Sommario

1 Scopo del documento

Lo scopo del presente documento è quello di fornire le informazioni principali necessarie per iniziare a lavorare con gli MDDTools:

- avere una panoramica delle funzionalità messe a disposizione dagli strumenti che costituiscono la suite
- installare ed iniziare a lavorare con gli strumenti

Verranno inoltre fornite alcune indicazioni aggiuntive utili per l'utilizzo degli strumenti MDDTools in contesti di sviluppo differenti dal CSI-Piemonte.

2 Suite MDDTools: overview degli strumenti a disposizione

In questo capitolo verrà fornita una panoramica introduttiva degli strumenti forniti dalla suite *MDDTools*.

2.1 Introduzione

MDDTools è una suite di strumenti per lo sviluppo di applicazioni gestionali basata su tecniche Model Driven.
La suite di presenta come un insieme di *plugin* integrati nel workbench *eclipse* che permettono la modellazione e la successiva generazione del software

2.2 GUIGEN: Sviluppo Model driven di applicazioni end-user

2.2.1 Introduzione

GUIGEN è un generatore **model-driven** di front-end applicativi. Permette la generazione di applicazioni a partire da un modello dichiarativo dell'applicazione e con l'aggiunta manuale di codice custom a completamento della logica di business (principalmente per la realizzazione dell'interazione con il backend, con servizi esterni o con risorse quali DBMS).

Il modello che descrive l'applicazione è:

- basato su un metamodello apposito (descritto in dettaglio nella reference guide disponibile in eclipse dopo l'installazione dei plugin);
- strutturato in un insieme di modelli interconnessi e che descrivono i vari aspetti dell'applicazione (la struttura delle dipendenze è descritta in dettaglio nella guida *online* disponibile in eclipse dopo l'installazione dei plugin);

Il modello così definito ricopre vari aspetti di una applicazione:

- caratteristiche strutturali
    - struttura delle schermate
    - struttura dei menu delle funzionalità

- caratteristiche dinamiche
    - flusso tra le schermate
        - eventi associati ai vari componenti di controllo (menu, pulsanti, etc.)
        - azioni (comandi) scatenate a fronte di un evento su un componente di controllo
        - modifica stato dei widget (disabilitazione/visibilità, ...)
- caratteristiche di binding dati
    - definizione di strutture dati (model)
    - binding bidirezionale tra widget e dati del model

Il metamodello secondo il quale occorre modellare l'applicazione è indipendente da una particolare modalità/tecnologia di realizzazione dell'applicazione, utilizzando concetti generali (es. widget, event-handler, panels) e non **technology-specific** (es. submit, JTree, ...): di conseguenza il generatore GUIGEN può in linea teorica generare un'applicazione modellata secondo tale metamodello utilizzando differenti tecnologie di implementazione.

Attualmente il target di generazione è rappresentato da una web-application J2EE compliant con l'attuale framework applicativo basato su tecnologie Servlet/JSP e framework struts2. Per approfondimenti circa il mapping tram modello GUIGEN e piattaforma specifica (ovvero per comprendere meglio le regole di generazione del codice a fronte del metamodello, è possibile consultare la sezione “design rationale” disponibile nell'help online di eclipse dopo l'installazione del plugin.

2.2.2 Funzionalità

Di seguito sono elencate le principali funzionalità messe a disposizione dallo strumento GUIGEN.

2.2.2.1 Definizione della struttura della User Interface dell'applicativo

La U.I dell'applicazione è organizzata in alcune aree prinicpali:

1. un **header** e un **footer** che contengono informazioni statiche non gestite da applicativo (che in genere sono determinate dalla collocazione di un applicativo in un particolare contesto (es. sito web) e che non varia da applicazione ad applicazione
2. una area applicativa **ApplicationArea** che dipende dalla particolare applicazione. A sua volta questa area applicativa è strutturata in:
    - una struttura a menu (**MenuBar**)
    - un'insieme di schermate applicative (**ContentPanel**) che si avvicendano a fronte del flusso funzionale dell'applicativo e sono visualizzate all'interno dell'area invariante (header, footer). Il menu è gestito come widget in modo da poter essere collocato in tutte le schermate.

Ciascuna schermata contiene **Widget** di vario genere strutturati in pannelli (**Panel**) che possono essere dei seguenti tipi:

- **CommandPanel**: tipo specializzato di **Panel** che può contenere solo **widget** di tipo **Command** (pulsanti)
- **DialogPanel**: pannello utilizzato per la visualizzazione di popup
- **FormPanel**: pannello destinato a contenere esclusivamente **sottopannelli**
- **MenuPanel**: tipo specializzato di **Panel** che può contenere solo **widget** di tipo menu o affini (attualmente solo **MenuView** o **TreeView**).
- **MsgBoxPanel**: pannello utilizzabile per la visualizzazione di messaggi o html custom (può contenere soloun tipo di widget: il **PlainText**
- **MultiPanel**: Pannello mutevole: può assumere differenti forme in base ai sottopannelli contenuti (il contenuto di ciascun sottopannello è a sua volta un **pannello**)
- **PanelDefUse**: Utilizzo di pannello definito esternamente: permette la definizione di frammenti di U.I. riusabili in più modelli / punti del modello.
- **StdMessagePanel**: pannello di visualizzazione di messaggi (di errore ed informativi)
- **TabSetPanel**: il classico pannello a tab (il contenuto di ciascun tab è a sua volta un **pannello**)
- **UserInfoPanel**: pannello utilizzato pe rmostrare in modo standard le informazioni relative all'utente collegato
- **UserDefinedPanel**: pannello altamente personalizzabile (inclusione di sorgente hand written) da utilizzare in casi molto particolari
- **WidgetsPanel**: tipo specializzato di **Panel** destinato a contenere esclusivamente **widget**
- **WizardPanel**: implementazione di wizard (il contenuto di ciascuno step è a sua volta un **pannello**)

Ciascun singolo pannello prevede uno schema di layout (selezionabile in un insieme di possibilità) secondo cui i sottopannelli e i widget vengono disposti:

- Disposizione a cascata verticale su un'unica colonna (**VerticalFlowPanelLayout**);
    - applicabile a sottopannelli e widget;
    - l'ordine di visualizzazione è dall'alto in basso a seconda dell'ordine in cui i componenti appaiono nel modello (non sono necessarie delle **layout specification** aggiuntive);
- Disposizione in sequenza orizzontale **left-to-right** su un'unica riga (**HorizontalFlowPanelLayout**)
    - applicabile a sottopannelli e widget;

    - l'ordine di visualizzazione è dasinistra a destra a seconda dell'ordine in cui i componenti appaiono nel modello (non sono necessarie delle **layout specification** aggiuntive;
- Disposizione a 5 quadranti (**UpDownLeftRightCenterPanelLayout**)
    - applicabile al pannello principale, al command panel (in questo caso sono disponibili solo le posizioni Left e Right e servono per implementare plusantiere di navigazione del tipo indietro-prosegui) e ai FormPanel (disponibile solo per le cartucce di layout "neutral" e "nuova rupar");
    - può contenere al massimo 5 sottopannelli, 1 al massimo per sezione, ed è necessario specificare esplicitamente per ciascun elemento il quadrante di destinazione (mediante una **UDLRCWidgetLayoutSpec**);
- Disposizione a griglia **n**X**m** (**GridPanelLayout**)
    - applicabile solo a sottopannelli
    - può contenere al max. **n** X **m** widget (uno per cella della griglia - ogni widget comprende tutti gli elementi grafici utili a realizzarlo, quali label, widget vero e proprio etc..)
    - per ciascun widget è possibile specificare con precisione la posizione nella griglia e, opzionalmente, il numero di colonne per cui la porzione **non-label** del widget si deve estendere (1 colonna = 1 posto per un widget completo, indipendentemente dal numero di elementi grafici utilizzati nel codice generato.

