Benutzeranforderungen als Zentrum und Treiber im Portalbauprozess

Lutz Prechelt

abaXX Technology AG
Forststr. 7
70174 Stuttgart
lutz.prechelt@abaxx.de

Abstract: Die Umsetzung eines Portals ist eine technisch schwierige und komplexe Aufgabe. Dennoch resultieren die schlimmsten Mängel des Endprodukts meist nicht aus technischen Fehlern, sondern von falsch oder unvollständig ermittelten oder umgesetzten Anforderungen. Dieser Beitrag beschreibt technische Maßnahmen, mit deren Hilfe die Anforderungsbestimmung beim Portalbau so mit der technischen Umsetzung verzahnt werden kann, dass solche Mängel weniger wahrscheinlich werden.

1 Erfolgsfaktor Anforderungen

Etwa ein Viertel aller Softwareprojekte werden unterwegs abgebrochen [Gi94], etwa drei Viertel(!) der versuchten Softwaresysteme kommen nicht zum Einsatz [Gi94][St95], weil sie nicht nutzbar oder nicht nützlich sind. Unter den diversen Gründen für diese hohe Misserfolgsquote taucht eine Gruppe immer wieder auf vorderen Rängen auf: Die Bestimmung der Anforderungen bzw. die Einbindung von Endbenutzern.

So tauchen in der bekannten Top 10 Risk List von Boehm [Bo91] gleich drei derartige Punkte auf: *Developing the wrong functions and properties*; *developing the wrong user interface* und *continuing stream of requirements changes*. Die sehr systematische Studie von Schmidt et al. [SLK01] findet unter ihrem Top 10 sogar fünf: *Failure to gain user commitment; misunderstanding the requirements; lack of adequate user involvement; lack of frozen requirements; failure to manage end user expectations.*

Wie seit längerem bekannt, liegt die einfachste und wirksamste Maßnahme gegen die meisten Aspekte dieser Risiken in der Umsetzung des Prinzips „Release early, release often“ [Ag01] und der zugehörigen engen Einbindung von Benutzern. Doch wie macht man das bei Portalprojekten mit ihrer unhandlichen Technologie und ihrer hohen Komplexität?

Die nachfolgenden Abschnitte stellen technologische Beiträge vor, durch die ein solches anforderungsorientiertes und risikoverminderndes Arbeiten unterstützt wird und skizzieren eine entsprechende Projektmethodik.

2 Web-GUI Prototyping

Der unverzichtbare Bestandteil jeglicher noch so frühen Release eines Portalprojekts (oder eines Prototyps) ist die Benutzungsschnittstelle. Zugleich liegen hier zahlreiche Möglichkeiten Endbenutzer nicht nur einzubinden, sondern zusammen mit ihnen viele wichtige Anforderungen zu erheben, korrigieren und vervollständigen und ihre Umsetzung benutzbar zu gestalten.

Folglich ist es in den meisten Portalprojekten zweckmäßig, mit einem Rapid Prototyping der Benutzungsschnittstelle und deren anschließender Validierung und Verbesserung zu beginnen.

Grundsätzlich sind HTML-basierte Schnittstellen dafür auch gut geeignet, weil sich mit HTML der visuelle Aspekt eines GUIs (samt „page flow“, also der Abfolge von Ansichten bei Bedienaktionen) recht schnell und einfach realisieren lässt und der HTML-Code bei der späteren wirklichen Implementierung teilweise wiederverwendet werden kann. Allerdings wird bei Portalprojekten die praktische Arbeit in dieser Hinsicht genau durch den Portalcharakter der Schnittstelle enorm erschwert: Es ist kennzeichnend für Portal-Webseiten, dass neben dem Element oder den Elementen, die für die gerade betrachtete Anforderung von direktem Interesse sind, zahlreiche andere Elemente im GUI vorhanden sind; erst dadurch entsteht der „Schaltzentrale“-Charakter der Portalseite. Naturgemäß werden die meisten solchen Elemente in zahlreichen verschiedenen Seiten (oder Ansichten) wiederverwendet, was reine HTML-Prototypen durch die dabei auftretende hohe Redundanz sehr umständlich macht. Außerdem muss es beim Prototypbau möglich sein, selektiv an einigen Stellen echte programmierte Funktionalität zu hinterlegen, was solche komponierten HTML-Prototypen schlagartig sehr kompliziert werden lässt.

