Spago Overview

Redatto da Luigi Bellio
Gabriele Ruffatti (revisione)

Indice

Informazioni sulla versione

Versione/Release n° :	1.0	**Data Versione/Release :**	29/04/2003
Descrizione modifiche:	Prima emissione		
Versione/Release n° :	1.1	**Data Versione/Release** :	6/12/2004
Descrizione modifiche:	Indice ristrutturato, inserito il paragrafo "Caratteristiche di Spago", apportate alcune modifiche formali.		

Scopo del documento

Il documento intende fornire una panoramica sugli aspetti principali del framework *Spago*, con particolare evidenza dei principi architetturali e funzionali a cui si ispira. L'aderenza alle linee guida J2EE e l'uso dei *pattern,* sia architetturali che di progettazione, conferiscono al framework la flessibilità necessaria per soddisfare le esigenze specifiche di un applicazione Java interattiva.

Riferimenti

[1] S*un ONE Application Framework*, Sun Microsystems (2002)
[2] http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html
[3] Inderjeet Singh, Beth Stearns, Mark Johnson, *Designing Enterprise Applications with the J2EE Platform*, Addison-Wesley (2002)
[4] G. Flurry, W. Vicknair, *The IBM Application Framework for e-business*, International Business Machines Corporation (2001)
[5] *Struts Application Framework*, Apache Software Foundation
[6] M. E. Fayad, D. C. Schmidt, R. E. Johnson, *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, John Wiley & Sons, Inc., New York (1999)
[7] Mohamed Fayad, Douglas C. Schmidt, *Object-oriented Application Frameworks*, The Communications of ACM (1997)
[8] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Software Architecture*, Addison-Wesley (1995)
[9] Wolfgang Pree, *Design Patterns for Object-Oriented Software Development*, Addison-Wesley, Reading, MA (1994)
[10] G. Booch, *Object-Oriented Analysis and Design with Applications*, Addison-Wesley Longman, Reading, MA (1993)
[11] J. Rumbaugh, M. Blaha, W. Premerloni, F. Eddy, W. Lorensen, *Object-Oriented Modeling and Design*, Prentice-Hall, Upper Saddle River, NJ (1991)

Per ulteriori informazioni sul framework Spago si rinvia ai documenti disponibili in http://spago.eng.it/docs_it/documentation/index.html

1 Caratteristiche di Spago

Spago è un J2EE Framework: un'infrastruttura software riutilizzabile che può essere specializzata per produrre soluzioni cosiddette verticali. Non rende quindi disponibili delle funzionalità applicative ma un'architettura ed un insieme di servizi che ne facilitano la realizzazione. E' costituito da un insieme di servizi e componenti espressi ad un livello di astrazione tale che ne consenta un agevole riutilizzo in più progetti di sviluppo software.

Spago e' stato sviluppato con il contributo di una community di progettisti e sviluppatori con competenze ed esperienze diverse al fine di confezionare una soluzione che risponde ad esigenze d'erogazione di servizi applicativi multicanale e d'integrazione verso infrastrutture esterne: consente di realizzare un applicativo web che integra infrastrutture già in esercizio (sicurezza, document management, workflow, ecc.) e contemporaneamente pubblica servizi su canali differenti.

Il paradigma seguito per l'implementazione dell'architettura è il Model-View-Controller che obbliga ad una chiara suddivisione di responsabilità fra i seguenti layer:

- **Pubblicazione** : HTTP su web container, SOAP, WAP, EJB (a breve: HTTP su portlet container, TCP/IP)
- **Logica applicativa** : controlli, elaborazioni
- **Accesso ed integrazione** verso fonti informative e/o transazionali.

Il framework e' stato progettato in modo da rendere indipendenti i layer sottostanti alla pubblicazione da ogni componente specifico del canale utilizzando l'XML quale supporto alla comunicazione (ad esempio, la logica applicativa viene sviluppata in modo indipendente dagli oggetti HttpRequest ed HttpSession, specifici del canale HTTP) .

