Industrial Experience with Building a Web Portal Product Line using a Lightweight, Reactive Approach

Ulf Pettersson
SES Systems Pte. Ltd.
ulfp@stee.stengg.com

Stan Jarzabek
National University of Singapore
Department of Computer Science
stan@comp.nus.edu.sg

ABSTRACT

Imprecise, frequently changing requirements and short time-to-market create challenges for application of conventional software methods in Web Portal engineering. To address these challenges, SES Systems Pte Ltd. applied a lightweight, reactive approach to support a Web Portal product line. Unique characteristics of the approach were fast, low-cost migration from a single, conventional Web Portal towards a reusable "generic Web Portal" solution, effective handling of large number of functional variants and their dependencies, the ability to rapidly develop new Web Portals from the generic one, and to independently evolve multiple Web Portals without ever loosing a connection between them and the "generic Web Portal". The initial Web Portal was built using state-of the-art conventional methods. The Web Portal was not flexible enough to reap the benefits of new business opportunities that required SES Systems to rapidly develop and further maintain many similar Web Portals. To overcome the limitations of the conventional solution, a reuse technique of XVCL was applied incrementally. During three weeks the conventional solution was converted into a Web architecture capable of handling nine Web Portals from a base of code smaller than the original Web Portal. In the paper, we describe the process that led to building the above Web Portal product line. We explain the difficulties in building an effective generic Web solutions using conventional techniques. We analyze SES Systems' reuse-based solution in qualitative and quantitative ways.

Categories and Subject Descriptors

D.2.2 [**SOFTWARE ENGINEERING**]: Design Tools and Techniques; D.2.10 [**SOFTWARE ENGINEERING]**: Design − Representations; D.2.13 [**SOFTWARE ENGINEERING]**: Reusable Software - *Domain engineering;* D.1.2 [**PROGRAMMING TECHNIQUES**]: Automatic Programming – Program synthesis;

General Terms

Design, Languages, Experimentation

Keywords

Web engineering, reuse, software product lines, maintenance, static meta-programming

1. Introduction

Imprecise and frequently changed requirements, and short time-to-market are the main challenges of Web Portal (WP) development [5][28]. Given that today WPs have grown from mere collections of static HTML pages to full-fledged business applications, meeting these challenges is not easy. In this paper, we describe how SES Systems Pte. Ltd. (referred to as SES, for short) applied reuse via product line approach to tackle the problem. The aim was to enable reuse-based rapid development of new WPs, to reduce the maintenance cost of all the WPs SES had to manage, and to control evolution of both already created WPs and reusable assets defined within the product line architecture.

SES created a Web Portal Product Line incrementally, applying extractive [18] approach to build a first-cut "generic Web Portal", and then applying reactive approach [18] to continuously extend and refine the "generic Web Portal" with new features requested by customers. The starting point for this evolution towards a product line was an ASP-based personal WP developed by the first author. The personal WP was first converted to a Team Collaboration Portal (TCP) used internally at SES. Then, it became a business product portal used in hospitals to help in Severe Acute Respiratory Syndrome (SARS)-related problems Finally, it was re-designed with XVCL [32] into a "generic Web Portal", called an XVCL-based Web Portal Product Line Architecture (WP Architecture, for short), on which SES has built by now 20 different Web Portals.

Unique characteristics of our approach and project experiences are:

- Short time (less than 2 weeks) and small effort (2 persons) to transform the TCP into the first version of the WP Architecture.
- High productivity in building new portals from the WP Architecture.
- Wide range of portals differing in a large number of inter-dependent features supported by the WP Architecture.
- Ease of evolving the WP Architecture in new, unexpected directions.
- Ease of handling multiple portal products during evolution (of both individual portals and the WP Architecture), without loosing the connection between the reusable base of portal components in the WP Architecture and the portal code.

The above characteristics and productivity indicators are often mentioned as important problems in product line practice, but

difficult to address with conventional software technologies [4][9][12][19].

We attribute the above to a successful merger of two design techniques, namely model-based design[1] (MBD) and XVCL. MBD helped us design the WP Architecture for reuse using conventional methods. However, there were clearly many design-level similarities, intra-module, inter-module and across portals, that MBD could not take advantage of. We applied XVCL on top of the MBD solution to fully exploit reuse opportunities arising from those similarities.

In the paper, we describe the Web Portal Product Line project at SES and lessons we learned from it. In Sections 2 and 3, we describe the ASP Web Portal that was a starting point for the project. In Section 4, we describe some of the problems and challenges encountered while evolving this portal based on its original design. In Section 5, we introduce XVCL. In Section 6, we describe how we applied XVCL to create the Web Portal Product Line Architecture. We also evaluate and quantify our results. In Sections 7, we provide a summary of our project experiences. In Section 8 and 9, we discuss related work and conclude the paper.

2. The Team Collaboration Portal (TCP)

The starting point for the project was a Web-based Team Collaboration Portal (TCP for short) developed using Active Server Pages (ASP) [1]. The TCP was developed incrementally, starting with a personal web portal developed by the first author. This personal portal facilitated information sharing via management of users, HTML-content, images and video-clips [22]. Over time, the functionality of the personal portal was extended to include access statistics, news/announcements, weather animations, and posting/feedback facilities. The personal portal was then brought to the company environment where it was deployed as a team collaboration tool for software development teams, and as this was done, the portal functionality evolved further to form the TCP.

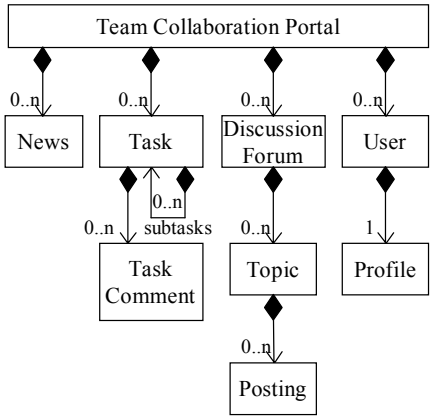

Figure 1: Partial functional view of the TCP.