Einen Lösungsansatz bieten Portlet-Technologien [Ja03], die ein Zusammensetzen einer Ansicht aus separat realisierten Einzelteilen erlauben. Wenn eigentlich nur HTML benötigt wird, ist Seitenbau mit Portlets aber leider relativ umständlich. Außerdem lösen Portlets nur einen Teil des Problems, denn Portlets sind Subapplikationen und auch im Innern solcher Subapplikationen gäbe es noch sehr viel Redundanz zu vermeiden. Da sich Portlets jedoch nicht verschachteln lassen, muss hier wieder zu anderen Mechanismen (wie JSP-includes) gegriffen werden, was den Aufwand weiter in die Höhe treibt. Drittens geht leider die Möglichkeit verloren, die Seitenabfolgelogik (page flow) mittels einfacher Hyperlinks zu simulieren.

Benötigt würde also eigentlich Folgendes:

1. Eine Basistechnologie für Teil-GUIs, ähnlich wie Portlets, jedoch möglichst leichtgewichtig und mit der Möglichkeit, die Teile ineinander zu verschachteln, damit sich die Redundanz minimieren lässt. Nennen wir solche Teile einmal *Parts*.
2. Ein Werkzeug, mit dem die zum Prototyping entscheidenden Schritte ohne manuelle Arbeit ausgeführt werden können: Ein neues Part definieren, ein Part auf einer Seite zusätzlich einsetzen (mit Auswahl aus dem Parts-Katalog) oder entfernen, das Arrangement von Parts auf einer Seite verändern und Page Flow

(genauer: Parts Flow, denn das meiste in der Ansicht bleibt ja oftmals gleich) definieren. Idealerweise sollte man diese Schritte direkt in der Portalansicht ausführen können.

Wie schon bei Portlets würde ein solcher Ansatz dazu führen, dass der Übergang vom statischen HTML-Prototyp zur (teil)funktionalen Implementierung gleitender wird, weil von Anfang an Strukturen eingerichtet werden, die dann bei der Implementierung nur noch mit Funktionslogik zu füllen sind.

Wie ein solches Werkzeug aussehen könnte, wird beispielhaft von dem in Abbildung 1 gezeigten abaXX PortalBuilder illustriert. Grundlage ist eine normale Ansicht des Portals im normalen Webbrowser. Der PortalBuilder äußert sich durch zwei zusätzliche Fenster, die mit im Browser angezeigt werden: Die Anatomiesicht, in der die Verschachtelung von Parts als Baum angezeigt oder verändert wird und den Inspektor, über den Parameter eines Parts angezeigt und verändert werden. Ferner wird in der Hauptansicht des Portals jedes Part mit einem gelben Namensschild in seiner oberen linken Ecke markiert. Die Anatomiesicht kann auf Galleriesicht umgeschaltet werden und zeigt dann die Bibliothek verfügbarer Parts an. Im Inspektor wird insbesondere die JSP-Datei eingetragen, in der sich das zum Part gehörige HTML- oder JSP-Fragment befindet. Beim Anlegen eines noch unbekannten Parts wird standardmäßig anfangs eine JSP benutzt, die lediglich die Beschreibung („comment“) der Partsdefinition anzeigt. Dieser Kniff ermöglicht eine Darstellung der Grobkonzeption einer Seite mit lauter noch nicht existierenden Parts in Minutenschnelle, gefolgt von einer schrittweisen Ausarbeitung mit HTML-Prototypen und/oder funktionalen Prototypen ganz nach Bedarf.