I widget disponibili condividono alcuni comportamenti comuni:

- possono essere disabilitati o abilitati a comando (abiltazione/disabilitazione/stato di default, comandabile singolarmente o utilizzando un concetto di **ScreenState**): se disabilitati non possono accettare input o comandi dall'utente e, se previsto dallo skin del portale, assumono un'apparenza grafica differente;
- possono essere resi invisibili/visibili a comando (visibile/invisibile/stato di default, comandabile singolarmente o utilizzando un concetto di **ScreenState**): se invisibili non vengono renderizzati nella schermata;
- possono essere visibili/invisibili o abilitati/disabilitati a fronte di **Security constraints** che possono essere o meno soddisfatti per l'utente correntemente collegato all'applicazione;
- possono essere posizionati con precisione nel pannello contenitore utilizzando una parametrizzazione specifica del tipo di layout del contenitore (vedi sopra);
- possono essere corredati da un **EventHandler** in grado di gestire eventi di interazione umana (per l'elenco delle tipologie di eventi supportate da ciascun widget e per dettagli sulla gestione degli eventi si vedano le sezioni apposite);

Widget disponibili

- **CommandWidget**: classe di widget che hanno lo scopo principale di gestire *comandi utente*. I **CommandWidget** condividono la possibilità di catturare eventi di user interaction (vedere sotto). Sono disponibili i seguenti widget:
    - Pulsanti (**ConfirmButton**) che permettono l'esecuzione di una serie di azioni/comandi
    - Pulsanti che permettono la pulizia della schermata senza esecuzione di nessuna logica applicativa (**ResetButton**)
    - Select list: selezione singola o multipla da una lista di valori (**ComboBox**)
    - Radio button: pulsanti a selezione esclusiva (**RadioButtons/RadioButton**)
    - Comandi di navigazione di TabSet e Wizard (**TabSwitcher**, widget implicito)
    - Menu (la struttura è definita centralmente nell'applicativo; questo widget è solo un segnaposto necessario per collocare il menu all'interno della schermata)
- **DataWidget**: widget destinati all'input/output di informazioni.
    - condividono alcuni comportamenti comuni aggiuntivi rispetto ai widget generici:
        - sono dotati di un *tipo* del dato associato
        - prevedono la possibilità di effettuare una serie di validazioni sull'input immesso
        - possono essere collegati al *model* mediante appositi binding (**AppDataBinding**)
        - possono essere valorizzati in base allo stato delle variabili (**ApplicationData**) dell'applicazione
        - possono passare l'informazione immessa dall'utente ai moduli di logica applicativa (esclusi **Image**, **PlainText**)
        - alcuni di essi possono essere anche dei **CommandWidget** (es. **Table**, **ComboBox**, **RadioButton**)
    - sono disponibili i seguenti widget:
        - Campi di immissione testo monolinea (**TextField**) o multilinea (**TextArea**)
        - Drop-down list (**ComboBox**) con possibilità di funzionamento a selezione singola o multipla
        - Pulsante on-off (**Check Box**)
        - Pulsanti a scelta esclusiva in un insieme di opzioni definite a tempo di design (**RadioButton**)

- Tabella di visualizzazione (**Table**) con possibilità di:
  - ordinamento
  - paginazione
  - export dei dati in vari formati (pdf, xls)
  - selezione di una o più righe per l'esecuzione di un'azione su di esse
  - editing "in place" di una cella con diversificazione della modalità di editing a fronte del tipo di dato della colonna:
    - textfield in caso di tipo Stringa
    - checkbox in caso di tipo boolean
  - editing in place con possibilità di selezionare il valore in una lista di opzioni, che possono essere le stesse per tutte le righe oppure diversificate riga per riga
  - possibilitè di rendere o meno editabile a seconda di condizioni valutate a runtime una singola cella

  ordinamento, paginazione, export dei dati in vari formati, selezione di una o più righe per l'esecuzione di un'azione su di esse, editing "in place" di una cella con diversificazione
- Albero espandibile (**TreeView**)
- Immagine (**Image**)
- Testo puro (**PlainText**, widget per sola visualizzazione)
- Tool di selezione data (**Calendar**)

Maggiori dettagli riguardo alle modalità di strutturazione della user interface sono disponibili nella sezione “ui patterns” della user guide di MDDTools-guigen integrata in eclipse.

2.2.2.1.1 Strutturazione modulare della GUI

E'possibile utilizzare per la strutturazione della User Interface un approccio a componenti: è infatti possibile:

- definire un pannello riusabile (con eventuale strutturazione in sottopannelli, binding, event handlers e comandi associati)
- utilizzare tali pannelli nel modello dell'applicazione, contestualizzandoli (ridefinizione degli application data collegati ai widget, ridefinizione di Actor/Roles/UseCases referenziati nei security constraints).

Funzionalità di *Rich User Experience*

Se abilitate, sono disponibili alcune funzionalità di arricchimento della user-experience. Attualmente le funzioni previste sono le seguenti:

- refresh selettivo di una porzione di schermata (per aggiornare i dati o visualizzare un tab o un pannello mutevole senza ricaricare l'intera pagina)
- ordinamento delle righe di una **Table** senza ricaricare l'intera schermata
- comparsa di tooltip sulle label dei widget a fronte del passaggio con il mouse sopra la label stessa
- comparsa di tooltip sull'intestazione delle colonne di una **Table** a fronte del passaggio con il mouse sopra l'header stesso
- possibilità di ridimensionare, nascondere, spostare le colonne di una **Table**
- maschera di immissione dati nei campi di testo coerenti con il tipo di dato associato al widget (es. solo caratteri numerici se il tipo è numerico)
- autocompletamento progressivo della selezione di una **ComboBox** a fronte dell'immissione dei caratteri iniziali della selezione stessa
- suggestion su **TextField** con interazione server-side per effettuare la selezione progressiva delle voci suggerite a fronte dell'immissione testuale
- suggestion su **ComboBox** con interazione server-side per effettuare la selezione progressiva delle voci suggerite a fronte dell'immissione testuale

