Stable Analysis Patterns:

A True Problem Understanding with UML

- Proceedings-

UML 2003 Workshop # 8

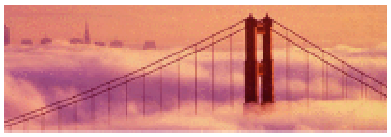
http://www.engr.sjsu.edu/~fayad/workshops/UML03

or

http://www.activeframworks.com/publications/workshops/UML03

In Association with the
6th International Conference on the Unified Modeling Language
<<UML>> 2003

San Francisco, California, USA -- October 20, 2003
http://www.umlconference.org



Mohamed E. Fayad, PhD, San Josè State University, U.S.A.
Haitham Hamza, University of Nebraska-Lincoln, U.S.A

University of Nebraska-Lincoln
Computer Science & Engineering Department
Technical Report No. 04-03-02
March 2004

Software Stability: Recovering General Patterns of Business

Mark Lycett, Chris Partridge and Sergio de Cesare
Department of Information Systems and Computing
Brunel University, Uxbridge, Middlesex. UB8 3PH.
United Kingdom.

Email: Mark.Lycett@brunel.ac.uk

Abstract

The software stability approach is required to balance the seemingly contradictory goals of stability over the software lifecycle with the need for adaptability, extensibility and interoperability. This workshop paper addresses the issue of how software stability can be achieved over time by outlining an approach to evolving General Business Patterns (GBPs) from the empirical data contained within legacy systems. GBPs are patterns of business objects that are (directionally) stable across contexts of use. The work explains, via a small worked example, how stability is achieved via a process of 'sophistication'. The outcome of the process demonstrates how the balance that stability seeks can be achieved.

1. Introduction

Change in the business environment, often in connection with changes in technology approaches, causes entire software programs to be reengineered – often at significant cost (see Fayad et al. 2001 for example). Typically, much reengineering is oriented toward meeting emerging adaptability, extensibility and interoperability requirements. The cycle of change can often be destructive in nature, however, as existing systems become more complex and increasingly difficult to understand. With this background, the notion of software stability has emerged as a means of identifying the enduring aspects of a system whilst acknowledging the need for adaptability and extensibility (see Fayad et al. 2002). Unsurprisingly, given the relative youth of the stability concept, many questions exist in relation to its purpose, role, achievement and the like.

This position paper addresses the issue of how software stability can be achieved over time by outlining an approach to evolving General Business Patterns (GBPs) – patterns of business objects that are (directionally) stable across contexts of use. The approach itself is rooted in developing patterns from the ***business*** knowledge that is embedded in existing software systems – and is based upon initial work described in Partridge (1996). The paper begins by noting the importance of the stability concept in the context of existing software systems. Section 2 outlines the motivation for the approach to stability and describes the approach to the evolution of GBPs, called sophistication. Section 3 describes the concept of sophistication. Section 4 provides an illustration of the sophistication process, via a condensed worked example, and describes how the process of evolution works. The paper concludes on the outcomes and focus of the exercise.

2. Stability and the Importance of Existing Systems

It is stated that stability, currently expressed via business objects and enduring themes, has the potential to reduce or eliminate the reengineering cycles commonplace in software engineering projects (Fayad et al. 2002). There is significant cost and effort associated with reengineering, and much of the literature in the area focuses on the problems that reengineering effort aims to overcome. These problems include observations that (a) many applications are not easily integrated and were built during a prior era's technology, (b) plans and documentation are either poor or non-existent and (c) the architecture is insufficiently flexible to meet the challenges of anticipated future change (O'Callaghan 1999).

Many existing systems, however, are now business critical – they deliver significant business value via the embodiment of substantial corporate knowledge in the form of business objects, processes, rules, events and the like. Moreover, the corporate knowledge that they embody is proven in the corporate working context on a daily basis. Given that many such systems have been in operation for years, we propose that the roots of stability lie in systems that exist and that reengineering efforts can be seen as a proactive means for mining aspects of stability. The difficulty with mining stability from existing systems is that we have not had the conceptual tools to undertake such work – it is this area that our effort addresses.