I punti di forza di Spago sono:

- **Multicanalità** : il framework consente in modo semplice di erogare i servizi indistintamente sul canale HTTP, WAP, SOAP ed EJB
- **Modalità di dispatching a moduli** : è una modalità più complessa di quella ad action che permette grande flessibilità ed elevato potenziale di riutilizzo del codice
- **Modalità di pubblicazione** : è possibile scegliere, tramite operazioni di configurazione, diverse modalità di pubblicazione delle informazioni in accordo con il canale su cui viene erogato il servizio
- **Distribuzione della logica di business** : con azioni a livello di configurazione è possibile scegliere se i servizi devono essere eseguiti dal web container o dall'EJB container, con rilevanti differenze a livello di gestione delle transazioni. Questo perchè il framework comprende un *session facade* a cui può essere delegata l'esecuzione di tutti i servizi. Dal punto di vista del codice non ci sono particolari differenze tra le due modalità, al punto che la modalità di esecuzione può, in alcuni casi, essere decisa al momento del deploy
- **Gestione della navigazione** : è un servizio che semplifica notevolmente la gestione delle navigazioni, anche se aumenta l'occupazione di memoria della sessione

- **Paginazione** : sono presenti alcuni moduli per la gestione di liste e dettagli che consentono una gestione semplice di questo tipo di servizio
- **Gestione flussi XML** : è disponibile un oggetto per la gestione efficiente di flussi XML (lettura/creazione/ricerca di attributi/valori).
- **Validazione** : è disponibile un modulo per la validazione lato server dei dati provenienti dai form. Il comportamento di questo modulo si controlla da file di configurazione, e prevede i tipi dati più comuni. E' estensibile tramite classi Java di validazione.

2 J2EE Framework

Un Java Enterprise Framework è un'infrastruttura software riutilizzabile che può essere specializzata per produrre soluzioni cosiddette "verticali"; non rende quindi disponibili delle funzionalità applicative ma un'architettura ed un insieme di servizi che ne facilitano la realizzazione. In particolare fornisce: un modello architetturale, applicativo e di sviluppo del software; un insieme di servizi generici presenti in qualsiasi soluzione gestionale; linee guida, standard, strumenti e metodi utili sia per organizzare lo sviluppo che le successive fasi di manutenzione.

I vantaggi fondamentali di un application framework possono essere ricondotti a:

- ***Modularità*** – i dettagli implementativi vengono nascosti da interfacce stabili. La localizzazione degli impatti dovuti ai cambiamenti della progettazione od implementazione migliora la qualità del software e riduce lo sforzo richiesto per comprendere e mantenere quello esistente.
- ***Riusabilità*** – la stabilità delle interfacce fornite da un framework consente la definizione di componenti generici che possono essere riutilizzati per creare nuove applicazioni. Può essere evitata la creazione e validazione di soluzioni comuni a requisiti applicativi e attività di progettazione del software ricorrenti. Il riuso di componenti di un framework può portare ad un sostanziale miglioramento della produttività dell'attività di programmazione, oltre che ad un aumento della qualità, delle prestazioni, dell'affidabilità e dell'interoperabilità del software.
- ***Estensibilità*** – le interfacce stabili di un framework possono essere estese dalle applicazioni attraverso l'uso di espliciti metodi "*hook*". Tali metodi disaccoppiano le interfacce stabili ed i comportamenti di un dominio applicativo dalle variazioni richieste da un applicativo in un contesto particolare. L'estensibilità di un framework è essenziale per assicurare la personalizzazione di nuovi servizi e caratteristiche applicative.
- ***Inversione del Controllo*** – l'architettura di un framework è caratterizzata da un'*inversione del controllo*. Alcuni passi standard di elaborazione di un applicazione possono essere personalizzati da gestori che vengono invocati attraverso il meccanismo di *dispatching* del framework. Al verificarsi di un evento, il *dispatcher* del framework invoca i metodi "*hook*" di gestori pre-registrati, i quali eseguono elaborazioni sugli eventi specifiche dell'applicazione. L'*inversione del controllo* consente al framework (e non all'applicazione) di determinare quale insieme di metodi specifici dell'applicazione devono essere invocati in risposta ad eventi esterni.