Securizzazione dell'applicativo

E' possibile modellare la maggior parte delle caratteristiche di sicurezza dell'applicazione:

- Autenticazione: è possibile definire il meccanismo utilizzato per l'autenticazione, scegliendolo tra:
  - Autenticazione in single sign on tramite **SSOBART**
  - Autenticazione in single sign on tramite **Oracle Portal - OPAUTH**
  - Autenticazione in single sign on tramite **Shibboleth**
- Autorizzazione/profilazione: è possibile definire eventuali constraint sulla visibilità o abilitazione di ciascun widget; i possibili tipi di constraint utilizzabili sono:
  - **Actor-based Security Constraint**: il widget viene visualizzato/reso attivo solo se l'utente corrente impersonifica un ben determinato Actor di IRIDE2
  - **Role-based Security Constraint**: il widget viene visualizzato/reso attivo solo se l'utente corrente appartiene ad un ben determinato Ruolo di IRIDE2
  - **UseCase-based Security Constraint**: il widget viene visualizzato/reso attivo solo se l'utente corrente è abilitato ad un determinato UseCase di IRIDE2

- **Custom Security Constraint**: il widget viene visualizzato/reso attivo solo se l'utente corrente soddisfa una logica custom (codificata manualmente)

Lo sviluppatore ha inoltre a disposizione nella business logic le info relative all'utente corrente, nell'ApplicationData denominato "currentUser".

2.2.2.2 Definizione del **model** gestito nell'applicativo

Lo strato di model nell'applicativo è costituito da:

- definizione di **tipi** utilizzati nella logica applicativa. I tipi possono essere:
  - tipi complessi (**ComplexType**)relativi agli oggetti che dovranno essere maneggiati dalla logica applicativa (strutture dati)
  - tipi semplici (**SimpleType**) user defined che si aggiungono ai tipi semplici base customizzandone alcuni aspetti (es. formato, conversione, logica di uguaglianza/confronto, ...)
- definizioni di **ApplicationData**, che rappresentano le porzioni di informazione (strettamente tipizzate) che vengono maneggiate all'interno della logica applicativa e dallo strato di view:
  - hanno un nome che li identifica a livello di applicazione e con il quale possono essere referenziati all'interno delle GUI o nella logica applicativa
  - sono tipizzati (tipi semplici o complessi o collezioni di tipi semplici o complessi)
  - possono "vivere":
    - per il solo tempo di una singola **interazione utente** (scope = USER_ACTION)
    - per tutto il tempo in cui l'utente permane in uno **stesso ContentPanel**(scope = PAGE) anche a fronte di più interazioni successive
    - per tutta la durata della **sessione utente** (scope = USER_SESSION)
  - sono coinvolti in *sessioni di editing*, con possibilità di effettuare *undo* delle modifiche apportate da U.I. tramite appositi comandi.

La logica di business è nettamente separata dalla logica di U.I e deve essere codificata manualmente:

- avendo a disposizione gli **ApplicationData** necessari, il cui valore è pre-compilato (se necessario) a fronte dell'input utente sulle GUI a cui tali dati sono collegati o a fronte del valore contenuto in sessione se l'**ApplicationData** è a scope USER_SESSION o PAGE;
- restituendo al gestore del contesto applicativo gli **ApplicationData** creati/modificati dalla logica, in modo che possano essere maneggiati dallo strato di presentation
- implementando metodi con una interfaccia ben definita
- il collegamento tra U.I, controller e model/logica è modellato come spiegato nel paragrafo successivo

2.2.2.3 Definizione del comportamento dinamico dell'applicativo

Il comportamento dinamico dell'applicativo si concretizza nella possibilità di a**ssociare una serie di comandi agli eventi di user interaction** che avvengono sull'applicativo. In particolare sono disponibili le seguenti possibilità/comandi:

- comandi di gestione dello stato della schermata
  - **ONOFFCommand**: abilitazione/disabilitazione di un set di widget definito a tempo di design
  - **VisibilityCommand**: rendere visibile/invisibile un set di widget definito a tempo di design
  - **ScreenStateCommand**: permette di far passare la schermata corrente ad un particolare stato di abilitazione/visibilità complessiva dei widget in esso contenuti;
- comandi di gestione del flusso di navigazione
  - **JumpCommand**: saltare ad una schermata (**ContentPanel**) differente da quella corrente
  - **JumpExtCommand**: saltare ad una pagina, tramite link diretto (in genere utilizzato per uscire dal contesto applicativo corrente e puntare, ad esempio, ad una pagina statica o ad un'applicazione esterna; talvolta utilizzato per eseguire una *action custom* all'interno dello stesso applicativo.
  - **ShowDialogCommand**: passare alla visualizzazione di una pagina corrispondente ad un **DialogPanel**;
- comandi di richiamo della business logic
  - **ExecCommand**: eseguire una logica applicativa java (il cui codice deve essere scritto manualmente) e, a seconda dell'esito eseguire altre sequenze di comandi
- comandi di gestione dello stato applicativo
  - **BeginEditSession**: inizia una sessione di editing su un insieme di **ApplicationData** mantenendo automaticamente un backup dei dati per un eventuale necessità di *undo*
  - **EndEditSession**: termina una sessione di editing su un insieme di **ApplicationData**, permettendo (se specificato) di effettuare l'*undo* delle modifiche effettuate durante la sessione
  - **ChkEditStatusCommand**: permette di verificare lo stato di una sessione di editing con la possibilità di effettuare logiche differenti a seconda che gli **ApplicationData** sotto osservazione siano stati o meno modificati.
- altri comandi
  - **SequenceCommand**: permette di eseguire una sequenza di comandi

I comandi che devono essere eseguiti a fronte di un evento di U.I su un particolare widget sono implementati automaticamente dal codice generato, ad esclusione dell'**ExecCommand**. In quest'ultimo caso, infatti, sebbene il codice generato provveda a richiamare automaticamente un metodo di business fornendogli il contesto applicativo e modificando il contesto applicativo con i risultati dell'esecuzione di tale metodo, il metodo di business deve essere codificato manualmente nel linguaggio target (java). Al momento questo rappresenta l'unico punto di intervento manuale post-generazione sul codice.

I possibili punti di aggancio di un comando sono:

- evento click su ConfirmButton: la logica viene eseguita alla pressione del pulsante
- selezione di una cella di una Table
- selezione di una voce di menu
- evento di cambiamento della selezione di una ComboBox: la logica viene eseguita ogni volta che viene selezionato un valore nella combo
- evento selezione di un radiobutton: la logica viene eseguita al click su uno dei radiobutton di un gruppo di radiobutton
- refresh della pagina (contentPanel): questa logica viene eseguita ogni volta che viene ricaricata una pagina, comprese le volte in cui il refresh è conseguente all'esecuzione di un altro evento.
- inizializzazione dell'applicazione: la logica viene eseguita ogni volta che si passa dalla pagina iniziale (con la possibilità di recepire eventuali parametri di inizializzazione nell'url di richiamo - **ActivationModel**)

2.3 SERVICEGEN: Sviluppo Model Driven di servizi in architettura SOA

SERVICEGEN è lo strumento model driven per la generazione di servizi secondo il paradigma SOA. Attualmente la tecnologia target è rappresentata dallo stack J2EE arricchito con le librerie di cooperazione applicativa C.S.I. Lo strumento permette di definire uno o più servizi realizzati all'interno di un componente di prodotto, secondo gli standard di sviluppo e le linee guida vigenti in *CSIPIEMONTE*.