[1] In model-based design, we create a model of an application from which we derive (generate or manually) parts of an application.

A subset of the functional modules of the TCP is illustrated in Figure 1. Some of the functional modules such as News are independent of other modules, while other functional modules have relationships (such as the Postings that is a composite of Topic that is a further composite of Discussion Forum). In addition to the functional modules illustrated in Figure 1, the TCP also provided facilities for site configuration and other usual infrastructure services for access control, data history capturing, statistics gathering, and database persistency.

3. Initial design of the TCP for reuse

Though a Web Portal product line was not our explicit target yet, we were aware of many similarities across the TCP. We viewed them as attractive reuse opportunities that could facilitate design solutions greatly reducing development cost, as well as the cost of future enhancements.

At SES Systems, we characterize and scope applications in terms of conceptual entities our applications should implement. We use entity models in the early stage of requirement analysis, and also in design and implementation. In the TCP, a functional module (Figure 1) was built from one or more Entity Modules, where each Entity Module (Table 1) implemented functionality related to a single entity.

At the top level, the TCP was designed around a Portal Foundation with complementary Entity Modules as illustrated in the Figure 2.

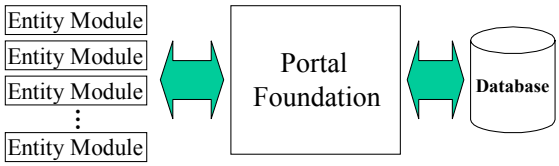

Figure 2. Top-level architecture of the TCP.

While some of the functional modules implemented specific and unique features, many features could be shared (after certain modifications) across the modules. Most of the shared features were focused around the management of entities underlying respective functional modules. For each entity such as News, Task or User we needed features to:

- Create an entity
- Edit/update of entity
- Delete of entity
- Display of entity
- Display of entity listing
- Capture and display of entity history

There were also similarities in the way infrastructure services interacted with various functional modules. From previous experience, we knew that visualization, persistency, search indexing and history capturing were done in a similar (and sometimes rather complicated) way across functional modules.

We built a number of models to identify and explicate similarities in the TCP analysis and design spaces. The TCP architecture was based on these models, facilitating reuse of similar solutions. We call this approach a model-based design, MBD for short, and explain it by examples below.

3.1. Entity Modules

Table 1 identifies most of the Entity Modules in the TCP.

Table 1. TCP Entities.

Entity Module	Description
Feedback	Capture user feedback on the portal.
Folder	Container for other entities.
Forum	Discussion forum.
Forum Topic	Topics within the discussion forum.
Help	General and context sensitive help information.
History	History records of entities (old versions).
HTML Content	Portal content in HTML format (on-line managed HTML content).
Menu	Menus for navigation within the portal (and to external URLs).
News	News/Announcement content.
Page Access	Control user access to pages.
Poll	Polls for user feedback.
Post	Comments posted for entities (such as comments on a HTML Content, or postings on a Forum Topic).
Profile	User-specific preferences.
Quiz	A simple on-line quiz (user self-tests applied after informal training sessions).
Rate Log	Capture user ratings of entities.
Search Index	Dynamic search index built as and when entities are updated.
Session Log	User session (login) history.
Site Information	Portal revision history, etc.
Task	Tasks assigned to users (ToDo & Action items).
User	Administration of users and their access rights.
User Action Log	Logging (audit trail) of all user input actions.

Figure 3 shows the entity model as a UML class diagram.

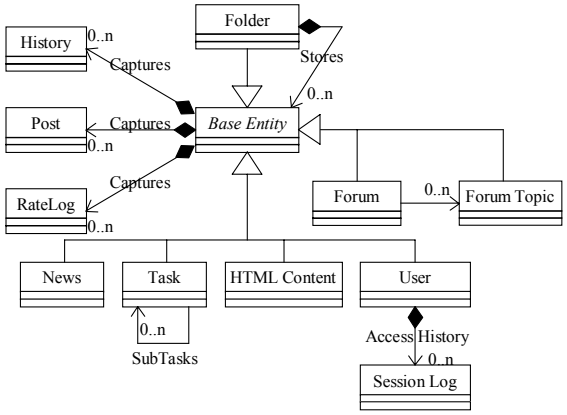

Figure 3. TCP model example.

As seen in the Figure 3, entities are derived from an abstract *Base Entity*. All the entities can be organized into Folders. The composites of the *Base Entity* (History, Post and Rate Log) are optional, applied only to entities that need them (e.g., Postings were required for Tasks, News, HTML Content and Forum Topics, but not for Forums, Users and Folders).

From the entity model, we identified some reuse opportunities such as the ability of the *Base Entity* to capture versions (History class), postings&comments (Post class), and ratings (RateLog class).

To further understand the common behavior of the entities, we developed a meta-model around the *Base Entity*, as illustrated in Figure 4.

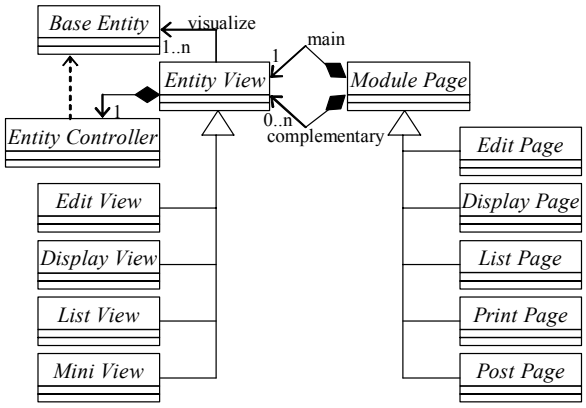

Figure 4. Meta-model around *Base Entity*.

The roles of the classes in the meta-model are as follows:

Base Entity: Defines entity attributes and mapping of attributes to and from database record set.

