Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Metodologie, schemi e tools per la descrizione di elementi software e per l'integrazione di codice legacy in componenti grid-enabled

Fabio Collura, Alberto Machì

RT-ICAR-PA-05-14 **Dicembre 2005**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Metodologie, schemi e tools per la descrizione di elementi software e per l'integrazione di codice legacy in componenti grid-enabled

Fabio Collura, Alberto Machì

Deliverable III° anno
Progetto MIUR FIRB Grid.it
Work Package 8
High Performance Component-based Programming Environment

Rapporto Tecnico N.:
RT-ICAR-PA-05-14 **Dicembre 2005**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Indice

Introduzione

Questo documento riassume lo stato di avanzamento del lavoro svolto durante il terzo anno di attività del progetto FIRB ***Grid.it*** **,**dalla Unità di Ricerca ICAR-CNR di Palermo nell'ambito del Workpackage WP8 "Ambienti di Programmazione".

Scopo generale della ricerca, operante in sinergia nel WP9 "Librerie scientifiche", è lo sviluppo di metodologie e strumenti software per l'integrazione di codice legacy nell'ambiente di programmazione ***Grid.it*** [1]. Durante il terzo anno ci si è concentrati sulle seguenti linee di intervento:

- La definizione di euristiche e modelli per la determinazione e controllo della Qualità del Servizio (QoS) da associare alla esecuzione di componenti di librerie scientifiche eseguibili su griglia computazionale (grid-enabled) (WP8 & WP9)
- Lo sviluppo di una metodologia e di strumenti per la integrazione nell'ambiente di programmazione ASSIST di codice generico non nativamente sviluppato in componenti (integrazione di moduli di libreria). (WP8)

Obiettivi di ricerca specifici sono:

1. Lo sviluppo di modelli ed euristiche per la definizione di funzioni e di profili di Qualità di Servizio per moduli di libreria grid-enabled. Lo sviluppo di una metodologia e di un middleware per il monitoraggio della QoS su griglia
2. Lo studio di patterns di coordinamento fra un processo di gestione del workflow di una applicazione distribuita su griglia ed un insieme di processi che adattano (tramite wrapping) moduli di libreria (grid-enabled legacy code
3. La modellazione di un processo di integrazione (wrapping) di codice applicativo legacy (sorgente, kernel computazionale) in un elemento software intermedio (modulo) e successivamente in componenti. I componenti del modello si conformano alle regole di composizione e di controllo di un framework a componenti standard (es. Web Services) con estensioni per la gestione da parte di un Manager esterno (es. ASSIST Application Manager) della QoS.
4. La definizione di un linguaggio di meta-descrizione di elementi software. La meta-descrizione costituisce una sorta di documento di identità di un elemento software che, per trasformazioni successive, fornisce informazioni utili durante tutto il suo ciclo di vita

(integrazione, deployment, esecuzione). La descrizione si riferisce sia alla funzionalità del software che ad aspetti non funzionali quali performance, dominio applicativo, tecnologia di codifica.

5. Strumenti di un SDK per la integrazione di codice legacy in componenti WebServices e loro meta-descrizione

Nel seguito sono esposti i risultati ottenuti per ognuna delle linee di intervento sopra indicate.

Integrazione di codice

Per l'integrazione di codice generico non nativamente sviluppato nell'ambiente di programmazione a componenti ASSIST [2] è necessario l'utilizzo di una metodologia rigorosa che, a partire da elementi software di varia natura (codice sorgente, codice oggetto, eseguibili) e realizzati in vari linguaggi di programmazione (C, C++, Fortran), permetta di creare un componente software ospitato in un framework di riferimento (Web Services [3]). Al fine di rendere il più agevole possibile l'adozione di tale metodologia, si sta cercando di progettare un ambiente software che, in maniera automatica e/o assistita, segua l'utente durante il processo d'integrazione. Tale ambiente sarà sviluppato secondo un approccio iterativo in modo da poter implementare ad ogni iterazione un caso specifico del processo di integrazione. Ciò permette di realizzare alcuni degli scenari di integrazione ben specifici con la possibilità di estensioni future.