3. The Concept of Sophistication

The danger with a reengineering project is that it unlocks a system view of the business from one technology set, makes some changes to aspects of content and/or behaviour, and locks the revised view into another technology set. The effort does not generally consider what the system view of the business actually means from a business perspective (a pervasive problem underlined by a specification view of systems development). This consideration provides a key point of departure for understanding and generating business content that is stable.

Stability is achieved by applying a process of sophistication. In outline terms, sophistication may be defined as the process of improving a business model - by removing differences between it and the things described in the model that exist in the real (business) world: The aim is to provide better theories and a more precise representation of the world. Sophistication thus provides the underpinnings for stability and improvements lie along the following dimensions:

- Explanatory power. The ability of the improved model can give increased meaning to the things and the relationships expressed.
- Fruitfulness. The degree to which the improved model can meet currently unspecified requirements or is easily extendable to do so.
- Generality. The degree by which the scope of the types in the improved model can be increased without the loss of information.
- Objectivity. The ability of the model to provide a more objective (shared) understanding of the world.
- Precision. The ability of the improved model to give a more precise picture of the business object: in particular, to index a thing to its mode of existence as opposed to its mode of representation and/or application.

- Simplicity. The degree by which the model can be made less complex.

Analysis of existing systems leads to the observation of sophistication gaps, categorised as follows. For each category, techniques and heuristics have been developed to liberate a model along the dimensions above.

- Generalisation. Redundancy analysis of things to produce a smaller collection that is more general but which typically contains more inherent information.
- Interpretation. Index-based analysis where information is treated as data rather than as a representation in order to tie that data back to what is represented (understanding the mode of existence).
- Destratification. Relationship-based analysis of things to better express the natural overlaps between them, removing the constraints of artificial hierarchies.
- Temporalisation. Spatio-temporal analysis of things to express natural temporal stages (relationships between things in time) in a systematic fashion.

4. The Process and Outcomes of Sophistication

The identification of sophistication gaps and action taken in respect of them leads to the development of business objects and patterns of business objects, which we term General Business Patterns (GBPs). Sophistication gaps and actions are best understood in the context of a small example. Figure 1 provides an example of a fragment related to public holidays in an existing Insurance system. For reasons of confidentiality we will call the company 'Acme' and the system 'Isure'. Note that the types GE015 and GE740 refer to interface specifications.