Il modello che descrive l'applicazione è:

- basato su un metamodello (descritto in dettaglio nell'help online disponibile all'interno di eclipse dopo l'installazione del plugin);
- strutturato in un insieme di modelli interconnessi che descrivono i vari aspetti del servizio (la struttura delle dipendenze tra i vari modelli è descritta in dettaglio nell'help online disponibile in eclipse dopo l'installazione del plugin);

Il progetto generato produce pacchetti funzionanti su una specifica target platform J2EE, a scelta tra:

- Weblogic Server 9.2
- Jboss 4.3 EAP

Per ciascun componente è possibile definire uno o più definizioni di servizio, ciascuna delle quali è caratterizzata da:

- Un codice servizio, così come verrà classificato nel repository dei servizi
- Una versione dell'interfaccia (in formato *major.minor*)
- La tipologia di servizio, da scegliere tra applicativo, di orchestrazione, infrastrutturale
- Un insieme di tipi user defined (entità, array tipati contenenti entità user defined, ed eccezioni)
- Un insieme di operazioni esposte dall'interfaccia del servizio, che possono utilizzare come parametri o valori di ritorno un tipo base csi o un tipo user defined (entity)
- Eventuali regole di autorizzazione all'utilizzo delle varie operazioni.

Il componente, oltre a definire un certo numero di servizi, li espone (*provides*) affinchè possano essere fruiti da sistemi utilizzatori. Questa esposizione è effettuata tramite uno o più **binding**, che possono essere attualmente dei seguenti tre tipi:

1. **Binding PA EJB**: è il binding classico, sempre presente, rappresentato da una Porta Applicativa CSI basata su tecnologia EJB e interazione RMI con il fruitore;
2. **Binding Bridge di PA SOAP**: è un binding opzionale che permette la fruizione del servizio da parte di un sistema (non necessariamente java) dotato di porta delegata soap;
3. **Binding front-adapter web-services**: è un binding opzionale che permette la fruizione del servizio a tutti quei sistemi che non possono utilizzare una porta delegata ma che sono in grado di instaurare una comunicazione web-services rpc soap/http.

A fronte di una applicazione logicamente definita secondo il modello appena esposto, il generatore di applicazione CSI genera:

- Per ogni componente:
    - Un progetto SVN standard dotato di build ant e contenente tutti gli artifact che realizzano tutti i servizi definiti ed esposti in quel componente. Il risultato finale del build sarà un EAR contenente:
        - Un *ejb-jar* per ogni binding *paejb*
        - Un *war* per ogni bridge *soap*
        - Un *war* per ogni front adapter *web-services*

    - Il frammento di configurazione log4j da inserire nella configurazione del server, contenente:
        - Le configurazioni per il log applicativo
        - Le configurazioni per lo stopwatch
        - Le configurazioni per coop-trace, se il componente contiene almeno un servizio coop-trace enabled (default)
- Per ogni definizione di servizio all'interno di ciascun componente:
    - L'interfaccia CSI del servizio
    - Le classi java corrispondenti alle entità e alle eccezioni user-defined
    - Le classi che implementano il servizio come Stateless Session EJB
    - Nel caso dei servizi orchestrati gli scheletri dei descrittori XML utilizzati dalla libreria *svcflow* dei flussi relativi a ciascuna operazione
    - Il build che costruisce i pacchetti interni relativi ai singoli binding e la distribuzione delle librerie client dell'interfaccia del servizio; nel caso di binding ejb, se il servizio è coop-trace-enabled, i file conterranno anche le configurazioni dei pluggable function handler appositi .
    - I file di porta delegata (a scopo di test o da distribuire ai fruitori - ejb sempre, soap solo se presente il relativo binding) – nel caso in cui il servizio sia coop-trace enabled, i file conterranno anche le configurazioni dei pluggable function handler appositi.
    - Un abbozzo di test JUnit per ciascun servizio
    - Un build specifico utile per generare gli stub axis relativi ai front-adapter web-services (se presenti) - a scopo di test

Lo scenario di utilizzo dello strumento prevede quindi la generazione *model driven* con round-trip dei servizi: definendo incrementalmente il modello dell'applicazione (secondo il metamodello specifico di SERVICEGEN) è possibile generare una applicazione completa CSI che può evolvere nel tempo con round trip monodirezionale modello → codice.

Compito dello sviluppatore è quello di arricchire il codice generato inserendo manualmente nelle apposite *regioni protette* il codice che implementa il servizio.

Nel caso si utilizzi SERVICEGEN per realizzare servizi di orchestrazione è possibile modellare l'orchestrazione, definendo:

- L'insieme dei servizi orchestrati (ovvero acceduti dall'orchestrazione), definiti come elementi di un *ResourceSet*
- La sequenza degli step (*nodes*) che costituiscono l'orchestrazione, che possono essere:
    - Nodo di *start*: nodo iniziale che marca l'inizio dell'orchestrazione e inserisce in uno o più *DataSlot* i valori di ciascun parametro di input del metodo di orchestrazione

- Nodo di *stop*: nodo finale che marca il termine dell'orchestrazione e permette di restituire come output del metodo di orchestrazione il valore contenuto in un particolare *DataSlot*
- Nodo di *trasformazione*: nodo che permette di effettuare trasformazioni di valori contenuti nel contesto di esecuzione (a partire da uno o più *DataSlot* e memorizzando il risultato in un *DataSlot*)
- Nodo di richiamo di un servizio PDPA:
    - per accedere ad un particolare metodo di uno dei servizi dichiarati nel ResourceSet
    - valorizzando i parametri con il valore di uno o più *DataSlot*
    - memorizzando l'eventuale valore di ritorno in un *DataSlot*
    - eseguendo una sottosequenza di step in caso di eccezione (*ExceptionHandler*)
- Nodo di Loop (*ForEach*) per ciclare all'interno di una collection contenuta in un *DataSlot*, ed eseguire una sottosequenza di step
- Nodo di branch condizionale, per eseguire una tra due possibili sottosequenze di step a fronte del risultato di una condizione logica

Nel caso dei servizi di orchestrazione il generatore si preoccupa anche di gestire automaticamente:

- la creazione dei file di configurazione delle PD dei servizi acceduti
- la risoluzione delle dipendenze rispetto alle librerie client dei servizi acceduti

2.4 DATAGEN: Sviluppo Model Driven di logiche di accesso ai dati

DATAGEN è lo strumento di generazione di codice specifico per la generazione dello strato software che realizza l'accesso ai dati mantenuti in un DBMS relazionale.

