2 show a trade off of modifiability for performance and availability. The extent that one quality affects another must be measured against an acceptable criteria. Perhaps the performance hit is hardly noticeable and the value of increased modifiability far outweighs the cost. The key point is trade offs are required. Following are a couple more examples from the author's experience.

One organization experienced very high traffic on their website during a holiday season. Performance of the site was very slow such that most site visitors were struggling to navigate products, add them to their cart, and checkout. The cause was expected traffic for the holiday was underestimated by a factor of 4. The amount of network bandwidth available from their Internet Service Provider(ISP) was not enough. Unable to increase the bandwidth in the short term, an executive decision was made to 'throttle' access to

the site; an example of the *manage event rate* tactic for performance. The result was a trade off of availability for performance. Users that did gain access to the site, experienced high performance while other users were blocked from the site. The rationale was a smaller set of users able to shop with ease would lead to more sales, as opposed to a large number of users struggling to move from page to page.

In another example, an organization traded modifiability for performance. A module responsible for sending out email marketing materials to millions of users was not performing well. The decision was made to rewrite the module in C++ because of the performance advantage over existing Java-based code. A highly experienced and skilled programmer was brought in to complete the work. The resulting module has exceeded performance requirements and been in production for several years. But the complexity of C++ and the algorithms used in the module make it difficult for any average programmers on staff to make changes. Several changes have been rejected because of the cost of bringing in a higher skilled programmer to change the module.

In a preceding availability example, televisions were priced wrong leading to several invalid purchases. A business trade off was made for an increase in availability. Pricing on the website is right a high percentage of the time. A business decided the loss of sales by denying orders is more costly then correcting the pricing mistake once in a while.

The main point is qualities can be judged only in the context of where they are needed. In the previous example, the highly skilled programmer was only directed to solve the performance problem. He or she was not concerned with modifiability. Therefore, in order to select the right tactics for achieving quality attributes, all quality scenarios must be captured in context. In any case, trade offs will have to be made and qualities prioritized. Figure 2.2 shows how rapidly complex this decision can become. The multitude of options across both functionality and quality makes designing, implementing, and maintaining software systems a complex task. Figure 2.2 illustrates how fast complexity can

Figure 2.2: Tactics

Qualities	Rules													
	1	2	3	4	5	6	7	8	9	10	11	64	
Availability	Y	N	N	N	N	N	Y	N	N	Y	Y	N	
Modifiability	N	Y	N	N	N	N	Y	N	N	N	Y	Y	
Performance	N	N	Y	N	N	N	N	Y	Y	N	Y	Y	
Security	N	N	N	Y	N	N	N	Y	N	N	N	Y	
Testability	N	N	N	N	Y	N	N	N	N	Y	N	N	
Usability	N	N	N	N	N	Y	N	N	Y	Y	N	N	
Tactics														
Ping / Echo / Heartbeat	✓						✓			✓	✓			
Exception Handling	✓						✓			✓	✓			
Voting	✓						✓			✓	✓			
Active / Passive Redundancy	✓						✓			✓	✓			
Spare	✓						✓			✓	✓			
Shadow	✓						✓			✓	✓			
State Resynch	✓						✓			✓	✓			
Rollback	✓						✓			✓	✓			
Removal from Service	✓						✓			✓	✓			
Transactions	✓						✓			✓	✓			
Process Monitor	✓						✓			✓	✓			
Semantic Coherence		✓					✓				✓		✓	
Generalize Module		✓					✓				✓		✓	
Limit Possible Options		✓					✓				✓		✓	
Abstract Common Services		✓					✓				✓		✓	
Hide Information		✓					✓				✓		✓	
Maintain Existing Interface		✓					✓				✓		✓	
Restrict Communication Paths		✓					✓				✓		✓	
Use an Intermediary		✓					✓				✓		✓	
Runtime Registration		✓					✓				✓		✓	
Configuration Files		✓					✓				✓		✓	
Polymorphism		✓					✓				✓		✓	
Component Replacement		✓					✓				✓		✓	
Adherence to Defined Protocols		✓					✓				✓		✓	
Increase Computation Efficiency			✓					✓	✓		✓		✓	
Manage Event Rate			✓					✓	✓		✓		✓	
Control Frequency Sampling			✓					✓	✓		✓		✓	
Introduce Concurrency			✓					✓	✓		✓		✓	
Maintain Multiple Copies			✓					✓	✓		✓		✓	
Increase Available Resources			✓					✓	✓		✓		✓	
Scheduling Policy			✓					✓	✓		✓		✓	
Authenticate Users				✓				✓					✓	
Authorize Users				✓				✓					✓	
Maintain Data Confidentiality				✓				✓					✓	
Maintain Integrity				✓				✓					✓	
Limit Access and Exposure				✓				✓					✓	
Intrusion Detection				✓				✓					✓	
Audit Trail				✓				✓					✓	
Record / Playback			24		✓					✓				
Separate Interface from Implementation					✓					✓				
Specialized Access Routines/Interfaces					✓					✓				
Built-in Monitors					✓					✓				
Cancel / Undo / Aggregate						✓			✓	✓				



(a) Everyday Logo



(b) Thanksgiving Logo

Figure 2.3: Google™ Holiday Transition

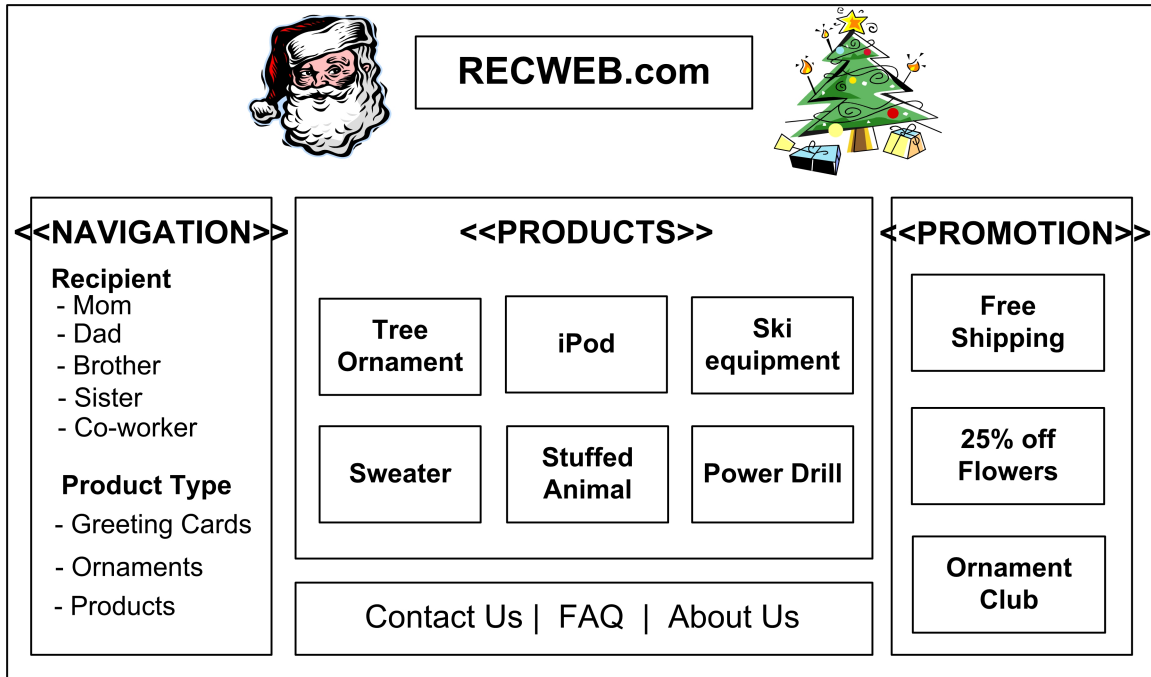
grow when just considering qualities let alone functionality. It shows six possible qualities which can either be important or not. This yields 2^n or 64 possibilities. Realistically, a quality should be assigned a level of importance rather than just an all or nothing proposition. Even so, that would make the decision table even more complex. Furthermore, the decision table doesn't even reflect that the adverse effect that one quality has on another.

2.2 Typical RECWEB Modifiability

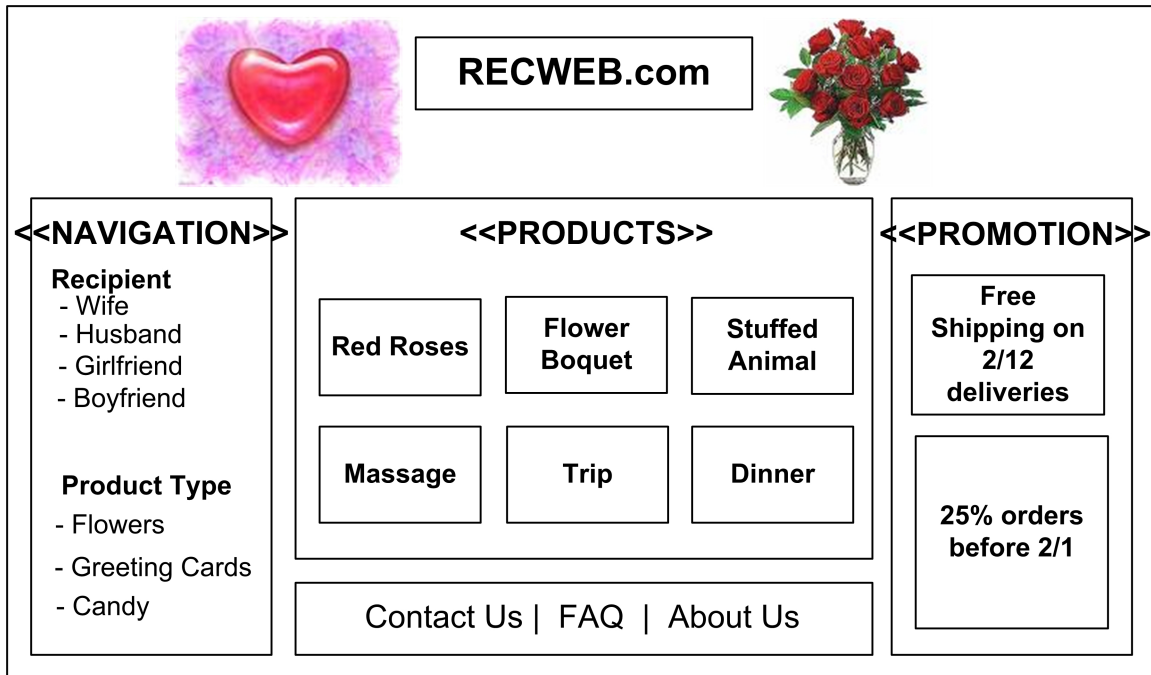
Consider the Google™ home page. Figure 2.3a shows the everyday logo, whereas Figure 2.3b shows a holiday logo. When a holiday occurs, making this change simply involves creating and moving a new logo file.

A RECWEB home page, on the other hand, involves many changes (See Figure 2.4). The content for logos, navigation, products, and promotions have to be changed. These changes can involve updating and moving many files and databases. Figure 2.5 captures this in a modifiability scenario.

This section describes parts of a typical RECWEB architecture that relate to modifiability. The focus is on the Valentine's Day Scenario. The RECWEB architecture applies well known modifiability tactics. The existing tactics do not provide sufficient



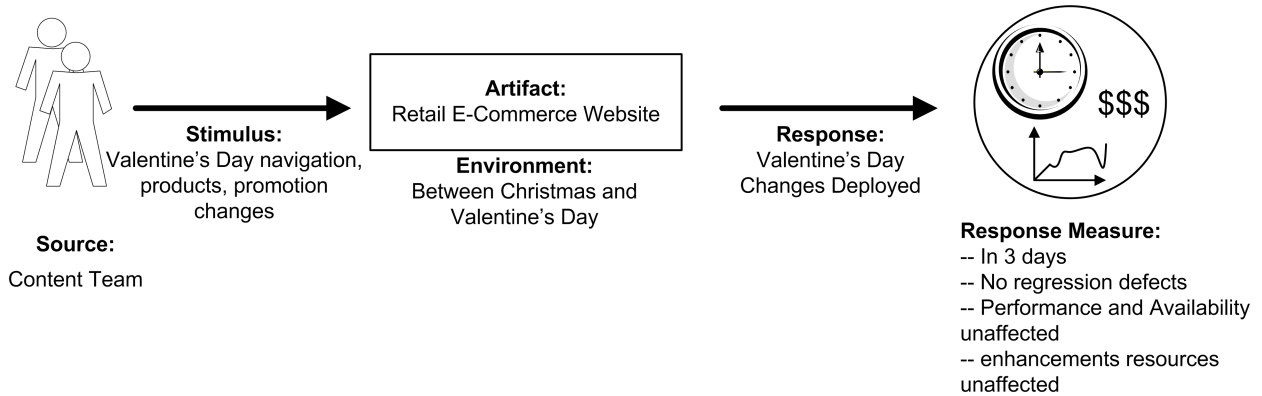
(a) Christmas Day Homepage



(b) Valentine's Day Homepage

Figure 2.4: RECWEB Holiday Transition

Figure 2.5: Valentine's Day Scenario

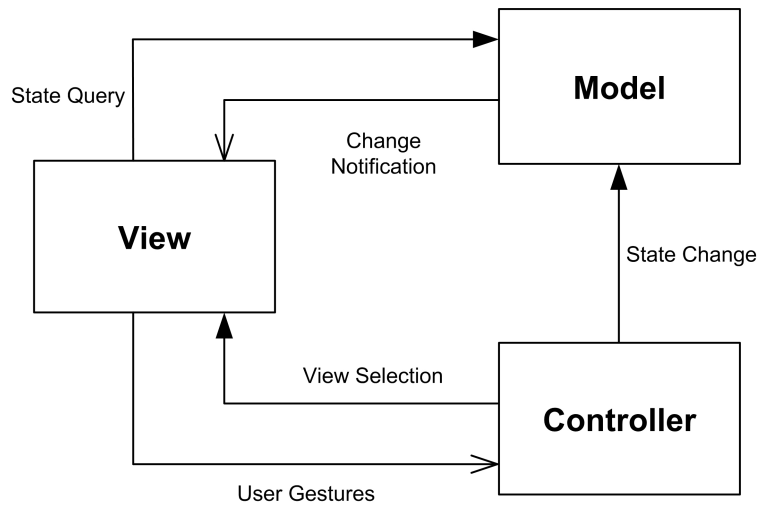


modifiability because they fail to scale during a RECWEB holiday transition.