Diese technische Basis erleichtert es erheblich, dass ein Anforderungsingenieur Anforderungen, die sich auf der Ebene der Benutzungsschnittstelle ausdrücken lassen, erfasst und in Form funktionstüchtiger Prototypen festhält. Dies geschieht in direkter ständiger Zusammenarbeit mit einem oder mehreren Endbenutzern oder anderen Beteiligten, eventuell unterstützt von einem einzelnen Softwareingenieur. Das dabei entstehende Produkt kann bruchlos in eine komplette Implementierung überführt werden, denn sowohl sein technisches Substrat (Partdefinitionen und JSPs) als auch seine Modularisierung (als Parts-Katalog) haben genau die später im Projekt benötigte Form.

Entscheidend für den Erfolg sind dabei weniger irgendwelche Einsparungen beim initialen Aufbauen, sondern vor allem die Vereinfachung beim wiederholten Umbauen.

3 Explizite Ablaufmodellierung

Der oben skizzierte Ansatz zur Anforderungsbestimmung und –beschreibung findet seine Grenzen bei der Beschreibung der Arbeitsabbläufe, die durch die Software unterstützt werden sollen. Zum einen enden die Möglichkeiten, den page flow korrekt wiederzugeben immer dann, wenn eine Verzweigung nicht explizit vom Benutzer, sondern aufgrund einer internen Entscheidung in der Geschäftslogik nötig wird. Zum anderen lassen sich alle internen Abläufe, die nicht direkt an der Benutzungsschnittstelle widergespiegelt sind, überhaupt nicht angemessen behandeln. Wir brauchen also zusätzlich eine explizite Beschreibung der wichtigsten Abläufe im System und zwar in etwa der Granulari-

tät, die der Endbenutzer wahrnimmt: Seine Interaktionen mit dem System und dessen wesentliche Aktivitäten dazwischen.

Damit eine solche Beschreibung erfolgversprechend bezüglich der Benutzereinbindung und der Brauchbarkeit der so ermittelten Anforderungen ist, sollte der Mechanismus folgende Eigenschaften mitbringen:

1. Die Notation muss einem Endbenutzer verständlich sein (also bevorzugt grafisch und prozedural/kontrollflussorientiert; nicht textuell oder deklarativ)
2. Die Beschreibung sollte sich für die Interaktionsschritte explizit auf die geplanten oder bereits skizzierten (Teil-)GUIs abstützen.
3. Die Beschreibung sollte ausführbar sein, damit ein iteratives Prototyping und eine schrittweise Implementierung ähnlich wie bei den GUIs möglich wird. Es darf also insbesondere nicht gefordert sein, dass alle in der Beschreibung erwähnten Schritte bereits wirklich implementiert sind.

Wie ein passendes Werkzeug aussehen könnte, wird beispielhaft von dem in Abbildung 2 gezeigten abaXX WorkflowModeler illustriert. Jedes Kästchen ist eine Aktivität. Die mit einem Bildschirmsymbol gekennzeichneten bringen ein Part (samt kompletter Umgebungsseite) zur Anzeige, die übrigen repräsentieren interne Arbeitsschritte, implementiert zunächst durch einen Dummy, später durch eine Java-Klasse. Auch hier bilden die bereits angefertigten Aktivitäten eine Bibliothek. Verzweigungen können durch explizite Entscheidungen (nicht gezeigt) oder die von Parts ausgelösten Ereignisse beschrieben werden. Der nötige Datenfluss wird den Aktivitäten in separaten Dialogen hinterlegt. Eine solche Ablaufbeschreibung wird per XML direkt zum Server hochgeladen und dort von einer nach WfMC-Vorgaben [Wo97] gestalteten Workflow Engine abgearbeitet, so dass sie ebenso einem interaktiven Prototyping unterworfen werden kann wie die GUIs.

Eine so ausgestaltete Portalbausoftware ermöglicht eine enge und unaufwändige Kopplung zwischen Anforderungsermittlung und technischer Umsetzung, so dass die anforderungsbedingten Projektrisiken stark sinken. Zugleich liefert sie durch die Flexibilität der Strukturen eine hohe Agilität bei eventuellen Anforderungsänderungen.

Literaturverzeichnis

[Ag01] Agile Alliance: Manifesto for Agile Software Development, http://agilemanifesto.org, 2001.

[Bo91] Boehm, B.: Software Risk Management.: Principles and Practices. IEEE Software 8(1):32-41, Januar 1991.