Il modello che descrive lo strato di accesso ai dati è basato su una coppia di metamodelli (uno per la struttura della base dati, ed uno per la modellazione delle logiche di accesso a tale DB – i metamodelli sono descritti in dettaglio) nella reference guide disponibile in eclipse dopo l'installazione del plugin);

Il generatore *datagen* (a differenza di quanto succede con *guigen* o *servicegen*) non genera un progetto autoconsistente ma crea gli artefatti necessari a definire lo strato di accesso ai dati all'interno di un altro progetto java preesistente, che può essere un progetto di front-end (guigen) o un progetto di servizio SOA (servicegen).

Allo stato attuale datagen genera DAO compatibili con i seguenti DBMS:

- Oracle 10
- PostgresSQL: 9.0

2.4.1 Funzionalità

DATAGEN ha l'obiettivo di permettere la realizzazione dello strato di accesso ai dati in due modalità differenti:

1. modalità *top-down*, che consiste nella modellazione dello schema relazionale e nella generazione di:
    - script DDL di creazione dello schema
    - classi java che implementano lo strato *DAO*
2. modalità *bottom-up*, che consiste nell'effettuare il *reverse engeneering* dello schema DB nella generazione di:
    - classi java che implementano lo strato *DAO*

La modalità *top-down* è attualmente non ancora utilizzabile. Per la modalità *bottom-up* è invece disponibile un *wizard* che permette di effettuare il *reverse engineering* a partire da uno schema relazionale esistente. In entrambe le modalità svolge un ruolo centrale il modello dello schema relazionale, con la sola differenza che nella modalità *top-down* è progettato direttamente, mentre nella modalità *bottom-up* è l'effetto di una lettura dei metadati dello schema oracle.

A questo modello dello schema relazionale si affianca un modello dello strato vero e proprio di accesso ai dati che, per ciascuna tabella permette di definire un **DataAccessObject** che può mettere a disposizione dell'utilizzatore:

- le funzioni per inserire un nuovo record (**Inserter**)(con la possibilità di utilizzare, se necessario, una sequence o altre tecniche per la creazione della *primary key* (**PKGenerator**)
- le funzioni per effettuare delle ricerche (**Finder**):
    - lettura di tutte le righe della tabella (**FindAll**)
    - lettura di un singolo record selezionato per *primary key* (**FindByPK**)
    - lettura di un sottoinsieme di record selezionati con criteri *custom* (**CustomFinder**, clausola WHERE da scrivere manualmente)
    - lettura di un sottoinsieme di record selezionati tramite la tecnica del *Query By Example* (**QBEFinder**)
- Le funzioni per effettuare modifiche ai dati di uno o più record, selezionati per *primary key* o altri criteri custom (**Updater**)
- Le funzioni per cancellare uno o più record, selezionati per *primary key* o altri criteri custom (**Deleter**)

L'implementazione dei *DataAccessObject* è realizzata utilizzando il framework *struts* e *jdbc-template*: per utilizzare i *DAO* nella business logic sarà dunque sufficiente utilizzare i consueti meccanismi di *localizzazione* dei bean relativi e utilizzarne i metodi.

Circa i dettagli sulla struttura e il design del codice generato si rimanda alla sezione “design rationale” disponibile nell'help on line di eclipse a fronte dell'installazione del plugin.

3 Getting started

In questo capitolo verrà fornito un percorso minimale utile per iniziare ad utilizzare praticamente gli strumenti. Per un utilizzo effettivo è invece necessario consultare la documentazione online dei vari strumenti disponibile in eclipse o le risorse ausiliarie di autoapprendimento (es. reference application).

3.1 Prerequisiti ed installazione

La suite di Model Driven Development *MDDTools* è basata sull'I.D.E *Eclipse*, le cui funzionalità sono estese tramite una serie di plugin specifici sviluppati in CSI che permettono di modellare le differenti parti di applicazione e generare il codice corrispondente.

I prerequisiti per una postazione di lavoro sono i seguenti:

- Sistemi operativi supportati: Microsoft Windows 2000, Windows XP, Windows Vista, Windows 7.
- JDK 1.5 (o superiori) a 32bit (il bundle base di eclipse è il bundle a 32 bit).
- almeno 2Gb di memoria RAM (consigliati 4Gb o più, specie se si desidera lanciare in locale l'application server)
- almeno 1Gb di spazio su disco (per la sola installazione)

Le attività necessarie per la predisposizione della postazione sono le seguenti:

1. installazione del *bundle* base di Eclipse predisposto dal CSI-Piemonte.
2. installazione delle estensioni di eclipse messe a disposizione dalla suite *MDDTools*

E' altresì prevista una distribuzione che contiene già una pre-installazione dei plugin nella versione più recente.

3.1.1 Installazione del bundle base di *eclipse*

Il bundle base di *eclipse* fornito dal CSI-Piemonte è un package preconfezionato dal CSI-Piemonte che contiene:

- il runtime dell'IDE *Eclipse* (attualmente nella versione 3.6 - “helios”)
- le estensioni necessarie per lo sviluppo secondo le tecnologie e il framework di sviluppo al momento in utilizzo in CSI:
  - strumenti di base per lo sviluppo in linguaggio Java
  - strumenti aggiuntivi per lo sviluppo di applicazioni J2EE (Web Tools Project)
  - integrazione embedded con server J2EE Jboss (per WLS è disponibile la modalità tramite debug remoto)
  - integrazione con SVN (subclipse)
- le tecnologie base per il model-driven-development, sulle quali sono basati i tool MDD
  - emf
  - openarchitectureware

Per l'installazione del bundle è sufficiente scompattare in una directory a piacere l'archivio scaricato.

N.B: in caso si verificassero problemi durante lo scompattamento utilizzare il tool "jar" per effettuare l'operazione.

3.1.2 Installazione dei plugin MDDTools

N.B: Questo passo non è necessario se si utilizza un *bundle preconfigurato*, ma solo se si utilizza il *bundle base*.

Una volta installato il bundle base è necessario installare i plugin che costituiscono la suite di tool MDD:

- *datagen* (modellazione e generazione dello strato di accesso ai dati),
- *guigen* (modellazione e generazione dello strato di user application),
- *servicegen (modellazione e generazione servizi in architettura SOA),*
- *mddtools* (documentazione e strumenti ausiliari).

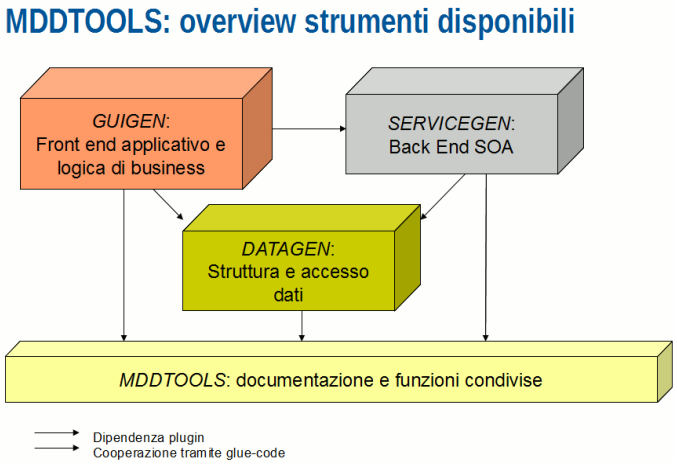

Illustrazione 1: mddtools: plugin e dipendenze

I plugin sono disponibili per il download nel formato dell' update-site eclipse.