Most RECWEB are implemented by modifying a commercial off the shelf (COTS) software package [84]. Modifying COTS systems can be challenging [9, 14, 34, 90]. Deviating too far from the original package makes upgrades to the COTS package near impossible [66]. In one project the author participated, the COTS package was four versions obsolete. In fact, the COTS vendor had discontinued support for the version running in production. The client was forced to replace the entire system with a new COTS package and a different architecture. Using a COTS package limits modifiability from the beginning and continues to deteriorate over time [57, 62]. Although, most RECWEB organizations still opt for COTS because most of the required functionality is available out of the box [8, 37, 65].

RECWEB COTS packages typically implement some flavor of the model-view-controller (MVC) architectural pattern (See Figure 2.6) [3]. The goal is a 'separation of concerns' so that changes do not cause ripple effects [30]. Essentially, MVC is the strategy that RECWEB use to achieve modifiability. The following are the three main "concerns" in MVC [1, 43, 80]:

Figure 2.6: MVC Structure [1]



Model The primary responsibility is to handle state changes from controllers and communicate state changes to views. The application state is a large set of data represented in a way that makes sense in the real-world. The model is the the means to manipulate this data.

View The primary responsibility is to render the state from the model and report to controllers any significant user interaction with the state. The view has the option of either explicitly requesting state changes from the model or being notified by the model when state has changed. The view is the means to providing a user interface to the model.

Controller The primary responsibility is to process user gestures from views, report any state changes to the model, and select the appropriate view in response. User gestures might be requests for navigating to a different part of the model or updating the model. Reporting state changes to the model is accomplished through the interface the model provides. Selecting the appropriate view for response is based

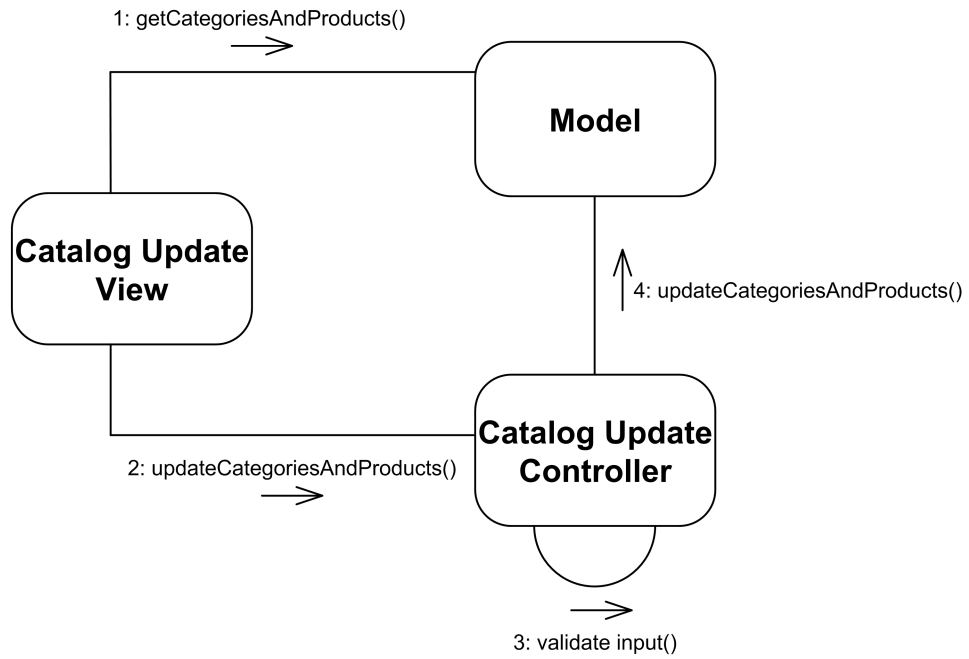
on the result of the model interaction or the initial user gesture.

The first issue for Valentine's Day is model data must be updated by some entity. Update controllers and views (Figure 2.8a) are one option. External entities, for example, data migration tools could also be used to update the data (Figure 2.8b). Most of the time a combination of the two options are made available depending on the scale of the updates. If large amounts of data are being updated, content personnel make updates to a staging database in advance. The updates are propagated to the production database when the holiday transition occurs. There are two major issues with this technique. First, the propagation often relies on proprietary features of relational database management systems. Second, referential integrity of the data is at risk when imported into the production system.

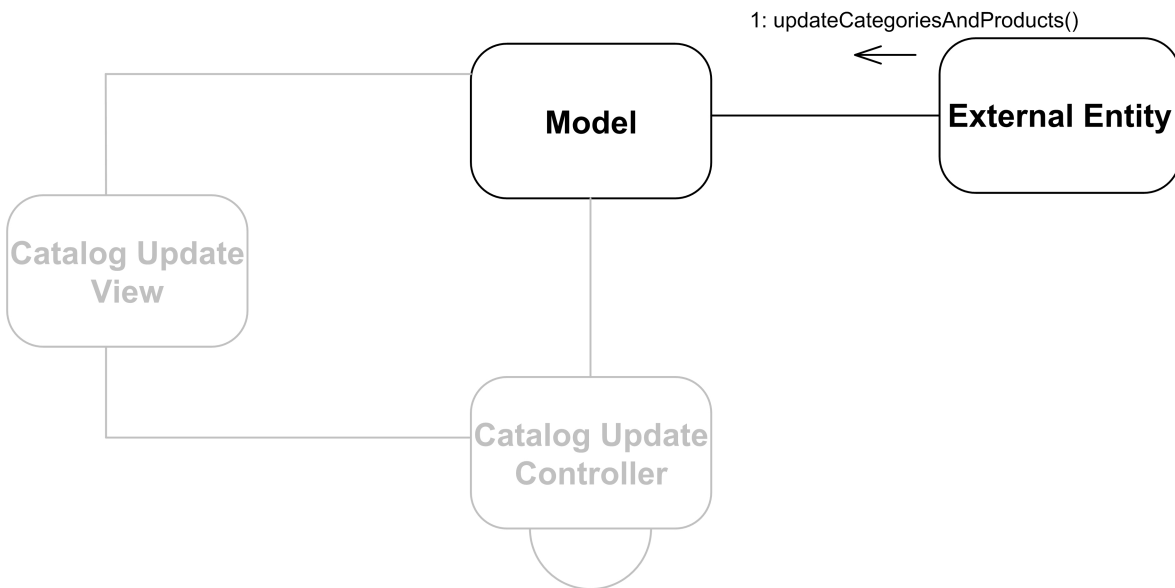
The second issue for Valentine's Day is updating views (i.e. home page) with new model data. Figure 2.9b shows how the views can *pull* the data. In this case, the controller selects a Valentine's Day specific home page view. The view is programmed to pull the right data from the model. The downside is multiple home page views have to be maintained. The other option shown in Figure 2.9a is to have one generic home page view that assumes data *push* to it from the model is correct. This view is only concerned with the rendering of the data. The downside is the model might require some knowledge of the view, for example, a "Home Page" category. This violates the separation of concerns of MVC. In addition, duplicate data such as products in multiple categories can make larger sets of data harder to manage. Both options be reasonably effective, but only with a lot of manual human interaction.

Updating model data is the common trigger for the aforementioned issues. Figure 2.9 shows typical entities in a RECWEB model. For a large RECWEB there could be thousands of products and hundreds of categories. In Figure 2.1 shows even a small

Figure 2.7: Model Updates

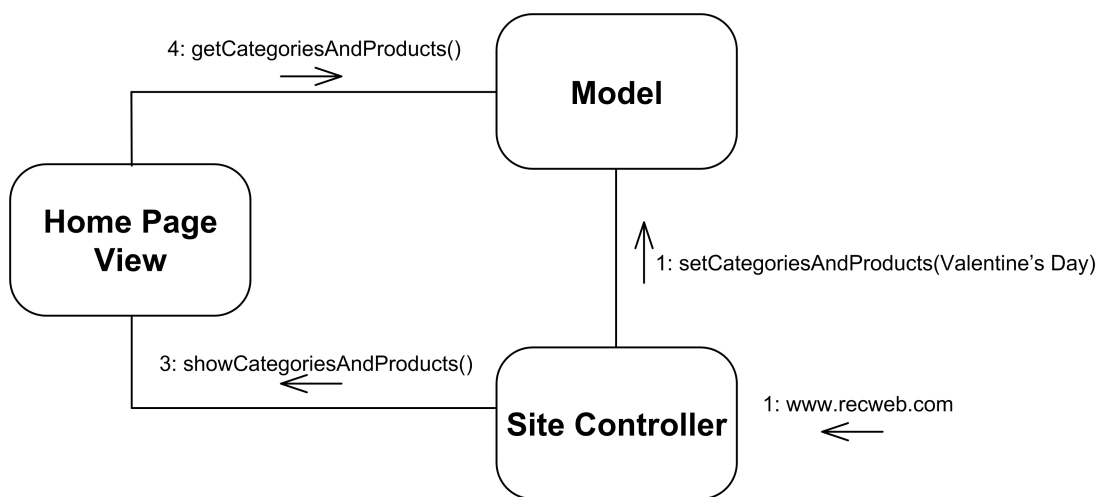


(a) MVC Update

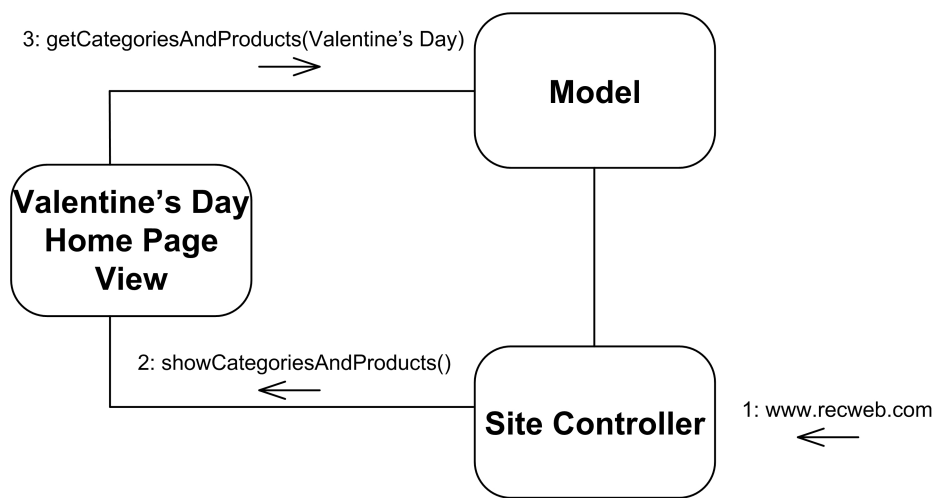


(b) External Update

Figure 2.8: View Updates

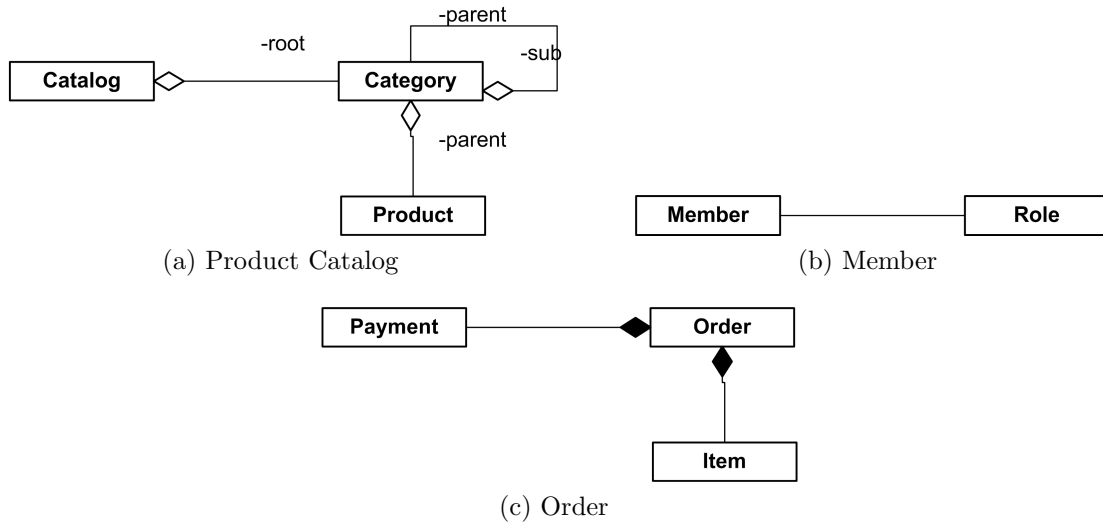


(a) Push



(b) Pull

Figure 2.9: RECWEB Model



subset of data can require many changes. On the flip side, promotions require less data changes, but across more parts of the model. For example, Figure 2.2e shows 'ornament club' as a role. Catalog or product entities can be filtered based on role. This functionality is often referred to as "personalization" [7,81]. Order data is also affected by membership roles. In the author's experience, catalog changes have been very difficult to manage, whereas promotions are fairly easy to deploy. However, promotions have also failed to expire because someone forgets to remove them from the model.

The model is fairly generic to accommodate the greatest flexibility across different retail domains. After working in several different domains, this abstraction is fine for parts like order management and membership. However, parts like product catalog always seem to require extending or stretching the generic model. For example, 'Greeting Cards', 'Product Type', and 'Wife' are all different, but nonetheless are treated as categories. Any special treatment of a category either requires extending the model or special handling in the business logic of the model. One might argue this is a reason NOT to use COTS.

Table 2.1: RECWEB Sample Tuples

ID	Name	Parent ID	Roles
1	Recipient	null	standard
2	Product Type	null	standard
3	Mom	1	standard
4	Brother	1	standard
5	Co-worker	1	standard
6	Wife	1	standard
7	Boyfriend	1	standard
8	Daughter	1	standard
9	Grandpa	1	standard
10	Greeting Cards	2	standard
11	Ornaments	2	ornament club
12	Electronics	2	standard
13	Flowers	2	standard
14	Sporting Goods	2	standard
15	Tools	2	standard

(a) Categories

ID	Name	Roles
1	Christmas Tree Ornament	ornament club
2	iPod	standard
3	Ski Equipment	standard
4	Sweater	standard
5	Red Roses	standard
6	Flower Bouquet	standard
7	Stuffed Animal	standard
8	Massage	standard
9	Trip	standard
10	Power drill	standard
11	Dinner	standard
12	Golf Clubs	standard

(b) Products

Category ID	Product ID
13	5
13	6
5	8
5	11
6	4
6	5
6	6
6	8
7	2
7	12
7	3
9	12
14	12
11	1
3	1

(c) Category Products

Username	Password
kathyjohns	*****
marksoenen	*****
anneblanco	*****

(d) Members

Username	Role
kathyjohns	standard
kathyjohns	ornament club
marksoenen	standard
annblanco	standard

(e) Roles

However, this work is not intended to contribute a ‘build versus buy’ [28, 51] debate. Rather, this work strives to find ways to increase modifiability in the presence of a generic model.