Entity Controller: Handles coordination between entity, entity views and Portal Foundation.

Entity View: Defines a presentation view of the entity (i.e. a sub-window displayed within a web page). Most Entity Modules implement specialized *Entity Views* such as:

- *Edit view*: A view for editing of an entity.
- *Display view*: A view for display of all attributes of an entity.
- *List view*: A view for displaying a sorted listing of entities.
- *Mini list view*: A view for a small sorted and/or filtered listing of entity objects (typically with entity specific behavior such as "Latest News", "My Tasks", "Latest Postings", etc).

Module Page: Represents a web page displayed in the Web browser. Each Entity Module typically defines a number of Module Pages where each such page includes a main view of the own entity and one or more complementary views (generated from arbitrary entities). Each Entity Module implements Module Pages as illustrated in Table 2.

Table 2. Module Pages for Entity Module.

Web Page	Main View	Purpose
List Page	List View*	Displays a list of entities.
Edit Page	Edit View*	Displays a form for entity editing.
Display Page	Display View*	Displays all details of an entity
Print Page	Display View*	Displays all details of an entity, but without a page header and without complementary views.
Post Page	Post Edit View#	Displays a form for postings related to an entity.

* Entity specific view.

General view shared by entities.

The personal portal example shown in Figure 5 illustrates how the various views were composed into the Web pages (this portal can be viewed at [22]).

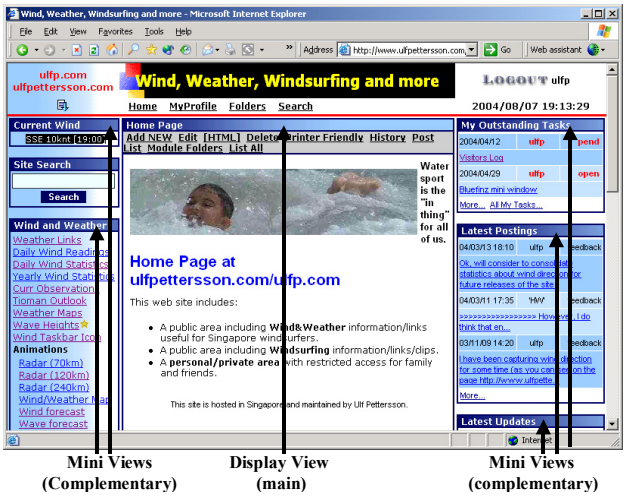

Figure 5. Web page composed by views.

3.2. Portal Foundation

The Portal Foundation (as illustrated in Figure 2) aimed to simplify the implementation of Entity Modules by providing:

- Session and Access Control.
- Page formatting.
- Site Configuration.
- Database persistency, indexing and search.
- Support for Base Entity meta-model patterns such as:
  - Entity visualization.
  - Entity persistency.
  - Entity indexing (to support generic search feature).
  - Entity history capturing (to support on-line history archive).

When the various entity related patterns (supported by the Portal Foundation) were implemented in the Entity Modules, it resulted in several repetitive code patterns (often termed as software clones). These clones were not seen as a major problem. In fact we felt that we arrived at a reasonable compromise between:

- Complexity risk arising from needs to handle too many and too complex variants of common functionality.
- Ease of Entity Module creation and modification.
- The amount of clones existing in the Entity Modules.

This assessment bears many similarities with common practices of clone assessment in the financial software industry as described by Cordy [10].

4. The product line problem

The Team Collaboration Portal (TCP) was first deployed to a few software development teams at SES. The key Entity Modules used were HTML-content/files/images (content management) forum, news/announcement, task, and change requests. At this point of time, no business vision existed for the TCP. It was merely an internal support tool rather than a business product.

During the SARS crisis in May 2003, SES developed a portal to facilitate analysis of people movements in hospitals. This portal was to register people movement within a hospital to facilitate tracing of people who could have been in contact with SARS infected persons. Under great time pressure (5 days), two of SES staff transformed the TCP into a People Tracking Portal and integrated this portal with an external Radio Frequency Identification (RFID) tracking system at two hospitals. Some key Entity Modules in the People Tracking Portal were: zone, visitor, visitor movement (including movement/contact analysis). With this deployment, the portal actually turned into a business product (even though from a business perspective this was more of a community service).

Several other portals were later developed and deployed. Even though development of new Entity Modules could be done with reasonable ease (typically 500-1500 code lines each), constantly growing need for enhancements to existing portals and development of new portals resulted in increasing difficulty to maintain the portal family.

Although a single team was maintaining most of the portals, it was not possible to maintain them from a single code base for the following reasons:

- The portals were deployed in a turnkey project manner rather than a product manner. As a result, some portals were in a frozen state where only critical defects would be corrected, and feature enhancements were thus not wanted.
- Some of the required feature enhancements were only applicable (and wanted) in one or a few portals and not in the whole portal family.
- Even though some entity modules were required in multiple portals, there were specific variations that applied to each individual portal resulting in difficulties to maintain a single code base.

At the same time, the SES team was given seven days to develop an eLearning demonstration portal that required about 20 new Entity Modules, and with the existing design and code volume required for each Entity Module, it was deemed impossible to achieve both design and implementation within a week.

In an effort to bring the situation under control, SES considered applying more specialized methods [28], but later decided to apply XVCL reuse technique [30][32]. A key factor in this decision was the ability of XVCL to handle variations to reusable elements in a selective manner, so that enhancements and modifications could be applied in specific products or throughout the product line. The decision was also influenced by encouraging results from earlier experiments where:

- XVCL could unify similarity patterns in Java Buffer Library and STL, resulting in reduction of maintained code by 68% [14] and 50% [3], respectively.
- Another team at SES Systems applied XVCL to a C# 3-tier application and achieved up to eight-fold effort reduction in enhancing the application.