N.B: E' necessario installare il plugin *mddtools* prima di installare gli altri tre.

N.B: E' disponibile un bundle contenente già i plugin preinstallati. Questo bundle è destinato all'utilizzo al di fuori dalla software factory CSI-Piemonte.

3.1.3 Usage tracking e configurazione iniziale dei plugin

La suite MDDTools contiene un sistema di raccolta di dati statistici sull'utilizzo dei tool, finalizzata al miglioramento dello strumento. Per tale motivo è consigliabile, al termine dell'installazione dei plugin, indicare i propri riferimenti nella finestra di configurazione del plugin MDDTools, accessibile tramite il menu window → preferences, e selezionando in seguito la voce "mddtools" nell'albero di navigazione.
All'interno della form è necessario inserire i propri riferimenti, che saranno utilizzati per la comunicazione diretta di novità inerenti gli strumenti mddtools.
E' possibile utilizzare il tracking in modalità anonima: è sufficiente fornire come nome il nome fittizio "anonymous" e come cognome un identificativo fittizio (es. "user_1"). Affinchè i dati inviati al collettore siano significativi dal punto di vista statistico è preferibile associare ad utenti differenti identificativi fittizi differenti: questo, pur garantendo l'anonimato, evita di falsare i risultati statistici unificando eventi in realtà riferiti a utilizzatori distinti.

E' inoltre possibile disabilitare totalmente il tracking selezionando un apposito check nella pagina di preferenze di MDDTools. Poiché i dati di tracking sono utilizzati con lo scopo di migliorare lo strumento e il supporto all'utilizzo di consiglia di mantenere il tracking attivo e di preferire piuttosto l'impostazione anonima.
Nel caso si decida di mantenere attivo il tracciamento è necessario impostare, sempre nella pagina di preferenze di MDDTools, le seguenti informazioni:

1. url del servizio di ricezione degli eventi di utilizzo: http://servizi.csi.it/mddtrksv
2. impostazioni di proxy nel caso in cui l'utilizzo degli strumenti avvenga all'interno di una rete non direttamente collegata ad internet

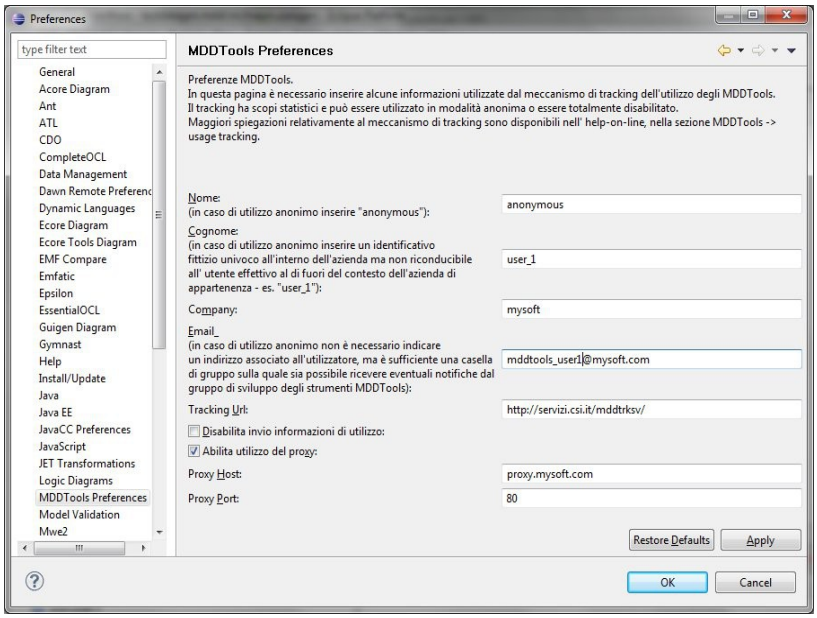

Illustrazione 2: configurazione iniziale installazione MDDTools

Attenzione: se non viene completata questa operazione e il tracking non viene disabilitato, gli strumenti di modellazione e di generazione non saranno funzionanti!

I dati che verranno inviati al servizio collettore sono i seguenti:

- dati identificativi/contatti dell'utilizzatore:
  - *nome e cognome* dell'utilizzatore (oppure *anonymous/<user_fittizio>* in caso di utilizzo anonimo)
  - *nome dell'azienda* di appartenenza dell'utilizzatore
  - *indirizzo e-mail* dell'utilizzatore (l'indirizzo è utilizzato come elemento di contatto per eventuali notifiche del gruppo di sviluppo MDDTools verso i suoi utilizzatori; se si sceglie di impostare il tracking in modalità anonima si consiglia comunque di fornire un indirizzo reale, in modo da poter ricevere tali notizie).
- contesto di esecuzione del tool:
  - *ip address* dove si sta eseguendo lo strumento (rilevato sul client => locale alla sottorete dell'utilizzatore)
  - nome del *sistema operativo* su cui è in azione il tool
  - *versione di Java* che sta eseguendo il tool
- informazioni di utilizzo:

- *nome e versione del modulo/plugin* che si sta utilizzando (guigen/datagen/servicegen)
- *tipo di operazione* (attivazione del plugin/generazione di un progetto)
- dati aggiuntivi (solo per operazione di generazione): *nome del modello* in input

La fonte di queste informazioni è la seguente:

- i dati identificativi e di contatto sono non in autocertificazione e non verificati (sono quelli inseriti manualmente come descritto in precedenza)
- i dati del contesto di esecuzione del tool sono letti automaticamente da sistema
- i dati relativi all'informazione di utilizzo dipendono appunto dall'attività degli utilizzatori

I dati sono conservati a tempo indeterminato in un Database nella server farm CSI-Piemonte, sono accessibili esclusivamente dal personale addetto all'evoluzione del prodotto MDDTools e sono utilizzati:

- per scopi statistici relativi alla distribuzione di utilizzo del prodotto
- per l'invio di comunicazioni mirate a utilizzatori di particolari versioni del prodotto al fine di segnalare tempestivamente evoluzioni, problemi, etc. etc.

N.B: A tutela della legge sulla protezione dei dati personali, se un utilizzatore degli strumenti decide di utilizzare la modalità anonima, CSI-Piemonte non ha modo di correlare gli identificativi fittizi con le persone reali ad essi associate, poiché tale associazione è conosciuta esclusivamente all'interno dell'organizzazione a cui appartiene l'utilizzatore.