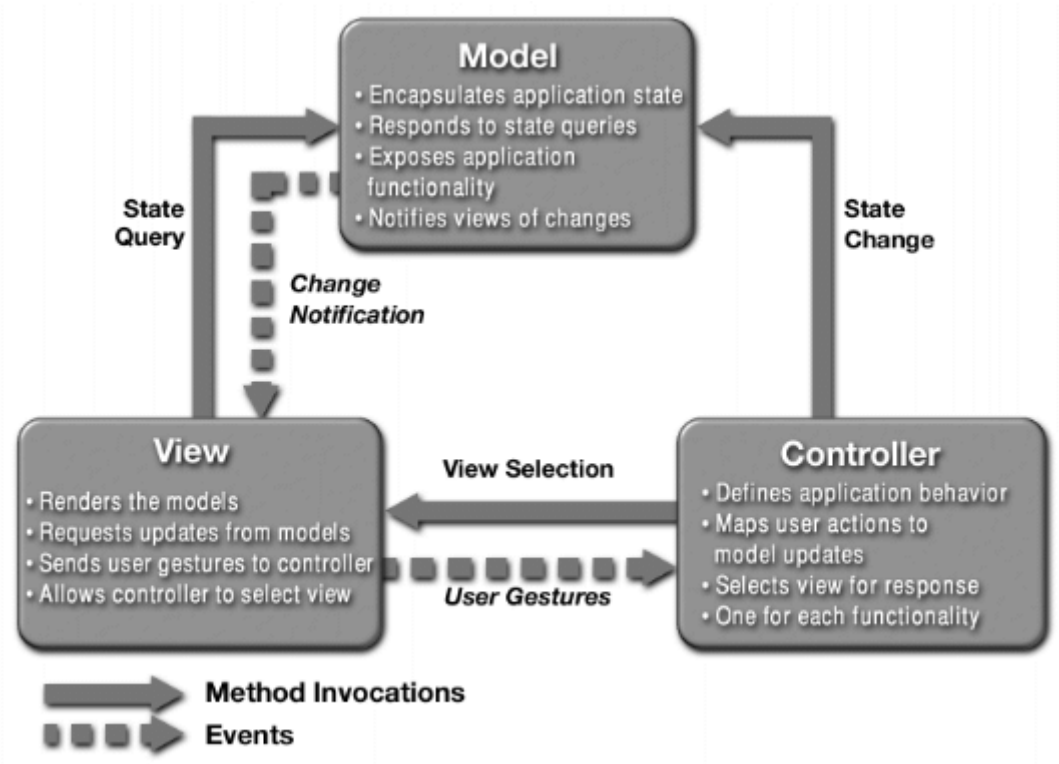
Un framework può inoltre essere catalogato in base alle tecniche utilizzate per estenderlo:

- ***Whitebox Framework*** – raggiunge l'estensibilità attraverso un uso intensivo di alcune caratteristiche dei linguaggi *object-oriented*, quali ereditarietà e *binding* dinamico. Le funzionalità esistenti vengono riutilizzate ed estese ereditando dalle classi base del framework e sovrascrivendo i metodi "*hook*" predefiniti tramite *pattern* come *Template Method*.
- ***Blackbox Framework*** – supporta l'estensibilità con la definizione di interfacce per componenti che possono essere inseriti nel framework attraverso la composizione e la delega di oggetti. Le funzionalità esistenti vengono riutilizzate con la definizione di componenti conformi ad una particolare interfaccia e con l'integrazione di questi componenti nel framework tramite *pattern* come *Strategy* e *Functor*.

3 Model View Controller

Model View Controller è il *design pattern* architetturale raccomandato dalle linee guida di Sun Microsystems per implementare applicazioni interattive. L'applicazione viene organizzata in tre moduli separati:

- ***Model*** – per la rappresentazione del modello di dominio e l'esecuzione della logica di *business*.
- ***View*** – per la presentazione dei dati e la gestione dell'*input* utente
- ***Controller*** – per il *dspatching* delle richieste ed il controllo di flusso



Il pattern *Model-View-Controller*
(fonte: http://java.sun.com/blueprints/patterns/MVC-detailed.html)

L'architettura MVC separa gli obiettivi della progettazione, riduce la duplicazione del codice, centralizza il controllo, e rende l'applicazione più facile da modificare. Sviluppatori con *skills* diversi possono concentrarsi sulle loro specifiche competenze (sviluppo di *custom tags*, presentazione, logica applicativa, accesso ai dati) e collaborare attraverso interfacce chiare e definite. Inoltre, può essere centralizzato il controllo su alcune caratteristiche dell'applicazione come la sicurezza, il *logging*, il flusso fra le viste. Le attività fondamentali eseguite alla ricezione di una richiesta, sono: interpretazione della richiesta, instradamento della richiesta alla logica di *business*, selezione e generazione della successiva vista.