Table 2.2 shows many common modifiability tactics [10] are present in the RECWEB architecture. The problem with the existing tactics is the assumption there is enough time to make changes. Effort to make a single change might be small. But a change must also be tested and deployed. In the case of RECWEB there could be hundreds or thousands of the small changes. The modifiability tactics simply do not scale.

2.3 Intelligent Agents

Using software to perform activities in place of humans is not new [11, 24, 27, 39, 48, 54, 58, 61, 63, 67, 86, 97]. For example, many software configuration management activities such as building and deploying software are automated [22, 31, 32, 38, 42, 68, 69, 95]. Software quality assurance activities such as testing and performance monitoring are also automated [33, 53]. The result can be lower costs and reduction of human labor. In the aforementioned examples, the software is acting as and *agent*.

Using software to make RECWEB holiday content changes in place of humans is more difficult. First, changes are not as well defined. The changes might be based on objective or subjective data provided by a human. Second, there is not enough time to make the changes. The amount of changes multiplied by the time to make each change exceeds the amount of time available between holidays. In order to automate RECWEB holiday content changes, an *intelligent agent* (IA) must be used.

An IA is a “software tool that carries out a task on behalf of a user or computer, typically relatively autonomously” [40]. A simple IA perceives conditions in the current environment and reacts with an action. For example, a RECWEB IA has to sense

Table 2.2: Modifiability Tactics in RECWEB

Localize Changes	
Semantic Coherence	The model, view, and controller are each responsible for a unique concern. Each concern functions without excessive reliance on the other. For example, should the model become unavailable, the view can still present data and support user interaction.
Anticipate Expected Changes	Selecting MVC as the architectural pattern for RECWEB is anticipating changes. We showed how changes to the model are often made to a staging database in advance.
Generalize the Module	We showed how a generic model can allow multiple domains to use the model. We also showed how a generic home page can capture the common user interface elements that do not change, while allowing other entities such as the controller to ensure the model is updated.
Prevent Ripple Effects	
Hide Information	The controller need not worry about the exact details of updating data in the model. On the flip side, the model has little knowledge of how the view goes about presenting data.
Restrict Communication Paths	The view can only communicate with the model in a read-only fashion. Any logic for updating the model must go through the controller. Furthermore, the controller must use the interface the model provides.
Use an Intermediary	The controller is an intermediary. It decouples access to the model and data from presentation.
Defer Binding Time	
Configuration Files	The intelligence in a Valentine's Day specific view can be controlled by a configuration file. For example, the exact state queries made to the model could be changed in a configuration file as opposed to source code.

what holiday season is active and locate content that is relevant. More complex agents are able to reason using outside knowledge or act according to desired goals. Finally, highly complex agents learn and adapt to changing environments becoming increasingly more intelligent over time. Although, the complexity of agents increases the difficulty of implementing them. RECWEB can start with a simple to moderately complex IA.

The study of intelligent agents is also known as Artificial Intelligence(AI). AI is a field with a vast amount of research results. However, AI has been slow to emerge in mainstream application development. A full discussion of AI would be out of scope because not all parts are relevant to the RECWEB modifiability problem. For example, motion and manipulation of objects is a type of intelligence required for robotics. Whereas, knowledge representation, model theory, and logic are more relevant to RECWEB. These parts of AI contributed heavily to the Semantic Web which is discussed in the next section. To avoid redundancy, this section only includes key aspects of AI.

2.4 Semantic Web

The Semantic Web, a project of the World Wide Web Consortium (W3C), makes data on the World Wide Web(WWW) available for automated processing by machines, for example intelligent agents [76,85]. Therefore, if RECWEB holiday content can adapted to use the Semantic Web, intelligent agents can handle content changes during holiday seasons. The previous section introduced intelligent agents as a part of AI. Semantic Web is often referred to as a 'playground' for AI. This section provides a background on the Semantic Web specifications.

Figure 2.10 shows a layered view of the Semantic Web. For the most part, Semantic Web is built on stable Internet technology. For example, Uniform Resource Indicators (URI) became standard during the birth of the World Wide Web(WWW) in the early

1990's. A URI is defined as “a compact sequence of characters that identifies and abstract or physical resource” [12]. URI are used to uniquely identify *resources*; a key concept of the Semantic Web. Access to resources also uses standard Internet technology such as Hypertext Transfer Protocol (HTTP) as well as Secure Sockets Layer (SSL) for encrypted communications.

Resource Description Framework(RDF) is key additional layer and the core of the Semantic Web. Any application of Semantic Web involves using RDF. RDF is about making *statements* about the aforementioned concept of *resources*. As an overview, Semantic Web technology involves an iterative process of:

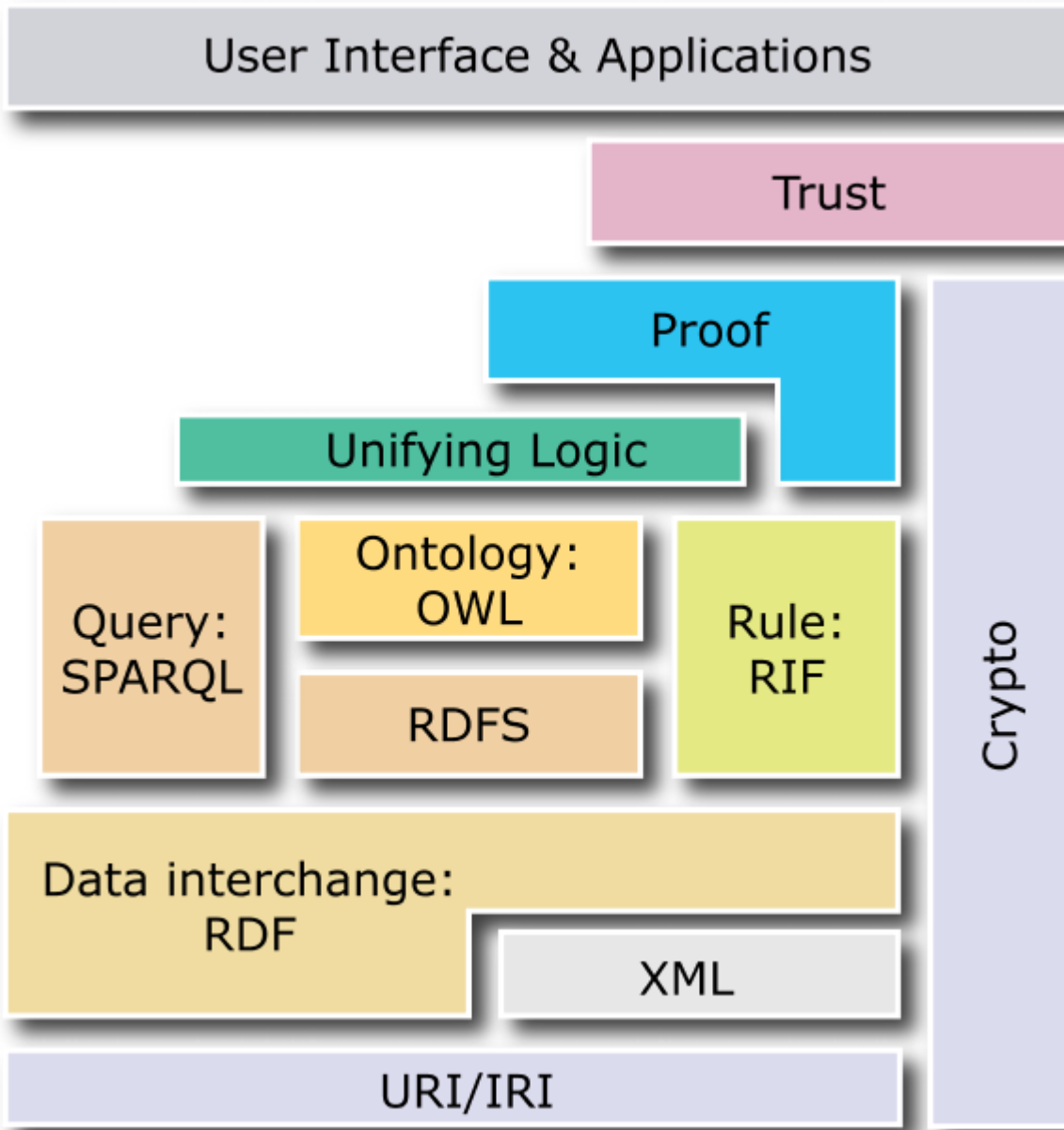
- *Mapping* existing data to RDF
- *Merging* multiple sets of RDF data
- *Querying* RDF data

These are the most basic steps for any application of Semantic Web. Section 2.4.3 describes more details of RDF.

SPARQL Query Language for RDF(SPARQL) and Web Ontology Language(OWL) are the other key layers shown in Figure 2.10. Sections 2.4.5 and 2.4.4 show how these enhance parts of the *map*, *merge*, *query* sequence. The key enhancements are expressiveness, inference, and reasoning about RDF data. These elements are not required for using the Semantic Web. However, the benefits of usage are high especially with respect to RECWEB.

This section begins with two example scenarios. The scenarios are used to describe processes that are typically done manually by humans. Alongside is a discussion on how Semantic Web helps automate these processes. Sections 2.4.3, 2.4.4, and 2.4.5 follow with descriptions of RDF, OWL, and SPARQL. Finally, Section 2.4.6 offers further justification for Semantic Web as an emerging technology with a stable specification.

Figure 2.10: Semantic Web Layercake [23]



2.4.1 Example #1: Buying a House

1. Buyer determines how much they can afford.
2. Buyer finds houses in the price range.
3. Buyer uses different information sources to compare neighborhood development, school districts, property tax, etc.
4. Buyer combines all the information to determine which house is the best fit.

Today, information and functionality is available for every aspect of the home buying process. For example, online applications are available for loan pre-approval. Websites can immediately generate a comprehensive report on the loan. The buyer uses the information to assess the financial burden of the purchase.

Finding houses is also facilitated by the Internet. Search engines can locate houses based on numerous criteria such as geographic location, number of bedrooms, year built, and lake view. The buyer can refine criteria to match their price range. The Internet saves the buyer the time of physically viewing properties that do not meet their criteria.

During the process, a buyer bookmarks websites, creates spreadsheets, downloads pictures, and saves electronic copies of government documents (tax information, urban development, etc.). Organizing the information is done on the computer but still a manual process. Files can be lost. Information can be out of date. The task of organizing the information can add stress to an already stressful process.

2.4.2 Example #2: Buying a Digital Camera

1. Select important features.
2. Find cameras with the features.

3. Read reviews on selected cameras.
4. Pick a camera.
5. Shop for best price and purchase camera.

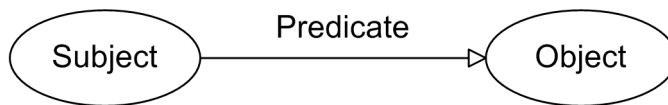
Like shopping for a house, relevant information is accumulated from multiple sources. Each source contributes to the buying decision. For example, a particular camera model might be well advertised and look attractive visually. However, several customer reviews might convey dissatisfaction of the camera performance. Perhaps a particular model lens has better technical specification such as mega pixels. A typical buyer might not understand those terms so they consult an online encyclopedia, for example, Wikipedia. Finally, other online sources tell the shopper the best prices, shipping costs, and service reputations of costs of online stores.

The preceding examples illustrate how Internet users manually build relationships between a set of resources. Relationships are linked together by a unique piece of information common across resources. For example, a camera has a manufacturer and model number. A house for sale has a unique listing number and physical address. A user connects the resources by inputting the information into another web application to get more information. For example, the model number can be used to download the user's manual from the manufacturer website. The physical address can be input into an online map application to show schools and restaurants near a house for sale.

2.4.3 Resource Description Framework (RDF)

The preceding examples show relationships between resources on the WWW exist. Hyperlinks only explicitly capture a relationship between two documents. Humans are able to perceive and process implicit relationships manually. Semantic Web uses RDF to capture knowledge of these relationships explicitly. A concrete representation of RDF can

Figure 2.11: RDF Triple



be read by intelligent agents. This enables the automation of the processes described in the examples in Section 2.4.2 and 2.4.1.

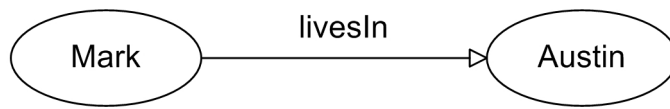
For example, the a web browser could automatically detect a physical address inside a section of text on the web page. The browser could render the text as a hyperlink to a geographical map on another website. Humans normally cut an paste the text into the map website. The website author could recognize the value of making the address a hyperlink to a map website. In both instances, time is saved because the link appears automatically.

The smallest part of RDF is a triple or a labeled connection between two resources [64]. Figure 2.11 illustrates the anatomy of a RDF triple: a subject, predicate, and object. The identity or name of a resource is captured in a string of characters called an Uniform Resource Identifier (URI). A URI must be the value for a subjects and predicates. The object in a triple can be either a URI or a literal string. Figure 2.13a shows an RDF triple capturing the statement 'Mark lives in Austin'. These examples illustrate the abstract data model for RDF.

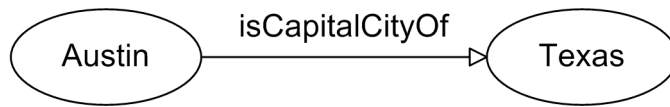
There are several ways to concretely represent RDF triples. Included are:

- RDF/XML
- Turtle
- n3
- RXR

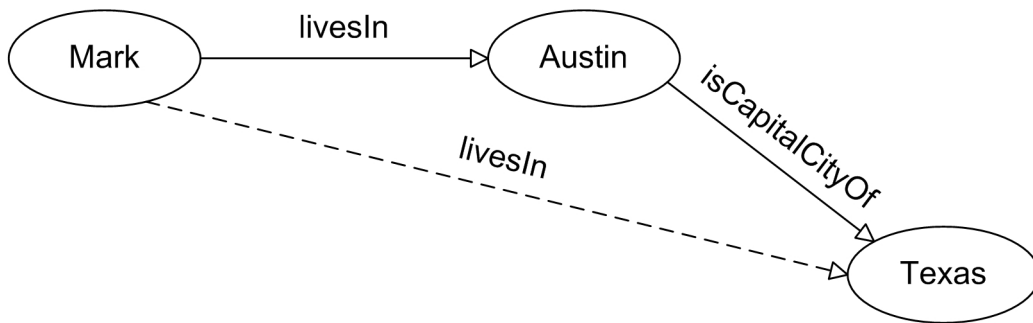
Figure 2.12: Reasoning Example



(a) Statement 1: Mark lives in Austin.