[Gi94] Gibbs, W.W.: Software's chronic crisis. Scientific American 271(3):86-95, September 1994.

[Ja03] Java Community Process: Java Specification Request 168: Portlet Specification, http://jcp.org/en/jsr/detail?id=168, 2003.

[SLK01] Schmidt, R.; Lyytinen, K.; Keil, M.; Cule, P.: Identifying Software Project Risks: An International Delphi Study, J. of Mgmt. Information Systems 17(4):5-36, Spring 2001.

[St95] Standish Group: CHAOS Report, 1995, http://www.standishgroup.com.

[Wo97] Workflow Management Coalition, http://www.wfmc.org

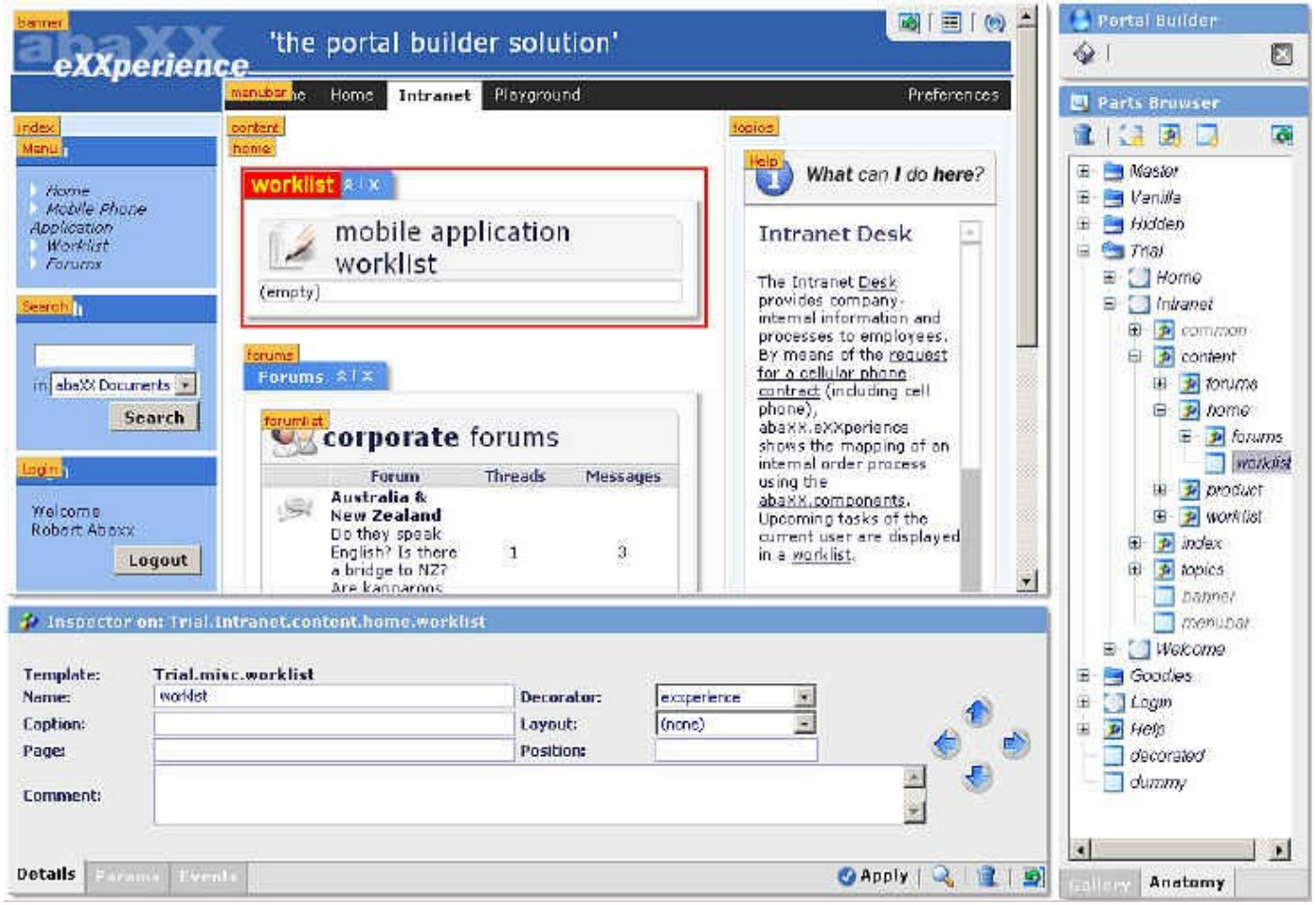

Abbildung 1: Mittelkomplexe Portalansicht (links) mit eingeblendetem PortalBuilder