5. Introduction to XVCL

XVCL (XML-based Variant Configuration Language) [32] is a meta-language, method and tool for enhancing changeability, maintainability, and reusability of programs. XVCL is applied on top of programs written in any programming language. The usual strategy is to develop a program solution using conventional design techniques, such as Object-Oriented or component-based. XVCL is then applied on top of the program, to inject extra degrees of changeability, non-redundancy or genericity, in the

areas that conventional techniques do not yield a satisfactory solution.

Using XVCL, we design generic, adaptable meta-components called x-frames. X-frames represent domain knowledge in the form of reusable assets, accommodating both domain defaults and variants. X-frame body is written in the base language, which could be a programming language such as Java, or a natural language such as English. XVCL commands allow the composition of x-frames (via <adapt> command), and also make x-frames customizable by allowing one to select pre-defined options based on certain conditions (via <select> command), by marking breakpoints where additional changes can be inserted (via <break> and <insert> commands), and by providing variables as a parameterization mechanism (via <set> and <value-of> commands).

X-frames are organized into a layered hierarchy called an x-framework. X-frames at lower-levels are building blocks of higher-level x-frames. The hierarchal x-framework enables us to handle variants at all the granularity levels.

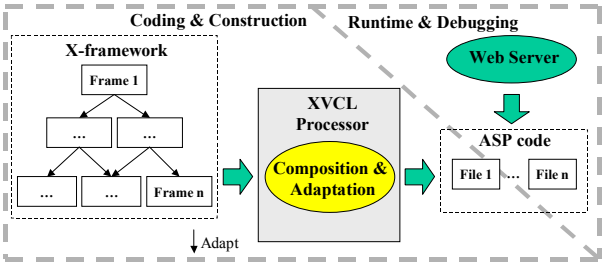

Figure 6. Construction of ASP Web Portals from a Web Portal Architecture (called an x-framework in XVCL jargon).

An x-framework is an XVCL implementation of a product line architecture concept [9]. XVCL supports automated configuration of variants in product line assets. Given specifications recorded in a special x-frame, the XVCL Processor traverses an x-framework, performs composition and adaptation by executing XVCL commands embedded in x-frames, and constructs components of a specific system, a member of a product line. In Figure 6, we see an illustration of the above in case of a Web Portal Product Line.

XVCL is based on Bassett's frame technology that has been applied by Netron Inc. to manage variants and evolve multi-million-line, COBOL-based information systems. An independent analysis showed that frame technology has reduced large software project costs by over 84% and their times-to-market by 70%, when compared to industry norms [4]. In 2000, a research team at the Software Engineering Lab of National University of Singapore developed XVCL [30][32], a refined form of frame language. XVCL refines original frames into a general-purpose meta-language that blends with contemporary programming and design paradigms. XVCL processors were also developed and in September 2002, XVCL was made available at an Open Source forum (fxvcl.sourceforge.net), from which the latest XVCL language specification, processor and source code can be downloaded. Since then, researchers at NUS and SES Systems have applied XVCL in many application domains (class libraries, business systems, Web Portals), programming languages (Java, C++, C#, ASP) and platforms (J2EE, .NET, Unix, Windows) [3][14][31][32].

6. Project application of XVCL

6.1. Web Portal Product Line with XVCL

SES team built an XVCL-based Web Portal Product Line in two phases. In the first phase, SES used XVCL to unify recurring patterns of similar design across the Entity Modules. This first approximation of the Web Portal Product Line Architecture was called an Entity Module Architecture. In the second phase, SES did the same for the Portal Foundation, creating a full-fledged Web Portal Product Line Architecture (WP Architecture, for short). The motivation for selecting this two-phased approach to building the WP Architecture was as follows:

- The most immediate business goal was to increase productivity of building and enhancing Entity Modules, while enhancements of the Portal Foundation were of the lesser concern.
- There were numerous variations in Entity Modules that required much effort in adapting Entity Modules to the needs of different portals, members of our WP Product Line. Variations in the Portal Foundation were significantly less and they were less complex.
- Based on earlier experiments [3][14][31], it could be expected that XVCL might help SES effectively handle types of similarities found in Entity Modules.

As exploiting reuse opportunities across Entity Modules was more important, beneficial, promising, instructive and risky, SES decided to launch an attack on it first. Two team members (with only general knowledge of the XVCL technique) completed the first phase in three working days. It should be also noted that these two persons were very familiar with the TCP code and that one person with extensive XVCL experience assisted them. A simplified view of the resulting Entity Module Architecture is illustrated in Figure 7.

As illustrated in Figure 7, the lowest level of the Entity Model Architecture consisted of several generic customizable x-frames grouped into views, pages and others. These generic x-frames were adapted by Entity Module x-frames to form specific Entity Modules. To construct the specific Entity Modules, these adaptations were done using various XVCL mechanisms. At the top level, WPs were formed by including the required, already customized, Entity Modules.

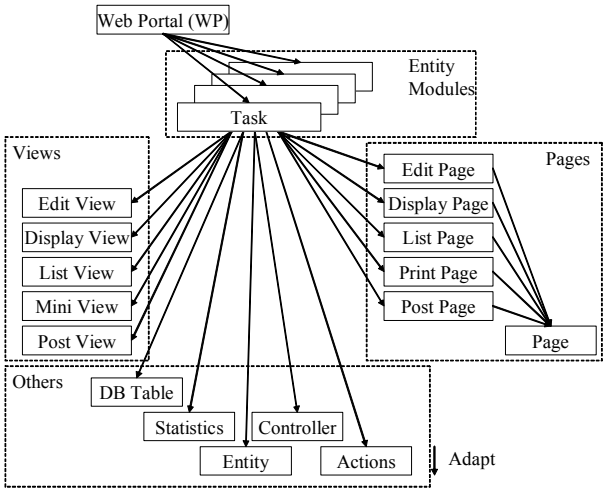

Figure 7. Entity Module Architecture for Phase 1.