(b) Statement 2: Austin is capital city of Texas.



(c) Inferred Statement: Mark lives in Texas.

Each is just a syntax for a notation. All of these conform to the abstract RDF data model. Machines can read and convert from one format to another. This work uses RDF/XML. For example, Figure 2.1 shows the statement 'Mark lives in Austin' in RDF/XML format. This example shows a simple *mapping* of data to RDF.

Listing 2.1: RDF/XML Example

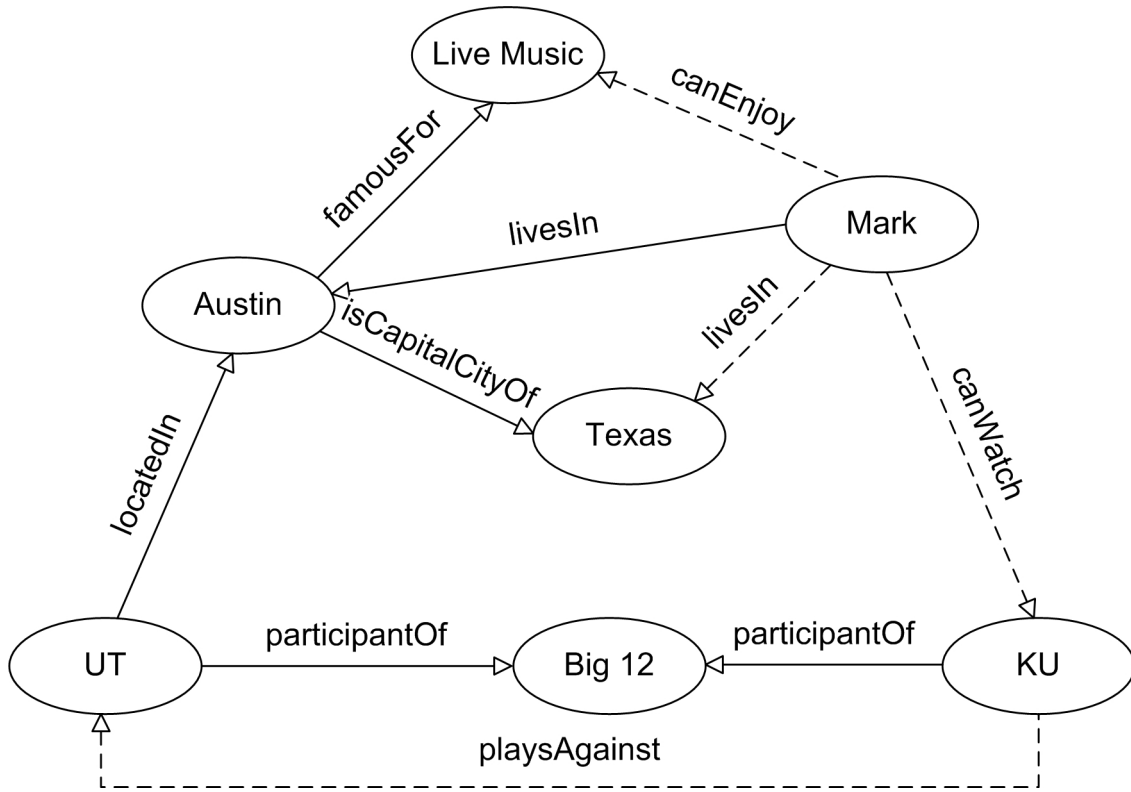
```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:recw="http://www.marksoenen.com/Ontologies/recweb.owl">
  <rdf:Description rdf:about="http://www.marksoenen.com">
    <recw:livesIn>Austin, TX</recw:livesIn>
  </rdf:Description>
</rdf:RDF>
```

A set of triples form a directed graph. Figure 2.13c shows a set of two triples and the corresponding directed graph. This example also illustrates the *merging* of RDF data. Figure 2.13 shows how additional data sets can continuously be merged to form a larger more meaningful directed graph. Again, the graph is something humans often do mentally. RDF enables the graphs to be built automatically by a machine [88].

The preceding examples also illustrate the RDF model allows meta data and data to be mixed together. For example, the statement 'a person must have a last name' is more of a restriction on any resource that is an instance of a person. The instance of a person in the example is 'Mark'. Other statements are used to describe relationships of instance data, for example, 'Mark lives in Austin'. This statement is valid and useful but wasn't explicitly required by another RDF statement. The statement 'a person must have a last name' indicates that another RDF triple is required that says 'Mark has the last name Soenen'.

The level of expressiveness also varies among RDF statements. The statement of a person's name doesn't serve much but as an identifying attribute of the instance of the

Figure 2.13: Continuous Merging

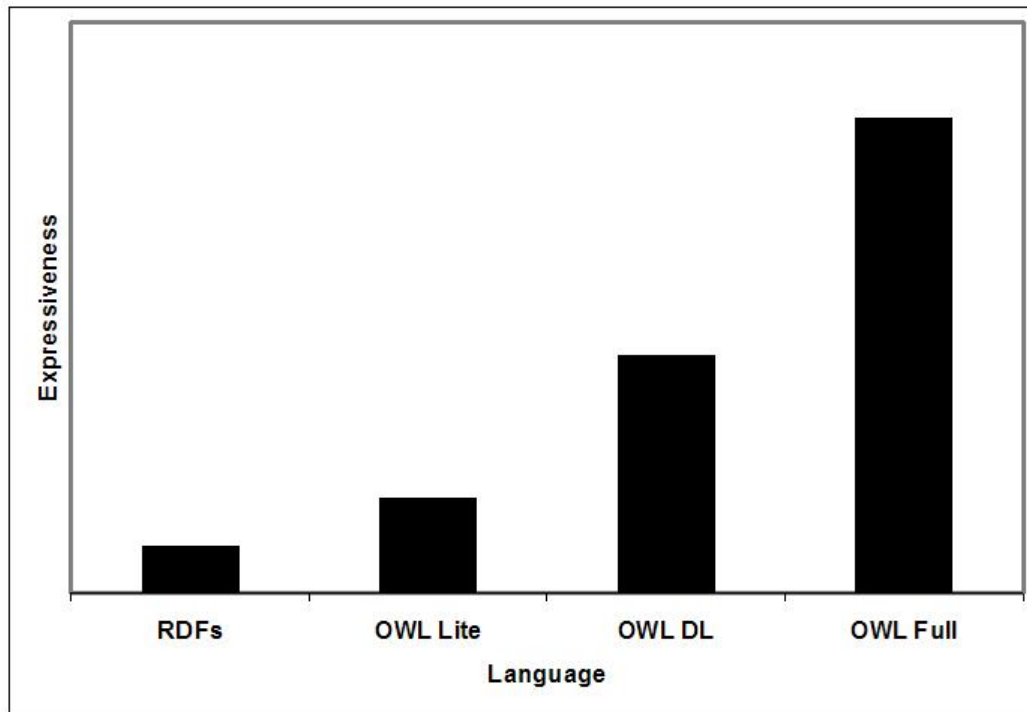


person. However, a property like 'lives in' has deeper meaning. Saying a person lives in a specific place ties that person to many of the attributes of that place. For example, Figure 2.13 shows that because 'Mark lives in Austin', and 'Austin is famous for daily live music', then 'Mark can enjoy live music daily'. The next section discusses how levels of expressiveness are captured in vocabularies.

2.4.4 Ontology Web Language (OWL)

Vocabularies play a key role in more expressive description of data relationships. A vocabulary can provide “extra knowledge for defining terms, restrictions, and extra relationships” [45]. For example, a vocabulary can make 'lives in' and 'famous for' properties

Figure 2.14: Expressiveness of Ontology Description Languages



available for use as predicates in RDF triples. The expressiveness and restrictions offered by a vocabulary can vary.

Ontologies, taxonomies, and thesauri some are ways to capture a vocabulary [92]. An ontology describes the concepts of a domain and the relationships between those concepts [71]. A taxonomy is the practice of classification. A thesaurus is a list of semantically orthogonal topic search keys. Web Ontology Language (OWL) is OWL is recommended by W3C for capturing more expressive vocabularies in schemas [87].

Concretely, vocabularies are represented in schemas. Schemas contain RDF triples that can be merged with other RDF data to conform to the vocabulary. Vocabularies are intended to be reused across several domains. Figure 2.14, shows the range of expressiveness of the common schemas.

OWL has many levels of expressiveness as shown in Figure 2.14. OWL Lite, OWL DL, and OWL Full each extend each other. In other words, OWL Lite is valid OWL DL which is valid OWL Full. But OWL Full is not valid in OWL DL which is not valid in OWL Lite. Each extension adds more expressiveness and restriction [47].

Automated reasoning is the biggest advantage to using OWL DL over the others. A *reasoner* is a piece of software that can read and analyze RDF data to infer new relationships. The new relationships are added as additional RDF triples to a graph. More meaningful queries can result. OWL DL provides the maximum amount of expressiveness still allowing automated reasoning [47]. This could be a very big time saver for RECWEB because it saves time for entering in all possible combinations of data. For example, if a certain product is a valid gift for a daughter, then a reasoner might also assume the gift could come from a mother. Since automation is important to the RECWEB problem, OWL DL is the best option.

2.4.5 SPARQL Query Language for RDF

Once RDF data is available, applications and intelligent agents need a way to access it. There are several query languages [73–75]. However, W3C has recommended the SPARQL Query Language for RDF as the standard. [78].

Similar to SQL, SPARQL is query language with syntax and semantics. A query can be made several different sources of RDF data. The query contains a set of patterns to match against the set of RDF graphs. Like SQL, filters and constraints can be specified to limit results.

The result of a SPARQL query is either a traditional set of tuples (i.e. rows of data) or RDF graphs. The graph is just a set of RDF triples drawn from multiple data sources. The results can express significant relationships that are not present when viewing just

one of the data sets individually.

2.4.6 Justification

W3C has developed many widely used Internet technologies. This includes several specifications, guidelines, software, and tools such as:

- Cascading Style Sheets (CSS)
- Hypertext Markup Language (HTML)
- Hypertext Transfer Protocol (HTTP)
- Portable Network Graphics (PNG)
- Simple Object Access Protocol (SOAP)
- Scalable Vector Graphics (SVG)
- Uniform Resource Identifier (URI)
- Uniform Resource Locator (URL)
- Web Services
- Extensible Markup Language (XML)
- Extensible Stylesheet Language (XSL)
- Extensible Stylesheet Transformations (XSLT)

The proven track record of W3C is putting the Semantic Web on a success path [35]. W3C has released stable specifications for most of key components of Semantic Web (RDF, SPARQL, and OWL). Like most application domains, a variety of tools are available [6,13,

52]. Many software vendors have added Semantic Web features into product lines [52,72]. Reference and learning material are abundant [46]. The aforementioned examples show the learning curve is small for basic entry into Semantic Web technology. Finally, large RDF datasets are beginning to accumulate [2, 4, 5]. The Semantic Web is emerging at the corporate and commercial levels [44].

W3C and others are calling for new applications to be developed in order to advance the adoption of Semantic Web [46, 77]. Several industries such as Health Care have already made large commitments to Semantic Web technology. A few case studies show the potential of Semantic Web in automating content management [21]. In addition, there are some use cases involving e-commerce [41]. However, there are no known published case studies or use cases related to increasing modifiability of seasonal systems like RECWEB.

Chapter 3

A Highly Modifiable RECWEB

Section 2.2 described the typical modifiability offered by a RECWEB. This section shows test cases, design, and implementation of a highly modifiable RECWEB.

3.1 Test Cases

Table 3.1: Selecting a Greeting


Description
"Happy New Year is a greeting for New Years"... "Today is January 1, therefore greet users with 'Happy New Year'"
Steps
<ol style="list-style-type: none">1. Open web browser to <code>http://localhost/home?month=1&day=1&year=2008</code>2. Verify that greeting is "Happy New Year!"
Results


Table 3.2: Selecting an Image


Description
"Santa Claus comes on Christmas Eve"... "Today is December 24, therefore an image of Santa Claus should appear on home page"
Steps
<ol style="list-style-type: none">1. Open web browser to <code>http://localhost/index.html?month=12&day=24&year=2007</code>2. Verify that image of Santa Claus is displayed.
Results
 <p>The screenshot shows a Mozilla Firefox browser window titled "RECWEB.com - Mozilla Firefox". The address bar contains the URL <code>http://localhost:6826/home?month=1&day=1&year=2008</code>. The main content area features a large illustration of Santa Claus on the left. To the right of the illustration, the text reads "RECWEB" in large bold letters, followed by "Lowest prices around!" and "Merry Christmas!". A small "RIGHTLOGO" is visible to the right of the promotional text. Below the main content area, there are three navigation tabs labeled "NAVIGATION", "CONTENT", and "PROMOTION". The status bar at the bottom of the browser window shows "Done".</p>