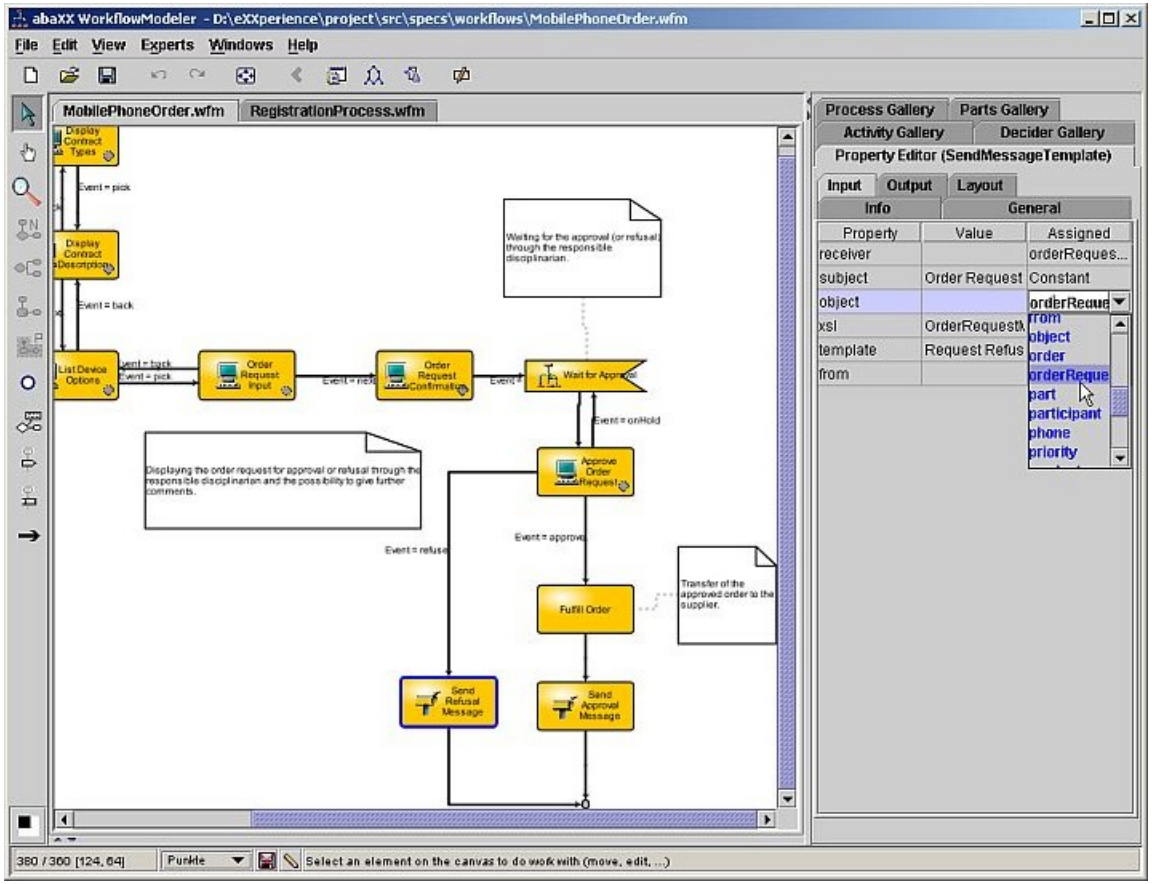

Abbildung 2: Einfache Ablaufbeschreibung, angezeigt im Workflow Modeler