3.2 Concetti comuni a tutti gli strumenti

Tutti gli strumenti contenuti nella suite mddtools sono basati su alcuni concetti condivisi che è necessario padroneggiare e che afferiscono all'utilizzo delle tecnologie di base EMF e Openarchitectureware. Di seguito si fornisce una breve panoramica delle attività tipiche, rimandando per i dettagli alla documentazione online.

3.2.1 Struttura dei workspace di progetto

L'utilizzo degli strumenti model driven mddtools prevede il popolamento di un workspace eclipse costituito da:

- un progetto generatore
  - progetto di tipo “openarchitectureware project”
  - contiene modelli e direttive di generazione
- uno o più progetti “target”, generato/i a partire dai modelli e secondo le direttive di generazione contenute nel progetto generatore[1].

Le tipiche attività di sviluppo sono sintetizzabili come segue:

- progetto generatore: modellazione
- progetto generatore: generazione
- progetto generato: completamento codice custom
- progetto generato: build e deploy

3.2.2 modellazione di modelli basati su tecnologia EMF

I modelli degli strumenti *mddtools* sono tipici modelli EMF. Di conseguenza per poter procedere alla modellazione occorre svolgere le seguenti tipologie di attività:

1 La tipologia dei progetti generati dipende dai prodotti dei vari generatori: ad esempio nel caso di guigen è possibile generare a partire da un modello guigen due progetti target: un progetto J2EE ed un progetto di risorse web.

- creazione di un nuovo modello: si realizza attivando appositi wizard (menu: *file* → *new*). I wizard sono classificati sotto le categorie:
  - guigen wizards
  - servicegen wizards
  - datagen wizards
- editing di un modello utilizzando l'editor strutturato (tree editor) relativo alla specifica tipologia di modello:
  - creazione guidata di un nuovo nodo figlio a seconda della struttura prevista dal metamodello (menu "new child → <tipo elemento>" contestuale ai vari nodi dell'albero)
  - modifica dei valori delle proprietà previste da un determinato nodo tramite la *view* "properties" di eclipse
    - impostazione di valori numerici o stringa nelle properties corrispondenti ad attributi semplici del metamodello
    - selezione di un elemento da elenco nel caso di properties corrispondenti ad attributi enumerati nel metamodello
    - selezione singola di un elemento da una lista in caso di associazioni n → 1 nel metamodello
    - selezione multipla di un elemento da una lista in caso di associazioni n → m nel metamodello
  - inclusione di modelli esterni (tramite "load resource") per una successiva referenziazione.

3.2.3 Generazione del codice

Una volta definiti i modelli è necessario configurare e, successivamente, lanciare uno o più *workflow* di generazione.
Il *workflow* di generazione è uno strumento messo a disposizione dalla tecnologia *oaw (openarchitectureware)* per eseguire la generazione del codice a fronte di un insieme di modelli.
La documentazione relativa alla configurazione dei *workflow oaw* è disponibile nell'help on line di eclipse:

- informazioni generali sui workflow oaw: nella documentazione del tool openarchitectureware
- informazioni sulle "cartucce" (*cartridges*) messe a disposizione dai tool MDD: nella sezione "cartridges" disponibile nell'help on line di eclipse a fronte dell'installazione del plugin *mddtools*.

3.2.4 Completamento manuale del codice generato

Il codice generato dagli strumenti *mddtools* necessita di completamento manuale (ad esempio per la realizzazione della logica di business). Tale completamento deve avvenire inserendo manualmente il codice nella porzione di sorgente demarcato da appositi commenti del tipo:

```
...
/*PROTECTED REGION ID(R-2003978741) ENABLED START*/
/// codice custom...
/*PROTECTED REGION END*/
...
```

Il codice inserito in queste regioni ha la proprietà di essere mantenuto a fronte di successive rigenerazioni del modello.
E' assolutamente da evitare l'inserimento di codice custom al di fuori da queste aree, pena la perdita del codice immesso alla rigenerazione successiva.

3.3 Utilizzo di MDDTools in contesti differenti dal CSI-Piemonte

MDDTools è una suite di strumenti sviluppati in CSI-Piemonte. Per tale motivo implementano standard di vario genere in uso nella Factory CSI-Piemonte. Alcuni aspetti relativi a questa caratteristica sono:

- l'utilizzo di un framework di sviluppo costituito da un insieme specifico di librerie
- la compatibilità con la piattaforma di middleware utilizzata in CSI-Piemonte (sia a livello di execution enviroment – application server, dbms – sia di servizi infrastrututrali – es. autenticazione/security)
- l'utilizzo di stili grafici specifici degli enti di riferimento del CSI-Piemonte
- l'adozione di standard di codifica/naming/alberatura in uso in CSI-Piemonte
- l'adozione di procedure di build specifiche.

Per l'utilizzo degli MDDTools all'esterno del contesto CSI-Piemonte è possibile/necessario adottare alcuni accorgimenti o tenere conto di alcune questioni, che sono spiegate nei paragrafi successivi.

3.3.1 Considerazioni riguardanti l'utilizzo del framework di sviluppo CSI-Piemonte

Il codice generato dagli MDDTools utilizza un insieme di librerie che sono indispensabili per il funzionamento delle applicazioni generate.
Le librerie utilizzate sono da considerarsi dei prerequisiti per l'utilizzo degli MDDTools. Per l'utilizzo interno alla software factory CSI-Piemonte tali librerie sono disponibili sul repository IVY o (nel caso ad esempio delle librerie JS) preinstallate direttamente nei vari ambienti; per l'utilizzo in contesti differenti da quello del CSI-Piemonte, invece, viene fornito un clone del repository IVY il cui utilizzo è descritto al paragrafo 3.3.6. In questo paragrafo verranno esposte invece alcune considerazioni riguardanti le differenti librerie, soprattutto per quanto riguarda le problematiche di licensing.
Le librerie utilizzate dai vari tool sono classificabili, da tale punto di vista, in tre categorie:

1. librerie open-source di terze parti (es. apache, …) disponibili liberamente in internet senz anessun corrispettivo economico
2. librerie open-source realizzate in CSI-Piemonte e utilizzabili senza nessun corrispettivo economico
3. liberie open-source di terze parti il cui utilizzo può essere soggetto a pagamento di un corrispettivo economico a fronte di particolari scenari di utilizzo.

Nella tabella seguente si evidenziano le sole librerie del terzo tipo, poiché il loro utilizzo richiede l'adeguamento dell'utilizzatore a tali politiche di distribuzione.