Table 3.3: Selecting Holiday Products

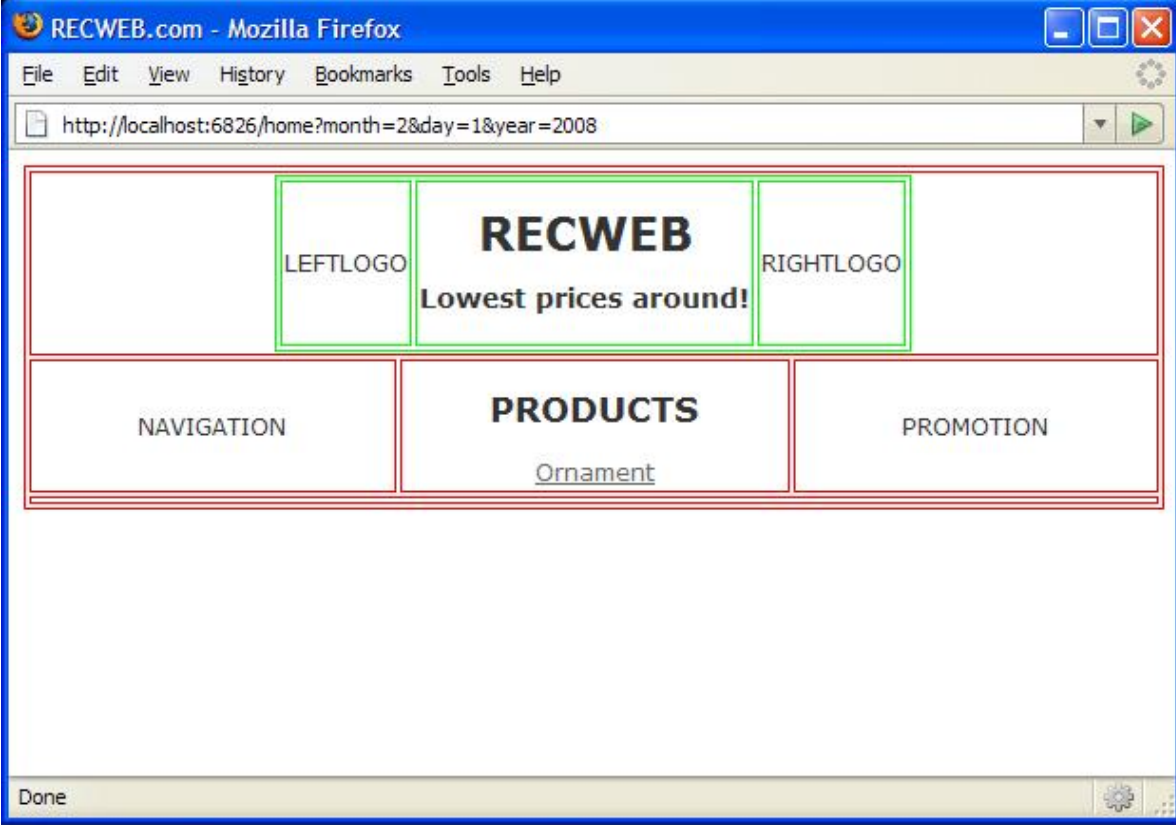
Description
Product A is an ornament. Ornaments are Christmas gifts”....”Today is Dec. 15, therefore Product A should be highly relevant”
Steps
<ol style="list-style-type: none">1. Open web browser to <code>http://localhost/index.html?month=1&day=15&year=2008</code>2. Verify that Christmas products are showing.
Results
 <p>The screenshot shows a Mozilla Firefox browser window. The title bar reads "RECWEB.com - Mozilla Firefox". The menu bar includes "File", "Edit", "View", "History", "Bookmarks", "Tools", and "Help". The address bar contains the URL "http://localhost:6826/home?month=2&day=1&year=2008". The main content area is divided into three columns: "NAVIGATION", "PRODUCTS", and "PROMOTION". The "PRODUCTS" column displays "Ornament". The header area contains "LEFTLOGO", "RECWEB", and "RIGHTLOGO", with the tagline "Lowest prices around!" below "RECWEB". The status bar at the bottom shows "Done".</p>

Table 3.4: Selecting Products and Categories

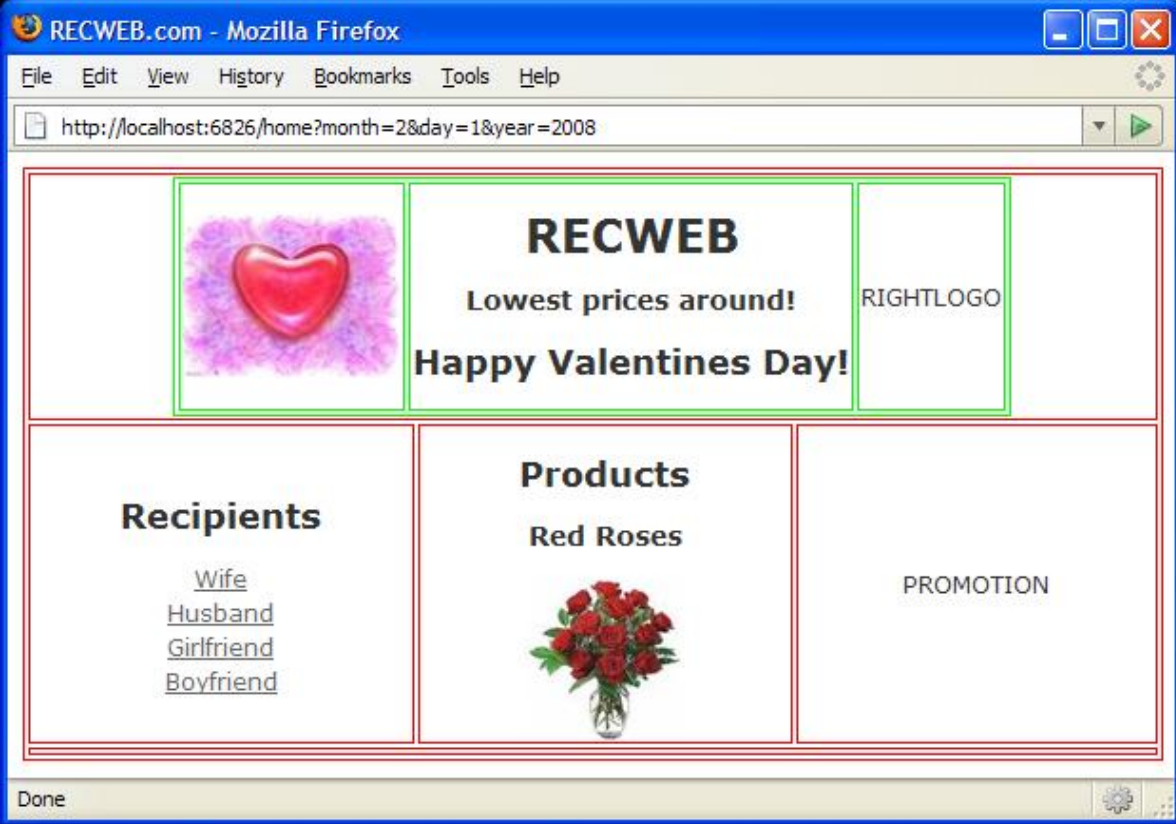
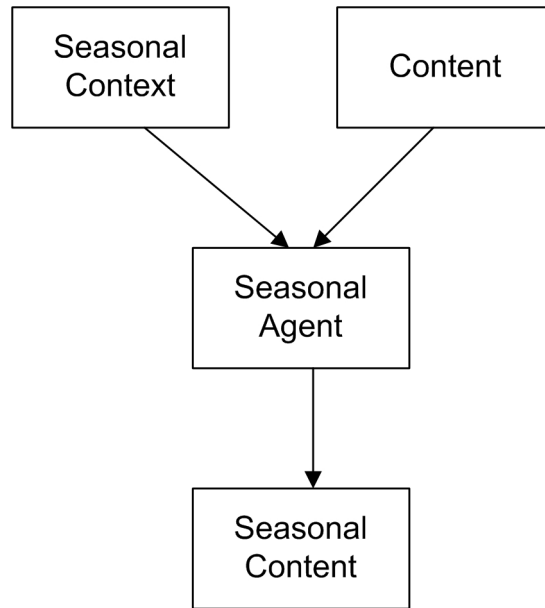
Description
Product B is red roses. Red roses are popular gifts for wives and girlfriends on Valentine's Day. Today is Feb. 1, therefore red roses should be placed in wife and girlfriend categories, and added to home page."
Steps
<ol style="list-style-type: none">1. Open web browser to <code>http://localhost/index.html?month=2&day=1&year=2008</code>2. Verify that red roses show on the home page.3. Verify that wife and girlfriend categories are on the home page.
Results
 <p>The screenshot shows a Mozilla Firefox browser window titled "RECWEB.com - Mozilla Firefox". The address bar displays the URL <code>http://localhost:6826/home?month=2&day=1&year=2008</code>. The page content is organized into a grid. The top row contains a heart image on the left, the text "RECWEB Lowest prices around! Happy Valentines Day!" in the center, and a "RIGHTLOGO" placeholder on the right. The bottom row is divided into three columns: "Recipients" with links for "Wife", "Husband", "Girlfriend", and "Boyfriend"; "Products" with the text "Red Roses" and an image of a bouquet of red roses; and "PROMOTION".</p>

Table 3.5: Selecting a Seasonal Products

Description
Product C is patio furniture. People sit on patios during the summer.”...”Today is Sept. 1, therefore patio furniture should go on sale”
Steps
<ol style="list-style-type: none">1. Open web browser to <code>http://localhost/index.html?month=9&day=1&year=2008</code>2. Verify that patio furniture is on sale.
Results
 <p>The screenshot shows a Mozilla Firefox browser window with the title "RECWEB.com - Mozilla Firefox". The address bar contains the URL <code>http://localhost:6826/home?month=9&day=1&year=2008</code>. The page content is divided into several sections, each outlined with a red border and labeled with text in a green box:</p> <ul style="list-style-type: none">LEFTLOGO: A placeholder for a logo on the left side of the header.RECWEB: The main site name in large, bold, black letters.RIGHTLOGO: A placeholder for a logo on the right side of the header.Lowest prices around!: A tagline centered below the site name.NAVIGATION: A placeholder for navigation links in the main content area.CONTENT: A placeholder for the main content in the middle of the page.Sale!!: A promotional banner in the right sidebar.Patio Furniture: A sub-header for the sale, with a corresponding image of a wooden patio set. <p>The status bar at the bottom of the browser window displays the word "Done".</p>

Figure 3.1: RECWEB Intelligence Dataflow

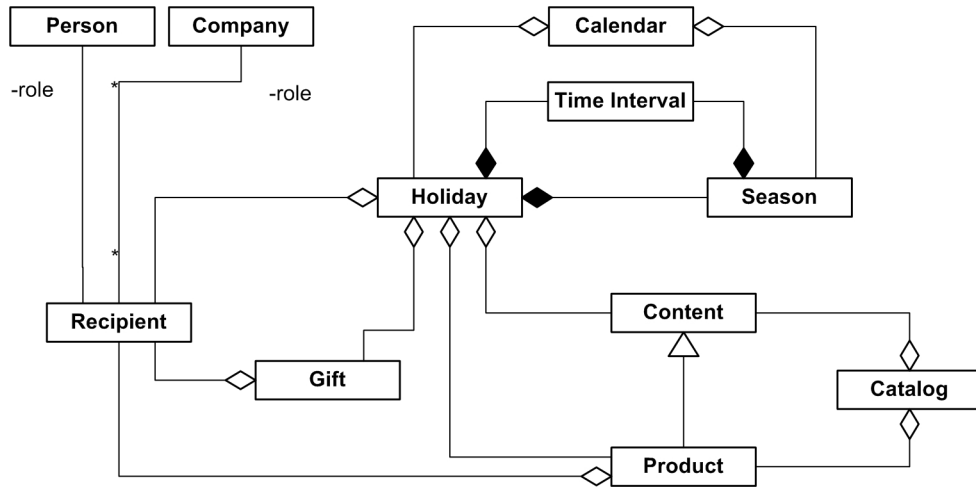


3.2 Design

Figure 3.1 shows a simple model for implementing the basic intelligence. The model is abstracted to a higher level and intentionally made informal. The purpose is to leave flexibility for implementation. For example, the following sections describe some details for each element in the model. However, an implementation is free to add additional details so long as the main output is conceptually seasonal content. Of course, the arrival at the output must be achieved with minimal human involvement and in an automated fashion.

Automating content changes must integrate with existing MVC-based RECWEB COTS packages, therefore Section 2.2 provides the baseline for this solution. For example, the responsibility of a controller is to process user gestures, report state changes to the model, and select views. In the new solution, none of these responsibilities are changed structurally. The main changes are adding seasonal automation functionality

Figure 3.2: RECWEB Seasonal Model



to the internals of controller and model elements. Sections 3.2.1 and 3.2.3 detail these changes.

3.2.1 Model

Seasonal Context An astonishing observation from the author’s experience with RECWEB organizations, is that very few make an effort to capture a seasonal context explicitly in systems. In fact, one organization’s entire business is based on seasonal products and they did little to model seasonal context. This section provides the foundation for capturing a seasonal context. Figure 3.2 shows an initial attempt at a seasonal context model.

A season is a time interval on a calendar. For example, Christmas season is the last Friday of November until December 25. Summer in North America is June 21 through Sept 20. Two seasons can overlap such as Christmas and Winter. One season is based on religious meaning and the other on climate.

Active season(s) can be determined by comparing the current date and time with

a seasonal calendar. An implementation can define the data structure to represent a seasonal calendar. In addition, the seasonal calendar data must be accessible to content agent(s). Typically, the seasonal calendar would remain static once created. For example, seasons based on climate nor holidays rarely change. However, different organizations are likely have different calendars. For example, a sporting goods retailer is more concerned with climate driven calendar. Whereas, a greeting card company is concerned with holiday calendars.

Each season has a set relevant attributes. For example, winter can involve dressing warm, drinking hot chocolate, and snow skiing. Christmas is a time for social gathering, shopping, and gift giving. Certain symbols or concepts are also attached to Christmas such as Santa Claus and Christmas trees. An implementation should be able to correlate the attributes for a season to relevant content.

The default seasonal context should be based on current system time. The frequency that the system time is checked and seasonal content regenerated is determined by the units of the time intervals for the seasons. In the case of RECWEB it could be daily. For example, 12:00 AM on December 26 would trigger the system to regenerate seasonal content. The content could switch the seasonal context from Christmas to Valentine's Day.

Ideally, the system should allow the seasonal context to be manually configured at runtime. A system administrator should be able to pick a date and time in order to simulate a season. For example, December 15 is Christmas season, but setting the date and time to January 1 allows for simulation of Valentine's Day. The benefits are twofold. First, an assessment of how well Valentine's Day content is prepared can be done real time. Second, enhancements depending on Valentine's Day seasonal context can be viewed with production data. Essentially, this greatly reduces guess work of determining the state of RECWEB content at future dates. On the flip side, previous dates could

also be examined.

Content In the Section 2.2, *RECWEB content* is defined as logos, navigation, products, and promotions. The bulk of seasonal changes are navigation and products. Therefore, this section focuses on the requirements for preparing navigation and products to be seasonally processed. However, nothing should prevent other implementations to use any type of content.

Product catalog data must contain enough information either explicitly or implicitly indicate seasonal relevance. For example, 'Christmas' could appear in a keywords attribute. This would be an explicit seasonal indicator because it matches the exact name of a holiday. However, 'Santa Claus' in a product title is an implicit indicator. This is because 'Santa Claus' is only a symbolic indicator of the Christmas holiday.