The Entity Module Architecture was driven by the meta-model of the Entity Module (see Figure 4). The similarities of the design greatly contributed to the short time required to construct the Entity Module Architecture, and even though we yet have to find any major weaknesses in the concept and structure of the Entity Module Architecture, we make no claims of this being a standard approach for success.

The second phase targeted at inclusion of the Portal Foundation into the WP Architecture. Even though we put no special effort to identify similarity patterns and unify them with XVCL, this phase was still important as it established a foundation for future enhancements.

A comparison between the code lines in the portal before and after application of XVCL is provided in Table 3.

A sample comparison between the Entity Module code lines before and after application of XVCL is provided in Table 4.

Table 3. Portal code line comparison

	Original TCP (ASP)	WP Architecture (XVCL)	Generated portal (ASP)
Portal Foundation*	15180	14021	16401
Entity Modules*	16322	1577	30474
Entity Module X-Frames	N/A	4119	N/A
Σ	**31502**	**19717**	**46875**

*XVCL code for 8 entities were constructed without use of the Entity Module x-frames and thus included as a part of the Portal Foundation

Table 4. Entity Module code line comparison

Entity Module	Original TCP (ASP)	WP Architecture (XVCL)	Generated portal (ASP)
Help*	722	133	1490
Web Page	514	50	1428
User	755	54	1717
Task (todo)	1295	147	2429
. . .	. . .	. . .	. . .
Σ	**16322**	**1577**	**30474**

From Table 3 we see that the number of code lines have been reduced from 31502 to 19717 (37% reduction). This moderate reduction is due to the fact that the Portal Foundation in the original TCP and WP Architecture remained almost unchanged, since we did not target at unifying similarities in this area. If we instead focus on the Entity Modules, we can see a reduction from 16322 to 5696 (65% reduction), and this number is similar to what we have seen in earlier experiments [14]. It should also be noted that the managed code lines for Entity Modules have been reduced from 16322 to 1577 (90% reduction). This significant reduction of managed code translated into productivity gains that allowed us to create more than 20 Entity Modules for e-Learning demonstration portal less than one week.

From Table 4, we see that the code line reduction differs across the Entity Modules. These variations are dependent on how much the individual Entity Modules differ from the generic Entity Module in the Entity Module Architecture. The increase of code lines across Entity Modules could sometimes be linked to certain variation types that were inducing extra repeated similarity patterns. In such cases, we could often improve the Entity Module Architecture by unifying them with XVCL.

An example of such an improvement is that during the development of the WP Architecture, each entity module x-frame adapted a single page x-frame many times to form all the different page types (edit page, display page, print page, etc). Even though the adaptation of each page only required about 4-8 code lines, we found it beneficial to consolidate such clones into more specialized frames, so that each page type could be generated from a single code line. This specialization is illustrated in the transition from a) to b) of Figure 8.

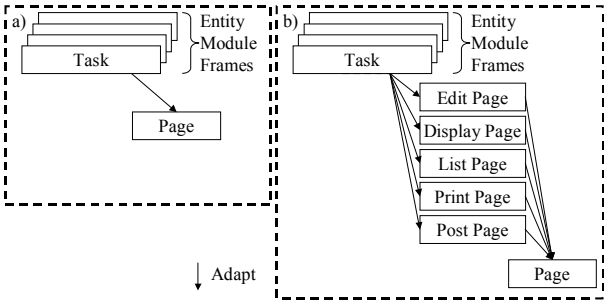

Figure 8. Evolving the WP Architecture to eliminate redundancies.

6.2. Enhancing the WP Architecture

Having implemented and used the WP Architecture, we continued to implement various enhancements required by new customers. Some of these enhancements were of similar kind to the enhancements that caused problems before the WP Architecture was created. We believe that meeting challenges of evolution, especially changes in emerging new, unexpected directions, is an ultimate test for the product line success. We believe our solution passed this test and in this section, we report some of our experiences in that area.

6.2.1. Entity attributes and business logic variations

The Entity Modules for management of Change Request [7] required special customization to cater for different levels of formality applied in small and big projects.

For small projects, we handled Change Requests in a quite informal manner by capturing only a few states (such as Submitted, Analyzed, Implemented, Closed, Duplicated and Rejected). We did not formalize the state transitions, but some business logic was required to capture the timing of state transition for the trend analysis purpose.

For larger projects, the Change Request would have additional attributes (some mandatory and some optional). Additional states and extended business logic were also required to enforce the state transition rules (e.g., Changes Requests had to enter the Approved state before transition to the Implemented state, etc.).

We started with a Change Request Entity Module for portals supporting small teams. This Entity Module was then enhanced with XVCL mechanisms to facilitate variations related to large teams. In particular, we created an x-frame to represent the specific attributes and business logic for Change Requests in portals for large teams. We then used this x-frame to extend the basic Change Request Module to cater for large team variations (Figure 9).

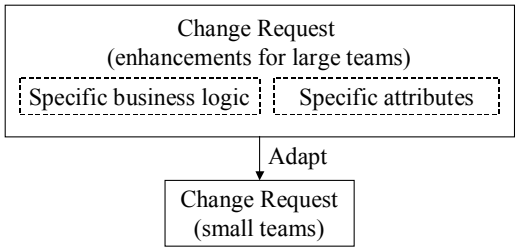

Figure 9. Extending the Change Request Module.

6.2.2. Entity relationship variations

Even though the entity attributes and business logic may be fully reused across portals, entity relationships may require customizations, too.

As an example, for certain portals a Log File Entity was created and defined as a composite of the Change Request for large teams. Again, we enhanced the Change Request Entity Module with XVCL mechanisms to represent the required relationships.