L'implementazione tipica del *pattern* MVC con tecnologia J2EE nella costruzione di un *application framework web* viene comunemente indicata con il nome *Modello 2*, in cui una *servlet*, con responsabilità di *Front Controller* e *Mediator*, gestisce la comunicazione con il *client* e l'esecuzione della logica di *business*, mentre la presentazione e realizzata principalmente attraverso pagine JSP. Il termine *Modello 1* viene invece usato per indicare un'architettura in cui il *client* invoca direttamente le pagine JSP senza la mediazione di un *controller*.

Nel M*odello 2* la *servlet* di controllo centralizza la logica per l'instradamento delle richieste basata sull'URL della richiesta, i parametri di input, e lo stato dell'applicazione. Il *controller* gestisce anche la selezione della vista, disaccoppiando le pagine JSP l'una dall'altra. Inoltre, la *servlet* di controllo fornisce un unico punto per la gestione della sicurezza ed il *logging* e, spesso, trasforma i dati in *input* in una forma adatta per il modulo *Model*. Per l'implementazione del modulo *View* vengono tipicamente usate pagine JSP per produrre contenuti di tipo testuale come HTML o XML, mentre sono preferibili le *servlet* per generare contenuti binari (RTF, PDF) o a struttura variabile. Il modulo *Model,* che rappresenta sia i dati che la logica di *business,* può essere implementato con *Enterprise JavaBean* per soddisfare requisiti come scalabilità, concorrenza, bilanciamento del carico, gestione automatica delle risorse, accesso a logica e dati di *business* condivisi, oppure con *JavaBean* standard per un accesso ai dati semplice e meno critico.

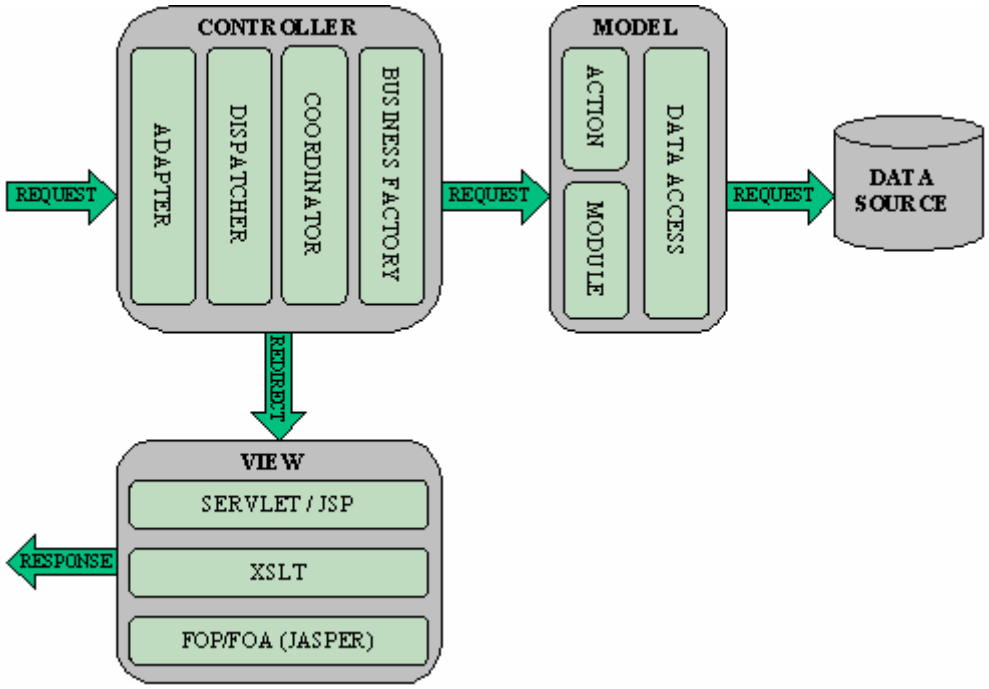
La separazione della logica di *business* dalla presentazione porta a diversi vantaggi:

- ***Minimizza gli impatti dei cambiamenti*** – le regole di *business* possono cambiare con modifiche minime (od assenti) al modulo di presentazione. Viceversa, la presentazione od il *workflow* dell'applicazione può cambiare senza influenzare il modulo di *business*.
- ***Aumenta la manutenibilità*** – la modifica, ad esempio, dei componenti di *business*, delle direttive di *include* lato *server*, di *custom tag*, *stylesheet*, altera il comportamento dell'applicativo in qualunque punto essi vengono referenziati.
- ***Fornisce indipendenza dal client e riuso del codice*** – la logica di *business,* disponibile come semplici chiamate a metodi di oggetti di *business,* può essere utilizzata da diversi tipi di *client*.
- ***Consente la separazione dei ruoli degli sviluppatori*** – la separazione tra logica di *business* e presentazione consente agli sviluppatori di concentrarsi sulla loro area di competenza: presentazione dei dati, elaborazione della richiesta, regole di business.

4 Architettura di Spago

Spago è sviluppato secondo il *pattern* architetturale MVC per lo sviluppo di applicazioni Java in cui l'interazione con il *client* può avvenire su più canali/protocolli. I *Front Controller* sono implementati come collaborazione dei seguenti oggetti:

- ***Adapter*** – ha la responsabilità di ricevere i dati della richiesta da uno specifico canale, trasformare i parametri nel formato adatto al modulo *Model*, scegliere la vista corretta. Esegue inoltre il *binding* del contesto conversazionale sulle risorse del *container* specifico
- ***Dispatcher*** – ha la responsabilità di identificare la modalità di esecuzione della logica di *business* fra quelle supportate e di ritornare il corretto coordinatore
- ***Coordinator*** – ha la responsabilità di coordinare l'esecuzione della logica di *business* secondo una specifica modalità
- ***Business Factory*** – ha la responsabilità di recuperare i riferimenti agli oggetti di *business* che cooperano all'esecuzione della richiesta.



Gli *adapter* disponibili sono:

- ***AdapterHTTP (HTML/HTTP)*** – una *servlet* che gestisce le richieste provenienti da un *client* HTTP, come un browser web o un dispositivo WAP
- ***AdapterSOAP (XML/HTTP)*** – un oggetto, registrato come *end-point* SOAP, che gestisce le richieste provenienti da un *client* SOAP
- ***AdapterEJB (XML/IOOP)*** – un *session bean stateful* che gestisce le richieste provenienti da un *client* IIOP

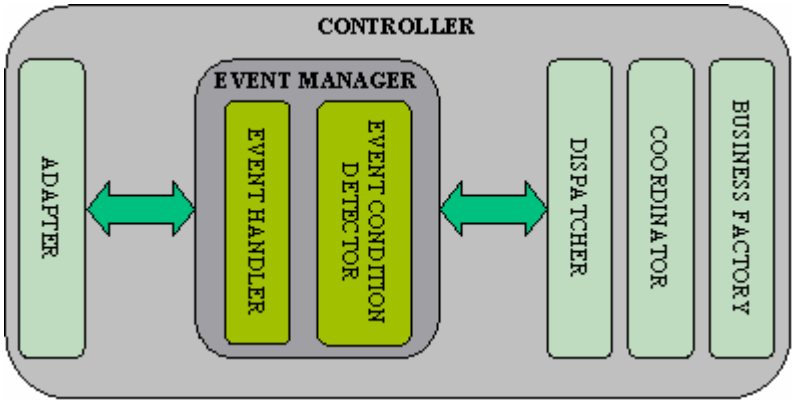
- ***AdapterJMS (XML/JMS)*** – un *message driver bean* che gestisce le richieste inviate come messaggi JMS

La logica per la scelta della vista corretta in base al canale, ai parametri della richiesta, allo stato dell'applicazione è censita come elemento di configurazione del framework. Oltre a gestire viste i cui contenuti vengono pubblicati mediante JSP e *servlet*, tipico del canale *web*, il framework è in grado di eseguire automaticamente la trasformazione XML/XSLT dei dati prodotti dalla logica di *business*. Quest'ultima è l'unica modalità di pubblicazione valida per tutti i canali.