If explicit seasonal indicators are absent, it is harder for an agent to determine the seasonal relevance of a product. This is a legitimate concern because organizations do very little to explicitly model seasonal contexts. Therefore, most product catalog data only contains implicit seasonal indicators. Addressing this problem is not required by the evaluation criteria set forth in Section 1.4. Although, Section 5 describes potential future work on the problem.

Seasonal Content Seasonal content is the output of processing a seasonal context with content or in this case product catalog data. How the seasonal content is used is up to the implementor. The only requirement is that the seasonal content is generated by the machine and not a human being. Again, the point is to free up human resources for enhancements activity.

There is also some subjectivity to determining what is and is not seasonal content. For example, 'Santa Claus Tree Ornament' is obviously relevant to Christmas. But in

another case, a marketing person might decide that 'iPods' should be marketed as a gift for mom on Christmas. This could be based on the opinion of the marketing person or on something else like sales data. Again, all of the decision making is up to the implementor, so long as the seasonal content generation is automated.

3.2.2 View

Structurally, there is little change to the views. The responsibility remains to capture/report user gestures and reflect the current state of the model. The previous section shows the major change is how current state of the model is computed. The views are already set up to reflect it.

Recall Figure 2.8 shows two patterns for view updates: *pull* and *push*. In the pull pattern, the view has knowledge of what season should be queried on the model. In the push pattern, except the controller queries the model and passes the seasonal results to the view. In both cases, the knowledge of season is manually set by a site maintainer in, for example, a configuration file. In the new model, the knowledge of season is automatically determined by system time. This determination could be made in either the view or the controller depending on which push/pull pattern is desired. However, the push pattern has the advantage of less parts to maintain. For example, only one home page need to be maintained. For this reason, the new tactic diminishes the value or need for the pull pattern.

The internals of the views might provide greater usability because more accurate seasonal content is projected. For example, customers don't have to look at Christmas categories when Valentine's Day is more relevant. This leads them to products faster. It also shows the current modifiability tactics applied by MVC work correctly. In this case, the model and controllers have changed. By applying tactics to *localize changes* as

shown in Figure 2.2, the view isn't required to change.

The role of model update views shown in Figure 2.7 is also diminished. The model will still need to be updated manually because the new tactic is being phased in. In addition, the model update views provide an override should content personnel not be satisfied with the automated results. Recall simulation is also a desired feature. Content personnel can set the time forward to see how well content is prepared for a future holiday or season. The model update view comes in handy for filling in gaps for future content. However, model update view would not be the main means of changing content as in the past.

3.2.3 Controller

Structurally, there is little change to the controller. The responsibility remains to process user gestures from views, report state changes to the model, and select appropriate view. The previous section shows the major change is functionality must be added to the controller to compute a seasonal context. The seasonal context is then used to query the model.

The controller inevitably must maintain reference to a calendar that contains information about holidays and seasons. The data format and how the calendar gets updated is up to the implementation. However, the calendar must be accessible to the controller. The controller also controls how often the calendar is read and the seasonal context is updated.

Once a seasonal context is determined, the controller must format and inject to the seasonal context into all queries to the model. This insures that the model data returned contains seasonally relevant content. The controller is responsible for pushing the right seasonal content to views.

3.3 Implementation

A complete COTS package is not feasible for implementation because of the expense of licensing such packages. A single license can cost thousands of dollars. In addition, organizations that license a COTS packages to run a RECWEB were unwilling to participate in this academic research. Lack of resources to dedicate and lack of COTS vendor support were cited as the two major concerns. However, several organizations were intrigued by the potential of increasing modifiability of their RECWEB. All recognized the problem of maintaining the site in the seasonal context.

Section 2.2 provides an abstraction of the basic MVC components inside of a typical RECWEB COTS package. The abstraction is based of the author's extensive use of such packages as well as publicly available COTS package documentation. The abstraction formed the baseline for the design in Section 3.2. The same abstraction is used in this chapter for implementing the design.

The design is implemented by building a simple MVC-based prototype web application. The web application dynamically generates a RECWEB home page based on Figures 2.4a and 2.4b. The home page shows RECWEB seasonal content. The seasonal content is automatically generated based on system time, but can be overridden by passing parameters in the URL. This simple prototype web application shows that the preceding design described in Section 3 is feasible. The following sections give the details of the prototype.

3.3.1 Model

The first step in establishing a *seasonal context* is determining what the current season is. The implementation created a function that receives two parameters: current date and calendar location. If no current time is specified then the current system date is

used.

The calendar format accepted is iCalendar (iCal). iCal is a standard format supported by numerous applications for calendar data exchange. iCal files can be created, exported, and imported very easily. The result is users of the new highly modifiable RECWEB have many choices on how to create seasonal calendars.

The implementation treats all events on a calendar as seasonal. Therefore, the iCal file passed as input should be exclusively used to denote the seasonal calendar for the particular user of the application. The season name should be stored in the 'Summary' field of the iCal event record. The 'dtstart' and 'dtend' fields are used to specify the beginning and end of the season. For example, Figure 3.1 shows iCal entries for Mother's Day and Summer.

Listing 3.1: Seasonal Calendar in iCal

```
1 BEGIN:VCALENDAR
2 PRODID:-//Google Inc//Google Calendar 70.9054//EN
3 VERSION:2.0
4 CALSCALE:GREGORIAN
5 METHOD:PUBLISH
6 X-WR-CALNAME:Seasonal
7 X-WR-TIMEZONE:America/Chicago
8 X-WR-CALDESC:
9 BEGIN:VTIMEZONE
10 TZID:America/Chicago
11 X-LIC-LOCATION:America/Chicago
12 BEGIN:DAYLIGHT
13 TZOFFSETFROM:-0600
14 TZOFFSETTO:-0500
15 TZNAME:CDT
16 DTSTART:19700308T020000
17 RRULE:FREQ=YEARLY;BYMONTH=3;BYDAY=2SU
18 END:DAYLIGHT
19 BEGIN:STANDARD
20 TZOFFSETFROM:-0500
```



```

21 TZOFFSETTO: -0600
22 TZNAME: CST
23 DTSTART: 19701101T020000
24 RRULE: FREQ=YEARLY;BYMONTH=11;BYDAY=1SU
25 END: STANDARD
26 END: VTIMEZONE
27 BEGIN: VEVENT
28 DTSTART;VALUE=DATE: 20080421
29 DTEND;VALUE=DATE: 20080526
30 DSTAMP: 20080306T224713Z
31 CLASS: PRIVATE
32 CREATED: 20080306T223820Z
33 DESCRIPTION: Mother's Day
34 LAST-MODIFIED: 20080306T223820Z
35 LOCATION:
36 SEQUENCE: 0
37 STATUS: CONFIRMED
38 SUMMARY: Mother's Day
39 TRANSP: TRANSPARENT
40 END: VEVENT
41 BEGIN: VEVENT
42 DTSTART;VALUE=DATE: 20080621
43 DTEND;VALUE=DATE: 20080921
44 DSTAMP: 20080306T225129Z
45 CLASS: PRIVATE
46 CREATED: 20080306T224144Z
47 DESCRIPTION:
48 LAST-MODIFIED: 20080306T224144Z
49 LOCATION:
50 SEQUENCE: 0
51 STATUS: CONFIRMED
52 SUMMARY: Summer
53 TRANSP: TRANSPARENT
54 END: VEVENT
55 END: VCALENDAR

```

The summary of an event can contain a plain text description of the season or a URI. The difference is the URI is more specific and easier to match to an ontology. If plain

text is used, the seasonal context function is restricted to keyword matching of seasonal content. A URI is easier when using a reasoner to match seasonal content. However, both options are there until a fully baked seasonal ontology exists.

The implementation took an initial stab at creating an ontology for seasonal content. The newly designed model in Figure 3.2 heavily influenced the ontology creation. Protege OWL is the tool used to create and manipulate the ontology. OWL DL is used to maximize the ability for automated reasoning.

To control scope only one top level ontology is created. However, careful attention is made to avoid any naming conflicts with other ontologies. Namespaces would prevent any collisions. However, the goal would be to reuse as many concepts from other ontologies. For example, the MILO ontology contains the concept of holidays and fixed holidays. Instead of creating the duplicate concept in a new ontology, SUMO could be used. But this is simply out of scope for this work.

After developing the ontology, the basic steps were followed for implementing a Semantic Web application: *map, merge, and query* RDF data. Mapping data to the new ontology is done in 3 ways:

- manual data entry
- simple string pattern matching
- using a simple natural language processor
- tagging images with embedded RDF

Enough data is mapped to the seasonal ontology to implement the merge and query from inside the prototype web application.

The reasoner also played a role in generating more RDF data. Several properties specified in the ontology were set up with inverse properties. This means using one of

these properties in an RDF statement automatically generated the counter statement. For example, *husband* is specified to *give gifts to a wife*. The reasoner automatically generates the inverse statement which specifies that a *wife* can *receive gifts from a husband*. The result is a lot of additional RDF data generated automatically saving time.

Images were also tagged with RDF data. When the data is all merged together, it enabled image content to be returned in the same queries for products and content. This represents a change from traditional RECWEB COTS applications which might only store a physical location of an image. In this new solution, the seasonal aspects of images are also made available in the same ways of regular content. This simplifies the application code that must query these sources.

3.3.2 View

The views created for the prototype are very typical of any dynamic web application. Each view focused on reading, formatting, and rendering HTML to be consumed by a web browser. No business logic is included per the MVC based design.

The push pattern is used to get data to the views. Controllers query the model and expose the returned data for use by views. Views do not query the model directly. This is directly reflective of the new design outlined in Section 3.2.

3.3.3 Controller

The controllers essentially function like most COTS package controllers. However, seasonal content is retrieved by performing SPARQL queries against the new ontology. The results of the query are made available for consumption by views. The controller is implemented as a middleware component so the output could be simply ignored by the view. This would be useful to run traditional relational database queries side by side.

Standard performance management techniques are enabled. Caching of the output of views and controllers are applied. For example, the seasonal context only changes at the stroke of midnight. Therefore, the seasonal context just needs to be computed once and the output stored in a cache. For all subsequent requests by views and controllers, the cached data is used. This is a very common feature of most COTS packages and can drastically increase performance.

Chapter 4

Evaluation

In Section 1.4, several assumptions and constraints are presented. These are a basis for evaluating the solution to the RECWEB modifiability problem. This section evaluates the new highly modifiable RECWEB against these criteria. The purpose is to show that the new design and implementation increases modifiability without drastically increasing costs.

The first goal is to get the seasonal content change scenario out of the critical path of the enhancements scenario. This work shows this is feasible by automating seasonal content changes. This accomplishment alone makes a RECWEB more modifiable. With content changes out of the critical path it becomes feasible to develop and deploy enhancements at any time. Furthermore, people and resources required for modifying and maintaining a RECWEB during peak seasons can conceivably be reallocated toward enhancement development.

The second goal is to not increase costs at the expense of modifiability. In the first step, the risk of enhancement development is reduced. This means less occurrences of high investment in an enhancement project only to see it delayed because of not making a holiday season deployment. The result is more quantity and successful enhancements

into the system. This should translate into increase revenue because enhancements bring a competitive edge. Such revenue would offset some of the costs of automating seasonal content changes.

So what are the costs of automating content changes? People, COTS integration, and quality tradeoffs all present cost constraints. The following sections address each of these constraints with respect to cost. The purpose is to show the new design and implementation of a highly modifiable RECWEB does not increase costs.

4.1 People

A common suggestion to any problem is to add more people. In some cases in retail, it certainly makes sense. For example, temporary help is very common in brick and mortar retail stores. A store will hire additional staff to help accommodate the increase in shoppers during the holiday seasons. The temporary staff helps with tasks such as keeping the shelves organized and stocked with the correct products. Holiday transition can certainly benefit from additional staff. For example, two aisles might be dedicated to seasonal products. On December 24th, these aisles would have all Christmas items. However, on December 26th the aisles need to be replaced with Valentine's Day items. This would involve temporary staff physically moving the items.

A minimum wage employee can be instructed to physically move items around a store. The movement of those items also do not affect the inventory of another store. On a RECWEB, technical skills are required to move items around the online store. The changes are seen by a much wider audience of the store. Aside from physically making the changes to the RECWEB, another skill is required to decide what changes to make. This can be subjective and require a marketing skill, or it can be merely driven by business rules. The point is this skill is far beyond hiring just temporary staff during the holidays.

Cost is the main reason temporary staff is not feasible. The hourly rate for a temporary employee in a brick and mortar store is exponentially lower than a resource required for a RECWEB. Figure 4.1 shows the types of human resources required for a RECWEB. The hourly rates are driven by the skill level required to make holiday specific changes or enhancements.

The rates are reflective of consulting based employees as opposed to lower cost full time salaried employees. However, if the full time salaried employee is not 100% utilized the cost becomes comparable to using consultants. A full discussion of this tradeoff is out of scope. The full time or salaried is still exponentially greater than a minimum wage temporary employee.

Table 4.2 shows staff cost estimates for just one holiday transition. The estimates are based off a waterfall development model. The holiday transitions are in fixed time box. Therefore, all development tasks (requirement, design, implementation, etc.) have to be fit into to a time box. The estimates show staff allocated appropriately.

Holiday transition staff can either be new hire or borrowed from enhancement projects. In the former, the additional resource costs money. In the latter, the borrowed resource jeopardizes the enhancement project time line.

Table 4.3 shows the amount of resources that can be removed from a highly modifiable RECWEB. This represents a cost savings of over 50%. In fact, the cost savings for just one holiday transition is \$177,200. In the typical case of seven holiday transitions, this adds up to just over \$1.2 million dollars. Therefore, the new highly modifiable RECWEB does not require temporary or borrowed resources. In fact, it frees up resources for other use. The freed up resources could be used to increase monitoring of availability and performance during high traffic. For example, additional testers and network engineers could be added to holiday transition teams.