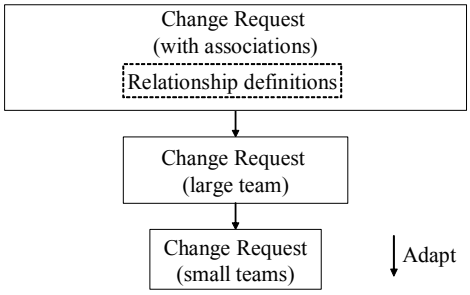

Figure 10. Extending the Change Request with relationships.

6.2.3. Other variations

To further support the Web Portal Product Line, we grouped the Entity Modules of our WP Architecture into four categories, shown in Figure 11.

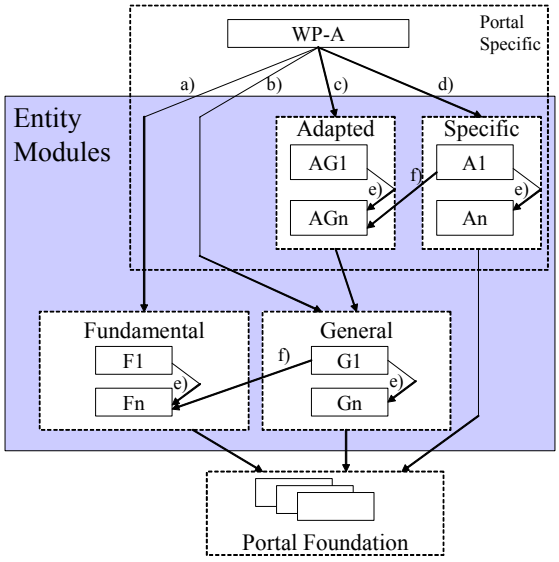

Figure 11. Entity Module organizations.

A specific portal, WP-A, was typically created using:

a) A common set of Fundamental Entity Modules (F1-Fn).
b) A selective set of General Entity Modules (a subset of G1-Gn)
c) A selected set of General Entity Modules adapted in a portal specific manner (AG1-AGn)
d) A set of Portal Specific Entity Modules (A1-An).

Within each group of Entity Modules, more specialized modules were created by adapting other modules e). Similar reuse was also applied for construction of General and Specific modules f).

The WP Architecture intentionally restricted cross-usage of parts specific to one WP c) and d) in other WPs built upon the WP Architecture. If during evolution such reuse was needed, developers converted d) structure to a b) or c) structure. This approach ensured that each WP only depended on the general and fundamental parts. By selecting this approach, SES could ensure independent evolution of WPs, without breaking their connection with WP Architecture. Individually evolving WPs could still benefit from upgrades of general or fundamental parts implemented in the WP Architecture. There was no need to keep traceability between reuse across WPs, either. Specific WP only needed to be re-processed if there was a change within the portal itself or in the general or fundamental parts. Using the solution above, new features of a Entity Module could be propagated to selected WPs, product line members, without affecting other WPs that did not need those features.

6.3. Building new portals based on the WP Architecture

We have developed nine portals based on the WP Architecture. Our WP Architecture facilitates incremental development of new portals, with each increment involving some or all of the following steps:

1. Functional Definition: A starting point for an entity model of the portal is the same as in Figure 3. This model is then refined with definition of typed attributes and input/display formats. The model is finally complemented by definition of business logic, with the intention of letting the model play an effective role in driving our architecture.

2. Portal Definition: First, we create an x-frame to indicate which Entity Modules provided by the WP Architecture are to be included into the new portal. Each such module may be reused "as is" or after adaptations.

3. Coding and testing: Entity Modules of a new portal are built based on “most similar” Entity Modules from the WP Architecture. At times, it is beneficial to select an already adapted Entity Module used in some other portal (as discussed in Section 6.2.3). The idea is to start with an Entity Module that requires minimum adaptation effort. The Entity Modules are then debugged and tested one at the time.

New portals may differ in reused entities, the details of their customizations, as well as in entities specific to the new portal (not catered for in the WP Architecture). Once implemented and tested, specific entities with reuse potential are included into the WP Architecture.

The level of customization of the nine first portals is illustrated in Table 5. The consolidated code lines for these portals are listed in Table 6. With our Web Portal Product Line, we have produced nine portals from a total of 24574 XVCL code lines (Table 6).

Comparing this with the 31502 ASP code lines of the original TCP (Table 3), we conclude that even though we have expanded from a single portal to a product line of 9 portals, we have reduced the code by 22%.

Table 5. Portal specific code lines

Portal Abbrev.	Entities			Portal specific XVCL
	Reused	Customized	Specific	
Apol	21	0	9	617
Cq	17	0	3	271
Csap	18	3	0	157
Demo	18	4	1	958
Ecap	19	3	4	363
Feptp	18	7	0	439
Gered	21	0	10	665
Eses	20	0	1	120
Ework	19	3	4	363
Σ	N/A	N/A	N/A	**3953**

Table 6. WP Architecture code lines for 9 portals

Item	XVCL	Remark
Portal Foundation	14021	(from Table 3)
Entity Module X-Frames (terminology)	4119	(from Table 3)
Shared entities	2481	Entities shared across portals
Portal specific code	3953	(from Table 5)
Σ	**24574**	Total for 9 portals

7. Summary of project experiences

7.1. Strengths of the adopted approach

With WP Architecture, SES observed significant benefits such as overall reduction of managed code lines by 37%, reduction of Entity Module code lines by 65%, and as much as 90% code line reduction for creation of new Entity Modules. For development of new Entity Modules, SES observed 4-8 times reduction in required effort. The size of the WP Architecture from which we generated nine WPs was 22% less than the size of the initial Team Collaboration Portal. This 22% reduction of code size is consistent with measurements/feedback that both effort and difficulty in maintaining the original TCP were significantly more than what is now required for the whole product line.

In component-based development, we usually try to stabilize an architecture, in particular component interfaces. The Web Portal project at SES started with the same perception. However, as the work progressed, it was found that, when applying XVCL, the stability of interfaces were less of an issue. Retrospectively, it seems that this phenomenon is due to non-redundancy achieved in the XVCL representation, but we have yet to come up with full interpretation of this result. This interface-tolerance and the ability of XVCL to absorb such changes with ease allowed SES to freely experiment with enhancements and changes to core Portal Foundation services and related mechanisms.