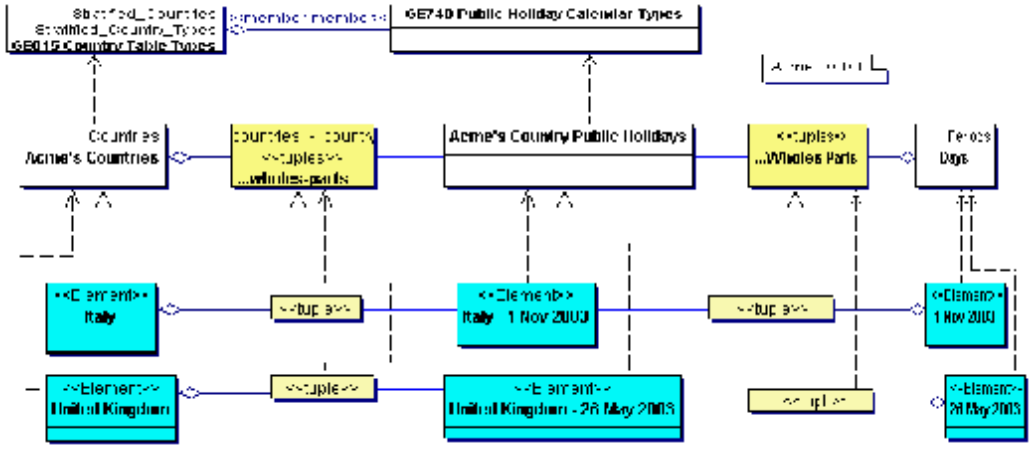


Figure 1. Acme System Fragment

The interface GE015 is implementation indexed – even though the application level description is identical, different implementations will have different tables (that are likely incompatible). This observation indicates that there is a common business pattern that links the application level with the different implementations (which can be identified as stratification). The interface GE740 is dependent on GE015 and thus inherits the index on implementation and the common business pattern (a kind of reflected stratification). GE740's HOLDTE (Holiday Date) attribute is constrained to representing only days, which restricts the kind of Public Holiday that GE740 can represent.

Given an understanding of the operational requirements if Isure, a number of competency questions were asked of the system that it could not accommodate:

CQ1. Can public holidays for both nesting (*e.g. United Kingdom – 16 May 2003*) and nested countries (*e.g. Northern Ireland – 13 March 2003*) be represented?

CQ2. Can non-country public holidays (e.g. *Valencia - 22 January 2003*) be represented?

CQ3. Can non-day public holidays (e.g. *Turkey - 10 February 2003 half-day*) be represented?

CQ4. Is it possible to recognise when a public holiday in a whole country (e.g. *United Kingdom - 26 May 2003*) impacts upon residents of a part country (e.g. *Northern Ireland*).

Sophistication was then undertaken in the following manner, dealing with one aspect at a time. First, country was destratified so that the countries and their public holidays from the different implementations could be combined. Second, countries were generalised so that non-country public holidays could be included. Last, days were generalised to periods so that non-day holidays could be included. The result is shown in Figure 2, which is a significant improvement in that the model is:

- More universal in that it is not implementation indexed as the original Isure system was – it will now work over multiple implementations.
- More general as it works at the level of Geo-Political Regions rather than stratified country and public holiday patterns.
- Simpler in that it no longer needs to deal with the stratified country and public holiday patterns.
- More precise in that it is (a) a more faithful representation of the common-sense notion of a public holiday and (b) does not include the spurious Isure public holidays stratification.