Table 4.1: Roles

<i>Role</i>	<i>Description</i>	<i>Hourly Rate</i>
Project Manager	Responsible for delivering the solution on time and within budget.	\$175
Architect	Responsible for vigourously understanding requirements. Makes high level design decisions that help achieve all functionality and qualities such as avaliablity, performance, and modifiability.	\$175
Developer	Responsible for implementing architect's design. Writes and maintains source code in multiple programming languages.	\$125
Tester	Responsible for verifying and validating all delivered artifacts. Writes and executes test cases reporting any defects.	\$100
Business Analyst	Analyzes impacts of changes on cost and revenue.	\$75
UI Designer	Responsible for designing user interface changes. Might build prototypes using HTML editors.	\$100
Graphics Artist	Responsible for designing media artifact for a site. Designs and develops images, videos, and styles for a site.	\$100
Marketing Manager	Responsible for making decisions regarding pricing, products, promotions, and placement.	\$100
Content Manager	Responsible for implementing business analyst and marketing manager's decisions.	\$75
Network Engineer	Responsible for the network, hardware, and software infrastructure in production, staging, and development environments.	\$125

Table 4.2: Cost Estimate - Current Seasonal Transition

Activity	Days	PM	Arch	Dev	Tester	NE	BA	UI	GA	MM	CM
Requirements	5	5	5	10	5	5	5	5	5	5	5
Design	10	10	10	20	5	3	10	10	10	10	10
Implement	10	10	2	20	10	5	5	10	10	10	10
Test	5	5	1	10	5	5	5	5	5	5	5
Deploy	1	1	1	1	1	1	1	1	1	1	1
Total Days	31	31	19	61	26	19	26	31	31	31	31
Cost		\$43,400.00	\$26,600.00	\$61,000.00	\$20,800.00	\$19,000.00	\$15,600.00	\$24,800.00	\$24,800.00	\$24,800.00	\$18,600.00
Grand Total											

Table 4.3: Cost Estimate - Seasonal Transition in Highly Modifiable RECWEB

Activity	Days	PM	Arch	Dev	Tester	NE	BA	UI	GA	MM	CM
Requirements	0	0	0	0	0	0	0	0	0	0	0
Design	0	0	0	0	0	0	0	0	0	0	0
Implement	5	0	0	5	5	0	5	10	10	10	10
Test	10	0	0	0	10	10	5	10	10	10	10
Deploy	1	1	1	1	1	1	1	1	1	1	1
Total Days	16	1	1	6	16	11	11	21	21	21	21
Cost		\$1,400.00	\$1,400.00	\$6,000.00	\$12,800.00	\$11,000.00	\$6,600.00	\$16,800.00	\$16,800.00	\$16,800.00	\$12,600.00
Grand Total											
		\$102,200.00									

Table 4.4 shows the costs of a typical enhancement project. Enhancement projects have about a six month time box to complete. Waterfall development is not always required because some projects might not require the full six months. Agile approaches such as Scrum are much more common. These approaches are iterative and involve several cycles of incrementally delivering a working software product. Each cycle, or *sprint*, encompasses a full set of activities: requirements, design, implement, test, deploy. At the end of each sprint a working set of functionality is delivered and demonstrated. All the estimates for enhancement projects are based on this approach.

Resources could also be made available for enhancements work. Project managers, architects, and developers can all contribute to enhancement projects. The savings from seasonal transition work is more than enough to fund an entire additional enhancement project. The other option is to augment an existing enhancement project to either shorten the project's sprint cycles (See Table 4.5) or utilize higher skilled team members (See Table 4.6) The RECWEB is more modifiable because it allows more enhancements and faster projects.

There is a cost for the initial development and maintenance of the highly modifiable RECWEB. The design and implementation in Sections 3.2 and 3.3 are fairly complex. Therefore, an architect and an above average developer might be required. Table 4.7 shows an initial estimate for this project.

Table 4.4: Cost Estimate - New Enhancement Project

Activity	Days	PM	Arch	Dev	Tester	NE	BA	UI	GA	MM	CM
Requirements	20	20	20	20	20	20	20	20	20	20	20
Sprint #1	25	25	25	75	10	5	5	10	10	5	10
Sprint #2	25	25	25	75	10	5	5	10	10	5	10
Sprint #3	25	25	25	75	10	5	5	10	10	5	10
Deploy	10	10	10	10	10	10	10	10	10	10	10
Total Days	105	105	105	255	60	45	45	60	60	45	60
Cost		\$147,000.00	\$147,000.00	\$255,000.00	\$48,000.00	\$45,000.00	\$27,000.00	\$48,000.00	\$48,000.00	\$36,000.00	\$36,000.00
Grand Total		\$837,000.00									

Table 4.5: Cost Estimate - New Enhancement Project with Shorter Cycles

Activity	Days	PM	Arch	Dev	Tester	NE	BA	UI	GA	MM	CM
Requirements	20	20	20	20	20	20	20	20	20	20	20
Sprint #1	25	25	25	100	10	5	5	10	10	5	10
Sprint #2	25	25	25	100	10	5	5	10	10	5	10
Deploy	10	10	10	10	10	10	10	10	10	10	10
Total Days	80	80	80	230	50	40	40	50	50	40	50
Cost		\$112,000.00	\$112,000.00	\$230,000.00	\$40,000.00	\$40,000.00	\$24,000.00	\$40,000.00	\$40,000.00	\$32,000.00	\$30,000.00
Grand Total		\$700,000.00									

Table 4.6: Cost Estimate - New Enhancement Project with Higher Skills

Activity	Days	PM	Arch	Dev	Tester	NE	BA	UI	GA	MM	CM
Requirements	20	20	20	20	20	20	20	20	20	20	20
Sprint #1	25	25	25	75	10	5	5	10	10	5	10
Sprint #2	25	25	25	75	10	5	5	10	10	5	10
Sprint #3	25	25	25	75	10	5	5	10	10	5	10
Deploy	10	10	10	10	10	10	10	10	10	10	10
Total Days	105	105	105	255	60	45	45	60	60	45	60
Cost		\$147,000.00	\$210,000.00	\$357,000.00	\$48,000.00	\$45,000.00	\$27,000.00	\$48,000.00	\$48,000.00	\$36,000.00	\$36,000.00
Grand Total		\$1,002,000.00									

Table 4.7: Cost Estimate - Initial Highly Modifiable RECWEB

Activity	Days	PM	Arch	Dev	Tester	NE	BA	UI	GA	MM	CM
Requirements	20	20	20	20	20	5	20	5	5	20	20
Sprint #1	25	25	25	75	10	5	5	5	5	5	10
Sprint #2	25	25	25	75	10	5	5	5	5	5	10
Sprint #3	25	25	25	75	10	5	5	5	5	5	10
Deploy	10	10	10	10	10	5	5	5	5	5	5
Total Days	105	105	105	255	60	25	40	25	25	40	55
Cost		\$147,000.00	\$210,000.00	\$357,000.00	\$48,000.00	\$25,000.00	\$24,000.00	\$20,000.00	\$20,000.00	\$32,000.00	\$33,000.00
Grand Total		\$916,000.00									

4.2 COTS Integration

Another key constraint is integrating with typical COTS RECWEB packages. The design and prototype fit well into the base MVC design pattern adopted by most RECWEB COTS packages. However, most of the additions to the system were functional components deployed directly inside existing pieces of the typical RECWEB COTS package. There is no reason to believe deploying this MVC-based design into a real RECWEB COTS package would not work.

Integration could even occur in a phased in approach. The implementation shows that the normal queries for content can be made in parallel with seasonal content queries. The only cost is to performance, but the implementation utilized well known performance improvement techniques. These are discussed further in the next section.

One downside is the inability to test the design and implementation inside of an actual COTS package. The cost of licensing a COTS product and staff to support is just too high for a pilot project. However, the implementation used the normal extension points offered by COTS packages. The author has extensive experience designing and developing such extensions. The integration of the new tactic could be easily accomplished. The upside is the common abstraction can apply to multiple different COTS vendors.

4.3 Quality Tradeoffs

Availability of a RECWEB is improved just by the fact content changes are constantly being made by the system and not humans. One of the main reasons availability suffers is a RECWEB has to be taken down to deploy new content. The new tactic reduces the amount of times this must occur. Additional monitoring resources are also available as mentioned in Section 4.1.

Performance should not be affected. Existing COTS performance management techniques are utilized, for example, caching. Any performance benchmarks will translate.

Simulation is possible because a current time can be passed into the new system. The system should behave as if the time passed in is the current time. This greatly improves the ability to analyze availability, performance, and modifiability for the future. For example, content personnel have the ability to peek into the future and view content state. This reduces the amount of surprises and guess work that occur during peak season.

Chapter 5

Conclusion

The goal of this research is to create a highly modifiable RECWEB that is able to overcome seasonal constraints plaguing current RECWEB. All evaluation criteria set forth in Section 1.4 were achieved. In addition, the improvements show no cost increases.

Several additional benefits were realized during design and implementation. While the goal is to automate content changes, it has been discovered that the automation could enable more than just one change at a time. For example, if a product is tagged as a good gift for a mother, then content can also be generated that a daughter might give that gift to a mother. This could save a lot of time for content personnel during any manual entry effort. Of course, once the content is in the system it can be automatically reasoned about to generate seasonal content.

One other efficiency gained is being able to query different kinds of content types at the same time and from the same data store. This is made possible because RDF data can be embedded inside images as well as just attached to raw data. For example, the seasonal agent can find both images and products relevant to a specific holiday from the same data store.

The final contribution of this work is a seed use case for an application of Semantic

Web involving a RECWEB. There is a call for such use cases and no use case exists specific to seasonal constraints on a RECWEB. The organization of this work provides a framework for refining such a use case into a more production ready system that can be validated.

This work can spawn off in several research directions. The most feasible being the aforementioned Semantic Web use case. This work shows the Semantic Web can be used as a tactic to increase modifiability of RECWEB. However, much more work needs to be done to make it production ready. Continuing several iterations of *map*, *merge*, and *query* should get the solution in a position to be submitted as a valid use case to W3C.

Mapping RECWEB data to RDF might be challenging. Most organizations do not explicitly capture seasonal context or even attributes. This means that someone or something needs to extract the information from the current RECWEB data. This is not an uncommon problem in any Semantic Web application. At the time of writing, W3C has just finalized the Gleaning Resource Descriptions from Dialects of Languages (GRDDL) standard. Future research could involve utilizing some of the emerging GRDDL tools and technologies to extract RDF from RECWEB data.

Natural language processing (NLP) is another method for extracting RDF from RECWEB data. In fact, Semantic Web case studies have shown NLP to be effective in creating machine readable data [82, 91]. Future research could focus on NLP as the method for extraction. This work is very rudimentary at exploiting NLP.

Merging other vocabularies with the OWL of this work would be highly beneficial. This work is just a first step at fully understanding explicit relationships involved with RECWEB. But OWL are meant to be shared and reused to reduce the amount of work to maintain them. Furthermore, more meaningful relationships can be inferred by using more common vocabularies with RECWEB OWL.

Querying RDF data is the final but very important step in the Semantic Web devel-

opment process. At the time of writing, SPARQL is also a new standard. As new tools and technologies to support SPARQL emerge, queries can be reevaluated and optimized.

Finally, one cannot discount the role of AI in this solution. The seasonal agent designed here should continue to be studied in both the context of Semantic Web and traditional AI. This work only implements basic intelligence in the seasonal agent. However, future research could really focus on more complex intelligence inside the seasonal agent.

Regardless of the direction taken, a highly modifiable RECWEB is needed. This work shows modifiability can be increased through automation of seasonal content changes while containing costs. Future research can use this pilot project as a framework for applying new techniques and technologies.

Bibliography

- [1] Java blueprints: Model-view-controller. <http://java.sun.com/blueprints/patterns/MVC-detailed.html>, 2002.
- [2] Swoogle: Semantic web search. <http://swoogle.umbc.edu/>, 2007.
- [3] Websphere commerce version 6 information center. <http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/index.jsp>, 2007.
- [4] Freebase: An open shared database of the world's knowledge. <http://www.freebase.com/>, 2008.
- [5] Schemaweb. <http://www.schemaweb.info/>, 2008.
- [6] Sw tools and systems. http://www.semanticweb.gr/index.php/Tools_and_Systems, 2008.
- [7] Gediminas Adomavicius and Alexander Tuzhilin. Personalization technologies: a process-oriented perspective. *Commun. ACM*, 48(10):83–90, 2005.
- [8] L. David Balk and Ann Kedia. Ppt: a cots integration case study. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 42–49, New York, NY, USA, 2000. ACM Press.
- [9] Robert Balzer, Alexander Egyed, Neil Goldman, Tim Hollebeek, Marcelo Tallis, and David Wile. Adapting cots applications: an experience report. In *IWICSS '07: Proceedings of the Second International Workshop on Incorporating COTS Software into Software Systems: Tools and Techniques*, page 7, Washington, DC, USA, 2007. IEEE Computer Society.
- [10] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*, pages 3–17. Addison Wesley, 2nd edition, 2003.
- [11] Steven M. Beitzel, Eric C. Jensen, David D. Lewis, Abdur Chowdhury, and Ophir Frieder. Automatic classification of web queries using very large unlabeled query logs. *ACM Trans. Inf. Syst.*, 25(2):9, 2007.

- [12] Tim Berners-Lee, R. Fielding, and L. Masinter. RFC 3986, Uniform Resource Identifier (URI): Generic syntax. <http://tools.ietf.org/html/rfc3986>, 2005.
- [13] Chris Bizer and Daniel Westphal. Developers guide to semantic web toolkits for different programming languages. <http://www4.wiwiss.fu-berlin.de/bizer/toolkits/index.htm>, 2007.
- [14] Barry Boehm and Chris Abts. Cots integration: Plug and pray? *Computer*, 32(1):135–138, 1999.
- [15] Frederick Brooks. *Mythical Man Month*, pages 3–17. Addison Wesley, 2nd edition, 2003.
- [16] Katy Byron. Wal-mart site knocked offline by traffic. http://money.cnn.com/2006/11/24/technology/walmart_website/index.htm?postversion=2006112414, 2006.
- [17] K. Selçuk Candan, Wen-Syan Li, Qiong Luo, Wang-Pin Hsiung, and Divyakant Agrawal. Enabling dynamic content caching for database-driven web sites. In *ACM SIGMOD 2001*, pages 532–543. ACM, 2001.
- [18] Gerardo Canfora and Aniello Cimitile. Software maintenance, 2000.
- [19] James R. Challenger, Paul Dantzig, Arun Iyengar, Mark S. Squillante, and Li Zhang. Efficiently serving dynamic data at highly accessed web sites. *IEEE/ACM Transactions on Networking*, 12(2):233–246, 2004.
- [20] Karen Clay, Ramayya Krishnan, and Eric Wolff. Pricing strategies on the web: evidence from the online book industry. In *EC '00: Proceedings of the 2nd ACM conference on Electronic commerce*, pages 44–55, New York, NY, USA, 2000. ACM Press.
- [21] Susie Cone and Kathy MacDougall. Case study: The swordfish metadata initiative: Better, faster, smarter web content. <http://www.w3.org//2001/sw/sweo/public/UseCases/Sun/>, 2007.
- [22] Reidar Conradi and Bernhard Westfechtel. Version models for software configuration management. *ACM Comput. Surv.*, 30(2):232–282, 1998.
- [23] World Wide Web Consortium. Semantic web layercake. <http://www.w3.org/2007/03/layerCake.svg>.
- [24] Robert Cooley. The use of web structure and content to identify subjectively interesting web usage patterns. *ACM Trans. Inter. Tech.*, 3(2):93–116, 2003.