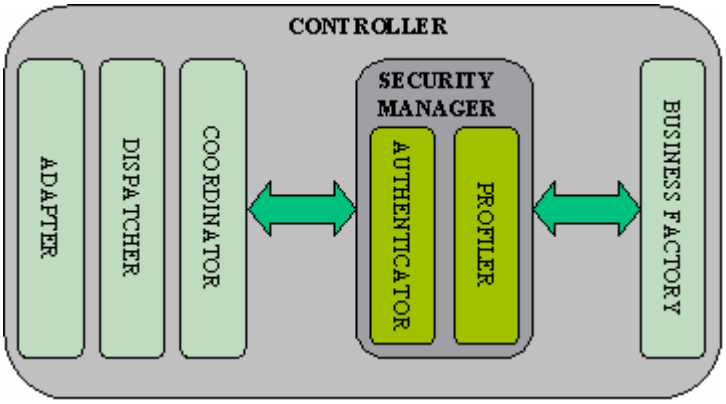
I *dispatcher* vengono registrati nel framework con un meccanismo a *plug-in* e costituiscono uno dei principali elementi di personalizzazione del framework. I *dispatcher* registrati di default sono:

- ***ActionDispatcher***: – verifica che la modalità di esecuzione della logica di business sia *action*, in cui si fa corrispondere ad una richiesta un singolo oggetto di business. Il coordinatore del servizio è l'*ActionCoordinator* il quale richiede il riferimento della *Action* alla *factory ActionFactory*.
- ***ModuleDispatcher***: – verifica che la modalità di esecuzione della logica di business sia *module*, in cui si fa corrispondere ad una richiesta più oggetti di *business* che collaborano fra loro. Il coordinatore del servizio è il *ModuleCoordinator* il quale richiede il riferimento dei *Module* alla *ModuleFactory*. La logica di collaborazione fra i *module* viene descritta attraverso un *workflow*.

Un'ulteriore elemento di personalizzazione del framework è costituito dal modulo *Event Manager* che consente di censire delle condizioni sullo stato di esecuzione di una richiesta a cui associare dei gestori. E' possibile ad esempio configurare gestori di *accounting* e notifica al verificarsi di specifiche condizioni sui dati di richiesta e di risposta del servizio invocato.



A livello di modulo *Controller* può essere configurato il *Security Manager* affinché verifichi i privilegi di esecuzione della logica applicativa associata ad un oggetto di *business*. La verifica delle autorizzazioni può essere abilitata su tutti gli oggetti di *business* o su alcuni di questi.



5 Dispatching della richiesta

Gli *step* eseguiti dal framework per evadere una richiesta sono:

- ***Request adapting*** – la struttura contenente i parametri della richiesta viene trasformata dal formato nativo del canale al formato multicanale interno.
- ***Session binding*** – viene recuperato lo stato conversazionale associato alla richiesta corrente.
- ***Request context building*** – viene costruito il contesto della richiesta composto dai parametri di input, il contesto conversazionale, il gestore degli errori e da alcuni parametri specifici del canale.
- ***Dispatcher detecting*** – viene identificato il *Dispatcher* corrispondente alla corretta modalità di esecuzione della logica di *business*.
- ***Business logic forwarding*** – il *Dispatcher* ritorna il *Coordinator* a cui viene delegata l'esecuzione della logica applicativa. A seconda della modalità configurata per la richiesta corrente, il *Coordinator* delega a uno o più oggetti di business l'implementazione del servizio. Due modalità, *action* e *module*, sono disponibili con il framework; altre possono essere implementate e configurate per personalizzare il framework alle specifiche esigenze.