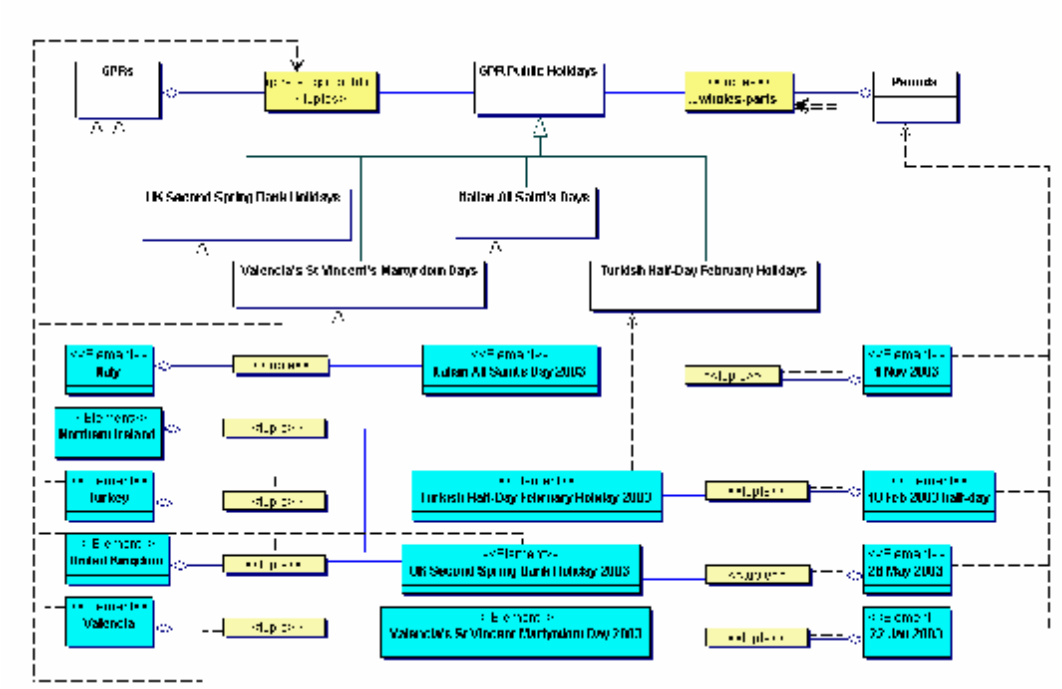


Figure 2. Generalised Public Holidays

GBPs emerge from a process of harmonising the many fragments of the system selected for analysis and both business objects and GBPs can be envisaged as (directionally) stable. In this particular case, Isure was examined against an existing GBP called Geo-Political Regions (GPR). From a GBP perspective, stability is achieved via a process of constant comparison. This approach has strong roots in the grounded theory approach (Glaser et al. 1967), where theory emerges from analysis of empirical data and is refined in the context of additional empirical data until a saturation point is reached (i.e., new data does not impact the model).

In this case, (a) theory can be considered as a model that is conceived of and taken to be true (Bhaskar 1978) and (b) empirical data is drawn from existing financial service systems. True equates to more sophisticated in this case and the notion of saturation needs to be considered from a pragmatic perspective. Consequently, we conceive of sophistication in terms of maturity levels – where a single GBP might be sophisticated enough to provide interoperability across several systems within an organisation, several systems across organisations and/or several systems across several domains for example.

5. Summary and Conclusion

Software stability concentrates on balancing the seemingly contradictory goals of stability over the software lifecycle with the need for adaptability and extensibility (Fayad et al. 2002). This workshop paper has outlined an approach for evolving stable business object and General Business Patterns from empirical reference data drawn from existing software systems. Using a small example of a sophistication instance we have part-illustrated the process of sophistication and described how the outcomes produce business objects and patterns that are (directionally) stable; and more explanatory,

fruitful, general, objective, precise and simple. The example has sought to illustrate fruitfulness and objectivity in particular.

The interested reader will have noted two things however: (1) we are operating at a computational independent level and (2) business content provides the focus of the work. The purpose of working at a computational independent level is one of ensuring that the outcomes can be tailored to any particular technology-set without being wed to it – that 'wedding' remains a challenge for both industry and academia. While the Model Driven Architecture (MDA) underpins our thoughts, the whole purpose of our approach is to liberate meaning in the business sense in order to ensure that systems represent the business in the most precise and universal manner possible. In that light, we have purposely sought to separate what there is (business content) from what the application does with it (application behavior) as a positive step in achieving adaptability and extensibility.

Our work related to application behavior is much less mature at this point in time, but is inherent not least in the fruitfulness of content – it is apparent that the model in Figure 2 works across multiple implementations and is rich enough semantically to both handle the existing system functionality and, at least, the functionality implicit in the competency questions above. In conclusion, it is our experience that stability models do demand a greater investment in analysis – there is significant potential, however, for achieving ongoing savings in development and maintenance costs (following Fayad et al. 2002).

References:

Bhaskar, R. *A Realist Theory of Science*, (2nd ed.) The Harvester Press, Sussex, 1978.

Fayad, M., and Altman, A. "An Introduction to Software Stability," *Communications of the ACM* (44:9) 2001, pp 95-98.

Fayad, M., and Wu, S. "Merging Multiple Conventional Models in One Stable Model," *Communications of the ACM* (45:9) 2002, pp 102-106.

Glaser, B.G., and Strauss, A.L. *The Discovery of Grounded Theory: Strategies for Qualitative Research* Weidenfeld and Nicholson, London, 1967.

O'Callaghan, A.J. "Migrating Large-Scale Legacy Systems to Component-Based and Object Technology: The Evolution of a Pattern Language," *Communications of the Association of Information Systems* (2:Article 3), 1999.

Partridge, C. *Business Objects: Re-Engineering for Reuse* Heinemann, Oxford, 1996.