Many XVCL concepts, such as parameterization, selection and iteration, are similar to programming language concepts. Even though there are specifics of XVCL that must be understood, it was found relatively easy to start with XVCL as compared to conventional programming languages and environments where developers in addition to the language must be familiar with infrastructural support such as class libraries.

Interestingly, SES built our WP Architecture without conducting any systematic domain analysis [25]. Web Portals built in the past was the only source of the domain knowledge. Extractive approach [18] proved an effective way to built the initial WP Architecture, and reactive approach helped SES gradually refine it as SES was addressing new types of Web Portals. The advantage of such an approach, as compared to proactive approach, is low investment and fast results [18].

A common problem with scalability of the product line approach is explosion of the number of variant features and feature dependencies [12]. This often leads to the explosion of similar component versions that hinders reuse. While the WP project dealt with a substantial number of variant features, SES avoided explosion of similar component versions by identifying similarity patterns in component design and building generic components from which multiple forms of concrete components could be produced.

7.2. Pitfalls of the adopted approach

XVCL structures organize design and code at the meta-level for enhanced genericity and changeability. This meta-level decomposition is parallel to conventional modular decomposition along component (function or class) boundaries. A part of applying XVCL is that XVCL commands become embedded into the source code. As a result, developers must know and manage multiple languages within the x-frames (XVCL, ASP, HTML) and this kind of complexity may have impact on productivity and maintainability [5]. There is a great opportunity here for XVCL-specific tools to help developers analyze "mixed strategy" solutions offered by Web technology plus XVCL.

Another problem occurs when using XVCL together with Integrated Development Environments (IDE) that perform code generation. Since many popular IDE's support such features (such as MS Visual Studio, Eclipse, etc), the perception of some software engineers may be that XVCL is a step backwards in time, and even though the XVCL Workbench being developed at NUS makes a move in the direction of visual support for XVCL, we see the need for further industrial collaboration in this area.

Debugging is yet another area into which XVCL induces extra complexities. In WP project, the runtime debugging was performed on the ASP code generated by the XVCL processor (Figure 6). As a result, developers had to maintain a 'mental picture' of mappings from x-frames to the runtime ASP code. This extra step made debugging more complex. Again, proper tool support can help developers better cope with those problems.

Engineering processes play an important role in industrial software development, and even though SES has demonstrated how XVCL can help in development of a Web Portal Product Line Architecture, the experience comes from application in a small development team, with primary focus on software change

management (as defined by [7]). Future studies should focus on how to apply XVCL in larger development teams, using full-fledged software engineering processes.

8. Related work

Studies show much similarities at the design and code levels across Web Portals, both inside portal modules and across modules [17][26]. To save effort, developers often use "copy-paste-modify" to reuse existing functionality. While such ad hoc reuse cuts development cost, in long run it does not pay-off, as it hinders maintenance [17]. Tight schedules and much repetition make Web domain an attractive candidate for exploring more systematic forms of reuse [28], including the product line approach [6].

Advanced scripting languages such as PHP [21] help developers unify certain patterns of repetition, but it is yet to be shown if and how such languages can support systematic reuse as required in the product line approach. A number of authors explored modeling of Web application with UML for both requirement elicitation and reuse [2][27]. Our model-based design techniques described in this paper uses UML conventions to create models that drive reuse-based design of portal modules in a similar way as extended Entity-Relationship models facilitated generation of user interfaces, database schema, and sometimes parts of business logic in CASE tools [18].

Among Krueger's three approaches to building a product line [18], SES first applied extractive approach to build a first-cut Web Portal product Line Architecture, and then reactive approach to refine it with new features. SES project experiences resemble Salion's experiences [19]. By applying reactive approach, both projects achieved similar productivity figures, despite differences in application domains (Web Portals vs. Supplier-Customer systems) and reuse techniques applied (XVCL vs. GEARS [5]).

9. Conclusions

SES Systems Pte. Ltd. applied a reuse technique of XVCL to build a Web Portal Product Line. In the paper, we described a process that led to building the solution, and project experiences.

Unique features of the adopted technical approach is a successful merger of advanced conventional techniques (such as a model-based design), with light-weight application of meta-level reuse technique of XVCL. SES built a Web Portal Product Line Architecture (WP Architecture, for short) incrementally, applying extractive approach [18] to convert the personal Web Portal developed by the first author of this paper, into the fist-cut "generic Web Portal". The "generic Web Portal" was then refined into a WP Architecture with reactive approach.

Unique benefits observed in this project include:

- Significant productivity gains when building new portals. Based on the WP Architecture, we could build new portal modules by writing as little as 10% of unique custom code, while the rest of code could be reused. This code reduction translated into estimated eight-fold reduction of effort required to build new portals with reuse of the WP Architecture, as opposed to the original situation.
- Significant reduction of maintenance effort when enhancing individual portals. The overall managed code lines for nine portals were 22% less than the original single portal.
- Ease of enhancing individual portal products with new features in without loosing the connection between reusable base of portal components in the WP Architecture and the portal code.
- Short time (less than 2 weeks) and small effort (2 persons) to transform the TCP into the first version of the WP Architecture
- Ease of evolving the WP Architecture in new, unexpected directions.

The approach allowed SES to successfully face a number of product line challenges such as explosion of variant feature and their combinations, and evolution of the WP Architecture and individual products.