- [25] Paul Dantzig. Architecture and design of high volume web sites. In *The Fourteenth International Conference of Software Engineering and Knowledge Engineering*, pages 17–24. ACM, 2002.
- [26] Anindya Datta, Kaushik Dutta, Helen Thomas, Debra Vandermeer, and Krithi Ramamritham. Proxy-based acceleration of dynamically generated content on the world wide web: An approach and implementation. *ACM Transactions on Database Systems*, 29(2):406–443, 2004.
- [27] Marie desJardins, Eric Eaton, and Kiri L. Wagstaff. Learning user preferences for sets of objects. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 273–280, New York, NY, USA, 2006. ACM Press.
- [28] Jens Dibbern, Tim Goles, Rudy Hirschheim, and Bandula Jayatilaka. Information systems outsourcing: a survey and analysis of the literature. *SIGMIS Database*, 35(4):6–102, 2004.
- [29] E. Dibella. Hacked for the holidays: how an anonymous network attack almost brought one small business to its knees. *netWorker*, 6(1):26–31, 2002.
- [30] E.W. Dijkstra. On the role of scientific thought. <http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD447.PDF>, 1974.
- [31] Jacky Estublier. Software configuration management: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 279–289, New York, NY, USA, 2000. ACM Press.
- [32] Jacky Estublier and Sergio Garcia. Process model and awareness in scm. In *SCM '05: Proceedings of the 12th international workshop on Software configuration management*, pages 59–74, New York, NY, USA, 2005. ACM Press.
- [33] Jacky Estublier, David Leblang, Andr#233; van der Hoek, Reidar Conradi, Geoffrey Clemm, Walter Tichy, and Darcy Wiborg-Weber. Impact of software engineering research on the practice of software configuration management. *ACM Trans. Softw. Eng. Methodol.*, 14(4):383–430, 2005.
- [34] John Favaro. On the scalability problem in cots-based programming environments. *SIGSOFT Softw. Eng. Notes*, 21(5):43–46, 1996.
- [35] Lee Feigenbaum, Ivan Herman, Tonya Hongsermeier, Eric Neumann, and Susie Stephens. The semantic web in action. *Scientific American*, pages 90–97, December 2007.
- [36] Simon Fong and Chan Se-Leng. Modeling personnel and roles for electronic commerce retail. In *SIGCPR '00: Proceedings of the 2000 ACM SIGCPR conference on Computer personnel research*, pages 45–53, New York, NY, USA, 2000. ACM Press.

- [37] Xavier Franch and Marco Torchiano. Towards a reference framework for cots-based development: a proposal. In *MPEC '05: Proceedings of the second international workshop on Models and processes for the evaluation of off-the-shelf components*, pages 1–4, New York, NY, USA, 2005. ACM Press.
- [38] Piero Fraternali. Tools and approaches for developing data-intensive web applications: A survey. *ACM Computing Surveys*, 31(3):228–263, 1999.
- [39] Vassil Gedov, Carsten Stolz, Ralph Neuneier, Michal Skubacz, and Dietmar Seipel. Matching web site structure and content. In *WWW2004*, pages 228–263, 2004.
- [40] Vladimir Geroimenko. *Dictionary of XML Technologies and the Semantic Web*, pages 3–17. Springer, 1st edition, 2004.
- [41] Rayid Ghani. Mining the web to add semantics to retail data mining. <https://www.accenture.com/NR/rdonlyres/42AA8DB9-F430-4DB5-A424-64050FAC9F80/0/ewmfghani.pdf>.
- [42] Jörg M. Haake. Facilitating orientation in shared hypermedia workspaces. In *GROUP '99: Proceedings of the international ACM SIGGROUP conference on Supporting group work*, pages 365–374, New York, NY, USA, 1999. ACM Press.
- [43] Stuart Hansen and Timothy V. Fossum. Refactoring model-view-controller. *J. Comput. Small Coll.*, 21(1):120–129, 2005.
- [44] Jim Hendler. Web 3.0: Chicken farms on the semantic web. *Computer*, 41(1):106–108, Jan. 2008.
- [45] Ivan Herman. Introduction to the semantic web. In *International Conference on Dublin Core and Metadata Applications*, <http://www.w3.org/2007/Talks/0831-Singapore-IH/>, 2007. W3C.
- [46] Ivan Herman. State of the semantic web. In *INTAP Semantic Web Conference*, <http://www.w3.org/2008/Talks/0307-Tokyo-IH/>, 2007. W3C.
- [47] Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, and Chris Wroe. A practical guide to building owl ontologies using the protégé-owl plugin and co-ode tools edition 1.0, August 2004.
- [48] Nora Houari and Behrouz Homayoun Far. Application of intelligent agent technology for knowledge management integration. In *Proceedings of the Third IEEE International Conference on Cognitive Informatics (ICCI'04)*, 2004.
- [49] IBM. Websphere commerce family. http://www-306.ibm.com/software/genservers/commerce/analyst.html?S_TACT=103BHW06&S_CMP=campaign, 2008.

- [50] Alexa Internet Inc. Alexa: The web information company. <http://http://www.alexa.com/>, 2008.
- [51] Jeremy Jaech, Stephen North, Mike Peery, Will Schroeder, and Jim Thomas. The visualization market: Open source vs. commercial approaches. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 113, Washington, DC, USA, 2003. IEEE Computer Society.
- [52] Elisa Kendall. Semantic web tools. <http://esw.w3.org/topic/SemanticWebTools>, 2008.
- [53] Shobhana Kirtane and Jim Martin. Application performance prediction in autonomic systems. In *ACM-SE 44: Proceedings of the 44th annual Southeast regional conference*, pages 566–572, New York, NY, USA, 2006. ACM Press.
- [54] In-Young Ko, Ke-Thia Yao, and Robert Neches. Dynamic coordination of information management services for processing dynamic web content. In *World Wide Web (WWW) Conference*, pages 355–365. ACM, 2002.
- [55] Ravindra Krovi, Akhilesh Chandra, and Balaji Rajagopalan. Information flow parameters for managing organizational processes. *Commun. ACM*, 46(2):77–82, 2003.
- [56] Mahesh Kumar, Nitin R. Patel, and Jonathan Woo. Clustering seasonality patterns in the presence of errors. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 557–563, New York, NY, USA, 2002. ACM Press.
- [57] John J. Kyaruzi and Jan van Katwijk. Concerns on architecture-centered software development: A survey. *J. Integr. Des. Process Sci.*, 4(3):13–35, 2000.
- [58] Alexei Lapouchnian, Yijun Yu, Sotirios Liaskos, and John Mylopoulos. Requirements-driven design of autonomic application software. In *CASCON '06: Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research*, page 7, New York, NY, USA, 2006. ACM Press.
- [59] M. M. Lehman. Lifecycles and the laws of software evolution. In *Proceedings of IEEE, Special Issue on Software Engineering*, pages 1060–1076. IEEE, 1980.
- [60] M. M. Lehman. Program evolution. *Journal of Information Processing Management*, 19(1):19–36, 1984.
- [61] Baoli Li, Wenjie Li, and Qin Lu. Topic tracking with time granularity reasoning. *ACM Transactions on Asian Language Information Processing (TALIP)*, 5(4):388–412, 2006.

- [62] Sharon Lymer, WenQian Liu, and Steve Easterbrook. Experience in using business scenarios to assess cots components in integrated solutions. In *CASCON '05: Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, pages 126–140. IBM Press, 2005.
- [63] Zhongming Ma, Gautam Pant, and Olivia R. Liu Sheng. Interest-based personalized search. *ACM Trans. Inf. Syst.*, 25(1):5, 2007.
- [64] Frank Manola and Eric Miller. Rdf primer. <http://www.w3.org/TR/rdf-primer/>, 2004.
- [65] Abdallah Mohamed, Guenther Ruhe, and Armin Eberlein. Decision support for customization of the cots selection process. In *MPEC '05: Proceedings of the second international workshop on Models and processes for the evaluation of off-the-shelf components*, pages 1–4, New York, NY, USA, 2005. ACM Press.
- [66] M. Morisio, C. B. Seaman, A. T. Parra, V. R. Basili, S. E. Kraft, and S. E. Condon. Investigating and improving a cots-based software development. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 32–41, New York, NY, USA, 2000. ACM Press.
- [67] Priya Nagpurkar, Hussam Mousa, Chandra Krintz, and Timothy Sherwood. Efficient remote profiling for resource-constrained devices. *ACM Trans. Archit. Code Optim.*, 3(1):35–66, 2006.
- [68] Tien N. Nguyen, Ethan V. Munson, John T. Boyland, and Cheng Thao. An infrastructure for development of object-oriented, multi-level configuration management services. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 215–224, New York, NY, USA, 2005. ACM Press.
- [69] Tien N. Nguyen, Ethan V. Munson, and Cheng Thao. Fine-grained, structured configuration management for web projects. In *World Wide Web (WWW) Conference*, pages 433–442. ACM, 2004.
- [70] Fred Niederman. Staffing and management of e-commerce programs and projects. In *SIGMIS CPR '05: Proceedings of the 2005 ACM SIGMIS CPR conference on Computer personnel research*, pages 128–138, New York, NY, USA, 2005. ACM Press.
- [71] Natalya F. Noy and Deborah L. McGuinness. *Ontology development 101: A guide to creating your first ontology*, 2004.
- [72] Zeljko Obrenovic, Tobias Burger, Pasquale Popolizio, and Raphaël Troncy. Multi-media semantics: Overview of relevant tools and resources. http://www.w3.org/2005/Incubator/mmsem/wiki/Tools_and_Resources, 2008.

- [73] Chimezie Ogbuji. Versa: Path-based rdf query language. <http://www.xml.com/pub/a/2005/07/20/versa.html>, July 20, 2005.
- [74] Uche Ogbuji. Versa, the rdf query language. <http://uche.ogbuji.net/tech/rdf/versa/>, 2007.
- [75] openRDF.org. User guide for sesame 2.0. <http://www.openrdf.org/doc/sesame2/users/ch09.html>, 2007.
- [76] Thomas B. Passin. *Explorer's Guide to the Semantic*, pages 236–268. Manning Publications, 2004.
- [77] David Provost. Hurdles in the business case for the semantic web. Master's thesis, Massachusetts Institute of Technology (MIT), Cambridge, Massachusetts, June 2004.
- [78] Eric Prud'hommeaux and Andy Seaborne. Sparql query language for rdf. <http://www.w3.org/TR/rdf-sparql-query/>, 2008.
- [79] K. Psounis. Class-based delta-encoding: a scalable scheme for caching dynamic web content. In *Proceedings of 22nd International Conference on Distributed Computing Systems Workshops*, pages 799–805, 2002.
- [80] Trygve Reenskaug. The model-view-controller(mvc) its past and present. http://heim.ifi.uio.no/~trygver/2003/javazone-jao0/MVC_pattern.pdf, 2003.
- [81] Gustavo Rossi, Daniel Schwabe, and aes Robson Guimar. Designing personalized web applications. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 275–284, New York, NY, USA, 2001. ACM Press.
- [82] Jesús Fernández Ruíz. Case study: An intelligent search engine for online services for public administrations. <http://www.w3.org//2001/sw/sweo/public/UseCases/Zaragoza/>, June 2007.
- [83] Adam Sarnier and Eugenio M. Alvarez. E-commerce magic quadrant, 4q2006. *Research*, 2006.
- [84] Adam Sarnier and Eugenio M. Alvarez. E-commerce magic quadrant, 4q2006. <http://http://www.atg.com/eCommerce/gartnerQ406/>, 2007.
- [85] Rahul Singh, Lakshmi S. Iyer, and A. F. Salam. The semantic e-business vision. *Communications of the ACM*, 48(12), 2005.
- [86] Yogesh Singh and Bindu Goel. A step towards software preventive maintenance. *SIGSOFT Softw. Eng. Notes*, 32(4):10, 2007.

- [87] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. Owl web ontology language guide. <http://www.w3.org/TR/owl-guide/>, 2004.
- [88] Felix Sockwell. The web within the web. *IEEE Spectrum*, pages 42–46, February 2004.
- [89] David Thames and Andrew Hunt. *Pragmatic Version Control Using CVS*, page 23. Pragmatic Programmer LLC, 2004.
- [90] Marco Torchiano, Letizia Jaccheri, Carl-Fredrik Sørensen, and Alf Inge Wang. Cots products characterization. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 335–338, New York, NY, USA, 2002. ACM Press.
- [91] José Luís Bas Uribe. Case study: Real time suggestion of related ideas in the financial industry. <http://www.w3.org//2001/sw/sweo/public/UseCases/Bankinter/>, May 2007.
- [92] Christopher Walton. *Agency and the Semantic Web*, pages 19–54. Oxford, 1st edition, 2007.
- [93] Rob Wojcik, Felix Bachmann, Len Bass, Paul Clements, Paulo Merson, Robert Nord, and Bill Wood. Attribute-driven design(add). Technical Report CMU/SEI-2006-TR-023, Software Engineering Institute, November 2006.
- [94] William G. Wood. A practical example of applying attribute-driven design (add, version 2.0). Technical Report CMU/SEI-2007-TR-005, Software Engineering Institute, February 2007.
- [95] Paris A. Zafiris, Nektarios P. Georgantidis, George E. Kalamaras, Sotiris P. Christodoulou, and Theodore S. Papatheodorou. A practitioner’s approach to evolving and remodeling large-scale www sites. In *Proceedings of the 34th Hawaii International Conference on System Sciences*. IEEE, 2001.
- [96] Uwe Zdun. Dynamically generating web application fragments from page templates. In *SAC 2002*, pages 1113–1120. ACM, 2002.
- [97] Qiankun Zhao, Steven C. H. Hoi, Tie-Yan Liu, Sourav S. Bhowmick, Michael R. Lyu, and Wei-Ying Ma. Time-dependent semantic similarity measure of queries using historical click-through data. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 543–552, New York, NY, USA, 2006. ACM Press.