Libreria	***versione***	***Utilizzo nei tool***	***Produttore***	***Licenza***
ExtJS	03.02.02	**Guigen**: utilizzata per la realizzazione delle funzionalità di Rich User Interface. L'abilitazione di tali funzionalità è opzionale. In mancanza di tale impostazione alcune delle funzionalità di U.I saranno non disponibili o limitate. Per i dettagli consultare la documentazione del tool guigen, sezione UI patterns, in cui vi è una matrice di presenza delle varie features nella versione arricchita e in quella standard.	http://www.sencha.com	Gli estremi della livenza di utilizzo sono disponibili all'url: http://www.sencha.com/products/extjs/license/
Weblogic.jar	9.2	Guigen, servicegen: è il runtime completo dell'application server weblogic 9.2; in caso di generazione per la target platform Weblogic 92 sono utilizzate per la compilazione dei sorgenti generati. A runtime sono ovviamente presenti nell'execution environment.	Oracle	Questo jar è disponibile gratuitamente per lo sviluppo, scaricando dal sito di oracle la platform o l'application server. Il runtime è soggetto a licenza a pagamento.

3.3.2 Considerazioni riguardanti gli execution environment

Gli execution environment supportati dalla versione attuale di MDDTools sono:

- application server:
  - Weblogic 9.2
  - Jboss 4.3
- DBMS:
  - Oracle 10
  - PostgreSQL 9.0

Queste versioni di middleware sono da considerarsi come **prerequisiti** per l'utilizzo degli MDDTools.

3.3.3 Considerazioni riguardanti i servizi infrastrutturali

Alcune delle funzioni messe a disposizione dagli MDDTools dipendono strettamente da servizi infrastrutturali specifici del contesto CSI. In questa versione l'utilizzo di tali funzionalità per lo sviluppo di applicativi non destinati al contesto della server farm CSI-Piemonte non è possibile.
Nel dettaglio le funzionalità non disponibili all'esterno del contesto CSI-Piemonte sono:

- **guigen**:
  - autenticazione ed autorizzazione: in CSI-Piemonte l'autenticazione e l'autorizzazione sono risulte dai servizi infrastrutturali Shibboleth e IRIDE2, che non sono disponibili al di fuori del contesto della server farm CSI. In dettaglio non sarà possibile:
    - definire un **AuthenticationMethod**
    - definire dei **SecurityConstraint** (neppure custom) sui widget o sulle voci di menu
  - funzioni GIS: le funzionalità GIS sono implementate su servizi cartografici GREASE, che non sono disponibili al di fuori del contesto CSI. In dettaglio non sarà quindi possibile utilizzare il widget **MapView**.
- **servicegen**: non è possibile utilizzare nelle orchestrazioni di servizi il richiamo di servizi supplier PAPD con localizzazione basata su csi-registry: il registry è infatti disponibile solo all'interno del contesto CSI.
- **datagen**: non dipende da nessun servizio infrastrutturale.

3.3.4 Considerazioni rigurdanti la stilizzazione grafica

Per la stilizzazione grafica delle applicazioni generate con guigen è necessario utilizzare uno skin, costituito da CSS e risorse grafiche che si agganciano al codice html generato a runtime.
Nella distribuzione pubbloica degli MDDTools è disponibile, per l'utilizzo in contesti differenti dal CSI-Piemonte, uno skin dimostrativo (denominato “neutral”) che può essere utilizzato come esempio/punto di partenza per la realizzazione di skin specifici del contesto di utilizzo degli MDDTools.
Il tempo tipicamente necessario per la realizzazione di un nuovo skin è stimabile nell'ordine delle 2/3 settimane; l'attivtà richiede principalmente competenze di grafica web (CSS) ma anche la conoscenza degli elementi strutturali utilizzabili per definire le interfacce in guigen.
Per dettagli circa la collocazione delle risorse (interne al pacchetto o sul web-server di esposizione), e in generale sulla stilizzazione grafica è possibile consultare l'help on line, nella sezione MDDTools-guigen user guide → guides → guida alla stilizzazione grafica.
La distribuzione dello skin di esempio contiene anche le librerie javascript che sono utilizzate per la realizzazione delle funzionalità di rich user interface.

3.3.5 Considerazioni riguardanti gli standard di codifica

Il codice generato dagli MDDTools rispecchia le scelte standard della software factory CSI-Piemonte. Tali scelte sono in gran parte ispirate a standard universalmente riconosciuti, con alcune piccole eventuali varianti. Si ritiene che tali scelte siano utilizzabili senza problemi anche in contesti extra-CSI. L'unica cosa che è necessario rendere personalizzabile è rappresentata dal prefisso di packaging java: per il CSI-Piemonte tale prefisso è standard ed è “`it.csi`”; gli strumenti permettono di personalizzare tale prefisso per l'utilizzo

in contesti differenti dal CSI-Piemonte. Le istruzioni su come personalizzare tale prefisso sono descritte nelle sezioni “cartridges” dell'help on line dei tre strumenti integrato in eclipse.

3.3.6 Considerazioni rigurdanti il build dei progetti

I progetti generati dagli MDDTools seguono gli standard di build in uso nella software-factory CSI-Piemonte.
In sintesi essi prevedono:

- l'utilizzo di IVYcome meccanismo per il reperimento delle librerie d aincludere nel progetto
- l'utilizzo di ANT come tool di make
- una struttura di make contenente i target necessari alle attività quotidiane di sviluppo ma anche per il corretto funzionamento della build-machine utilizzata per il rilascio dei pacchetti negli ambienti di collaudo e produzione.

Si ritiene che tali scelte siano utilizzabili senza problemi anche in contesti extra-CSI in quanto non eccessivamente vincolanti ad esempio a tecnologie proprietarie.
Merita attenzione solo la necessità di avere a disposizione un repository contenente gli artefatti (librerie) utilizzate a build time da IVY. Tale repository è rappresentato da una struttura di cartelle contenenti le varie librerie che viene esposto tramite un semplice web-server HTTP, che la componente client di IVY contatta per reperire le librerie.
In CSI-Piemonte tale repository è centralizzato ed è ospitato dal server repart.csi.it. In contesti differenti sarà sufficiente predisporre su un qualsiasi web-server l'alberatura delle librerie e indicare nelle direttive di generazione l'hostname del repository. Le istruzioni su come personalizzare tale prefisso sono descritte nelle sezioni “cartridges” dell'help on line dei tre strumenti integrati in eclipse.
Per quanto riguarda invece il contenuto del repository viene fornito un clone del repository CSI contenente tutte le librerie direttamente referenziate dal codice generato dagli strumenti MDD.

3.4 Risorse

Il “kit” MDDTools comprende i seguenti elementi:

- bundle MDDTools con i plugin preinstallati
  - funzioni condivise
  - guigen
  - servicegen
  - datagen
  - documentazione integrata in eclipse
- snapshot dei sorgenti dei plugin mddtools
- documento di “getting-started” (questo documento)
- clone del repository degli artefatti da utilizzare per il build delle applicazioni generate dagli MDD-Tools
- reference application
  - sorgenti del progetto generatore
  - sorgenti del progetto generato
  - documento descrittivo
  - script SQL per la creazione del DB di test
  - risorse grafiche per la stilizzazione dell'applicazione
- skin “neutral” per la stilizzazione grafica delle applicazioni generate da guigen. Serve come punto di partenza da cui creare lo skin del sito di esposizione

Questi elementi sono sufficienti sia per un utilizzo didattico sia per l'utilizzo effettivo.