SES Systems Pte. Ltd. has a seven-year long partnership with the Reuse Group at the National University of Singapore (NUS). A joint research project under the Singapore-Ontario research programme, which also included University of Waterloo and Netron, Inc. from Toronto, resulted in the XVCL method and first pilot projects at SES Systems. This collaboration culminated in the application of XVCL to supporting the Web Portal Product Line, as described in this paper. It is hoped that the future joint NUS/SES projects can bridge the gap between XVCL as a mere powerful language and its integration into the best practices of industrial software engineering. In particular, it is planed to apply reuse techniques in more application domains, formulate systematic methods based on project experiences, define processes, and implement tools that can bring XVCL into industrial applications.

References

[1] Active Server Pages – ASP, http://msdn.microsoft.com/library/default.asp?url=/nhp/default.asp?contentid=28000522

[2] Aversano, L., Canfora, G., De Lucia, A., Gallucci, P., "Web site reuse: cloning and adapting" *Proc. 3rd Intl. Workshop on Web Site Evolution,* (WSE'01), pp.107 – 111

[3] Basit, H.A., Rajapakse, D.C., and Jarzabek, S. "Beyond Templates: a Study of Clones in the STL and Some General Implications," accepted for Int. Conf. Software Engineering, ICSE'05; available at http://fxvcl.sourceforge.net

[4] Bassett, P. Framing software reuse - lessons from real world, Yourdon Press, Prentice Hall, 1997

[5] BigLever Software Inc. www.biglever.com

[6] Capilla, R., Duenas, J.C., "Light-weight product-lines for evolution and maintenance of Web sites," *Proc. Seventh European Conference on Software Maintenance and Reengineering,* (CSMR' 2003), pp. 53 - 62

[7] Crnkovic, I. "Change Measurements in an SCM process," *System Configuration Management SCM-8 proceedings*, Brussels, Belgium. January 1998

[8] System Configuration Management SCM-8 proceedings, Brussels, Belgium. January 1998

[9] Clements, P. and Northrop, L. *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2002

[10] Cordy, J. R., "Comprehending Reality: Practical Challenges to Software Maintenance Automation," *Proc. 11th IEEE Intl. Workshop on Program Comprehension*, (IWPC 2003), pp. 196-206

[11] Czarnecki, K. and Eisenecker, U. Generative Programming: Methods, Tools, and Applications, Addison-Wesley, 2000

[12] Deelstra, S., Sinnema, M. and Bosch, J. "Experiences in Software Product Families: Problems and Issues during Product Derivation," Proc. Software Product Lines Conference, *SPLC3*, Boston, Aug. 2004, LNCS 3154, Springer-Verlag, pp. 165-182

[13] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns – Elements of Reusable Object-Oriented Software*, 1995, Addison-Wesley

[14] Ginge, A., and Murugesan, S. "Web Engineering: An Introduction," *IEEE MultiMedia*, 8(2), 2001, pp. 14-18

[15] Ginge, A., and Murugesan, S.: "The Essence of Web Engineering," *IEEE MultiMedia*, 8(2), 2001, pp 22-25

[16] Jarzabek, S. and Li, S. "Eliminating Redundancies with a "Composition with Adaptation" Meta-programming Technique," *Proc. ESEC-FSE'03, European Software Engineering Conference and ACM* SIGSOFT Symposium on the Foundations of Software Engineering, September 2003, Helsinki, pp. 237-246

[17] Lanubile, F., Mallardo, T., "Finding function clones in Web applications," Proc. Seventh European Conference on Software Maintenance and Reengineering, (CSMR' 2003). pp.379 – 386

[18] Krueger, C. "Eliminating the Adoption Barrier," Point-Counter Point Column, in *IEEE Software*, special issue on Initiating Software Product Lines, July/August 2002, pages 28-31

[19] Krueger, C. "Salion's Experience with a Reactive Software Product Line Approach," *5th Int. Workshop Product Family Engineering PFE5*, 2003, LNCS 3014, Springer-Verlag, pp. 317-322

[20] McClure, C. *CASE is Software Automation*, Prentice Hall, 1989

[21] Minsky, M., "A framework for representing knowledge," *The Psychology of Computer Vision*, P. Winston (ed.), New York: McGraw-Hill, 1975

[22] Personal Web Portal, http://www.ulfpettersson.com/.

[23] PHP: http://www.php.net/

[24] Pressman, R.S., et al. "Can Internet-Based Applications Be Engineered?," *IEEE Software,* vol. 15, no. 5, Sept 1998, pp. 104 – 110

[25] Prieto-Diaz, R. "Domain analysis for reusability," *Proc. COMPSAC'87*, October 1987, Tokyo, Japan, pp. 23-29

[26] Rajapakse, D. and Jarzabek, S. "An Investigation of Cloning in Web Portals," submitted for publication

[27] Rossi, G., Schwabe, D., and Lyardet, F., "Abstraction and Reuse Mechanisms in Web Application Models," *Proc. Conceptual Modeling for E-Business and the Web*(Er 2000), Workshops on Conceptual Modeling Approaches for E-Business and the World Wide Web and Conceptual Modeling, pp.76-90

[28] Schwabe, D. ,Esmeraldo, L., Rossi, G., Lyardet, F., "Engineering Web Applications for Reuse," *IEEE MultiMedia*, vol. 8, no. 1, 2001, pp. 20-31

[29] Wong, K., "Toward Reusable and Evolvable Web Sites," *Proc. 1st Annual Workshop on Web Site Evolution*, (WSE'99), pp. 49 - 52

[30] Wong, T.W., Jarzabek, S., Myat Swe, S., Shen, R. and Zhang, H.Y. "XML Implementation of Frame Processor," *Proc. ACM Symposium on Software Reusability, SSR'01,* Toronto, Canada, May 2001, pp. 164-172

[31] Zhang, H. and Jarzabek, S., "An XVCL-based Approach to Software Product Line Development", *Proc. 15th Int. Conf. on Software Engineering and Knowledge Engineering* (SEKE'03), San Francisco, USA, 1 - 3 July, 2003

[32] XVCL (XML-based Variant Configuration Language) method and tool for managing software changes during evolution and reuse, http://fxvcl.sourceforge.net