5.1 Modalità Action

Oggetti di *business* di questo tipo evadono per intero la richiesta di un servizio all'applicativo. In pratica, esiste una corrispondenza univoca fra servizio ed oggetto di *business* (lo stesso oggetto di *business* può invece essere usato per evadere richieste diverse). Ogni *action* è inoltre caratterizzata da uno *scope*, indicando con questo termine l'ambito di vita dell'oggetto, che può valere:

- ***Request*** – viene istanziata una nuova *action* ad ogni nuova richiesta.
- ***Session*** – per tutte le richieste appartenenti alla stessa conversazione (sessione) il servizio viene evaso dalla stessa *action*.

- ***Application*** – per tutte le richieste indirizzate al medesimo *container* (JVM) il servizio viene evaso dalla stessa *action*. E' importante sottolineare che un oggetto di *business* con tale *scope* non rappresenta un vero *singleton*, in quanto ne esiste una diversa istanza per ogni JVM come ad esempio nei nodi di un *cluster*.

5.2 Modalità Module

Ogni *module* è un oggetto di *business* che può collaborare con altri *module* per evadere la richiesta di un servizio. Tutti i *module* che collaborano per lo stesso servizio formano un'unità logica chiamata *page*. La risposta al servizio invocato è rappresentata dall'unione delle risposte di tutti i *module*. Attraverso un semplice *workflow* di esecuzione della logica applicativa viene descritto in quale ordine e sotto quali condizioni vengono invocati i *module* che formano una *page*. Al termine dell'esecuzione di ogni *module* viene calcolato quali sono i successivi *module* da invocare e con quali parametri di richiesta. L'insieme delle regole che verifica se esiste una transizione fra un *module* ed un altro viene chiamato *conditions*, mentre l'insieme delle regole che determinano i parametri di richiesta di un *module* viene chiamato *consequences*. Chiamiamo invece *dependencie* la relazione per cui successivamente all'esecuzione di un *module* sorgente si verifica un dato insieme di *conditions* e viene invocato un *module* destinatario con i parametri di richiesta recuperati in base ad un dato insieme di *consequences*.

L'esecuzione della logica di *business* secondo questa modalità, se da un lato richiede un maggior numero di informazioni da censire, dall'altro consente di identificare degli oggetti con precise responsabilità, riutilizzabili nell'esecuzione di più servizi.

6 Logica di presentazione

I contenuti prodotti dalla logica di business vengono rediretti al modulo *View* che gestisce la pubblicazione in modo congruente con il canale da cui proviene la richiesta. Se nessun *publisher* è configurato per una data richiesta proveniente da un particolare canale il framework ritorna al *client* la rappresentazione XML dei contenuti. Questo consente, durante lo sviluppo di un'applicazione, di verificare la corretta esecuzione della logica di *business* anche in assenza di un'implementazione del modulo *View*.

Per tutti i canali disponibili è possibile applicare uno o più *stylesheet* alla rappresentazione XML dei contenuti prodotti. Per i soli canali *web* e *WAP* invece è possibile gestire la pubblicazione attraverso JSP o *servlet*. Il framework inoltre mette a disposizione alcuni *custom-tag* per gestire il *rendering* di contenuti particolari quali una collezione di oggetti pubblicata come lista paginata, un dettaglio di uno degli oggetti della collezione, la visualizzazione dei messaggi utente.

7 Servizi trasversali

I servizi e componenti trasversali a tutti i moduli sono:

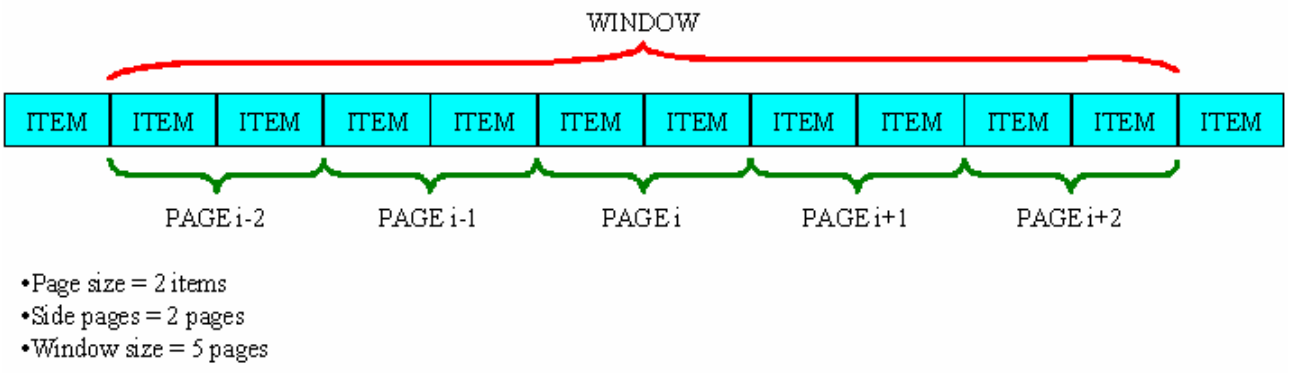
- ***Tracer*** – per la memorizzazione su *file* dell'attività applicativa; ad ogni messaggio viene inoltre assegnata una *severity* che ne discrimina l'effettiva memorizzazione a seconda del livello minimo di *tracing* configurato.
- ***Error Handler*** – ha la responsabilità di mantenere uno *stack* degli errori applicativi e non; ad ogni errore viene inoltre assegnata una *severity*; due tipologie di errori sono state identificate al momento:
  - ***User Error*** – errori riscontrati nella logica applicativa ed identificati da un codice; tale codice viene risolto a *runtime* con una descrizione catalogata dell'errore.
  - ***Internal Error*** – errori generati da sottosistemi terze parti (quali ad. es. l'*SQLException* di un attività JDBC).
- ***Request Container*** – ha la responsabilità di mantenere le informazioni il cui contesto è la specifica richiesta di un servizio (per il canale HTTP ha la stessa vita dell'oggetto *HttpServletRequest*).
- ***Session Container*** – ha la responsabilità di mantenere le informazioni il cui contesto è la conversazione del *device* con l'applicativo (per il canale HTTP ha la stessa vita dell'oggetto *HttpSession*).
- ***Application Container*** – ha la responsabilità di mantenere le informazioni il cui contesto è l'applicativo stesso; tale componente trasversale alle conversazioni con i *device* assume fondamentalmente il significato di una *cache* (è infatti possibile specificare un tempo di scadenza degli oggetti in esso mantenuti).
- ***Configuration*** – ha la responsabilità di mantenere i dati di configurazione dell'applicativo censiti su *file* XML.
- ***InitializerManager*** – ha la responsabilità di coordinare le attività di inizializzazione dell'applicativo nella fase di *bootstrap*.
- ***Navigator*** – ha la responsabilità di mantenere lo *stack* delle richieste dei servizi evasi consentendo la riesecuzione di un precedente servizio nelle medesime condizioni di *Request Container* e *Session Container*. Consente, ad esempio, di sostituire il *refresh* ed il *back* di un *browser* con una gestione applicativa e più sofisticata delle due funzionalità.

8 Automatismi

Il framework mette a disposizione alcuni componenti per implementare funzionalità riutilizzabili da diverse applicazioni interattive:

- ***Paginazione*** – una collezione di oggetti qualsiasi, recuperati ad esempio da un database, da un directory server o costruiti internamente all'applicativo, può essere suddivisa (paginata) in sottoelenchi (pagine) in cui il meccanismo di *caching* del paginatore è configurabile per rispettare il compromesso fra occupazione di risorse

(quale la memoria necessaria per mantenere la collezione) e i tempi di risposta per il recupero di una pagina (numero di *query* eseguite verso il *data-source*).



Il paginatore mantiene nella *cache* una finestra di osservazione (*window*) composta da un certo numero di pagine (2 * *side pages* + 1) e ricarica i dati quando deve ritornare una pagina non contenuta nella finestra. La nuova finestra sarà centrata sull'i-esima pagina richiesta.

- ***Generazione automatica lista e dettaglio*** – è possibile censire a livello di configurazione del framework le informazioni necessarie a recuperare una collezione di righe da una tabella (vista, *stored-procedure* o più in generale un *resultset*) ed a descrivere le caratteristiche per la pubblicazione della lista paginata.

Allo stesso modo può essere descritta sia la logica di business, sia quella di presentazione, per gestire la pubblicazione del dettaglio di un elemento della lista.