La soluzione adottata si basa sulla definizione rigorosa di alcuni ***software element*** da utilizzare come risultati intermedi del processo di integrazione [4]. Il primo passo consiste nell'analizzare il software legacy da integrare (***legacy code***) e nel ricondurre tale elemento in uno o più *software element* introducibili nel processo d'integrazione. Una volta ottenuto l'elemento di partenza si procede nelle varie fasi di produzione in maniera incrementale, estendono via via l'elemento software con caratteristiche di interfacciamento, di controllo, di parallelismo, di adattamento e di composizione fino ad ottenere un package (***meta-component***) contenente librerie e moduli necessari per il travestimento da componente nel framework di riferimento.

In seguito sono riportati i risultati ottenuti lungo questa linea che sono suddivisi nei seguenti sotto-task:

- definizione della metodologia d'integrazione end-to-end;
- definizione del modello di descrizione degli elementi software

- definizione degli elementi software che intervengono nei vari stadi della metodologia;
- definizione del linguaggio di descrizione degli elementi software;
- librerie e tools di supporto alla descrizione ed all’ integrazione di elementi software

Il Processo d'Integrazione End-To-End

Il processo d'integrazione end-to-end ha come obiettivo la realizzazione di un componente ospitabile in un framework di riferimento, utilizzabile in una applicazione d'interesse.

Il processo d'integrazione è un processo incrementale, in cui le singole fasi portano alla realizzazione di elementi software intermedi, riusabili in percorsi d'integrazione differenti. Il punto di partenza è un elemento software (*legacy-code*) opportunamente preparato (wrappato) per l'integrazione. Le varie fasi del processo estendono l'elemento software di partenza con caratteristiche di interfacciamento, di controllo, di parallelismo, di adattamento e di composizione.

Il processo end-to-end può essere schematizzato secondo il diagramma a pagina seguente (fig.2). Le classi di elementi software che intervengono nel processo d'integrazioni sono:

- ***legacy code***
- ***kernel***
- ***module*** (service,adapter,control)
- ***component*** (meta,virtual,concrete)

Il ***legacy code*** è l'elemento software di partenza. Il *legacy code* può essere in codice sorgente, in codice oggetto, in codice eseguibile o dinamicamente collegabile. Dal *legacy code* è possibile estrarre (unwrapping) un nucleo computazionale (*kernel*) oppure effettuare un wrapping direttamente verso un *module*.

Il ***kernel*** è un elemento software architetturale di computazione che presenta una singola interfaccia di chiamata. Il *kernel* va inserito in un *module*.

Il ***module*** è un elemento software architetturale di assemblaggio, orientato al soddisfacimento di un requisito funzionale o non-funzionale ben preciso. Il *module* presenta sia interfacce di servizio che di memoria e di I/O. Il *module* può essere un'unità di servizio (simple/parallel service), un'unità di adattamento (application/relevant data adapter) o un'unità di controllo (control). Un *module* di servizio normalmente contiene un *kernel* per elaborare dei dati applicativi, eventualmente accedendo a risorse di memorizzazione esterne. Tale modulo può essere parallelizzato secondo due strategie: tramite parallelizzazione del kernel o tramite parallelizzazione dell'intero modulo. Il *module* va inserito in un *component*.

Il ***component*** è un elemento software architetturale di composizione, contenente almeno un modulo di servizio (simple/parallel) che ne realizza le funzionalità applicative. Il *component* è definito a vari livelli di esistenza sulla base degli aspetti applicativi, di qualità del servizio e di framework considerati. Il ***meta-component*** deriva dall'assemblaggio (incrementale) di moduli di servizio (simple/parallel), di moduli di adattamento applicativi (adapter) e di moduli di controllo (control). A tale livello di esistenza, il componente è in grado di gestire gli aspetti applicativi (interfacciamento dei dati) e di qualità del servizio (pattern di controllo) inoltre definisce la struttura del grafo astratto dei processi in termini di nodi meta/virtuali (ossia con gradi di libertà sul mapping fisico dei processi). Il ***virtual-component*** deriva dalla realizzazione dei meccanismi di comunicazione del *meta-component* secondo le regole del framework di riferimento. A tale livello di esistenza, il componente è in grado di gestire gli aspetti architetturali (porte,protocolli) e di coordinamento (modelli di controllo del ciclo di vita). Infine, il ***concrete-component*** deriva dall'istanziazione del *virtual-component* secondo un preciso grafo astratto dei processi in termini di nodi fisici (ossia con vincoli di piattaforma d'esecuzione).

L'approccio incrementale all'integrazione di codice legacy permette di modellare il processo d'integrazione come una sequenza di attività, ciascuna volta all'arricchimento, alla trasformazione o all'assemblaggio di elementi software. Il seguente diagramma di attività (fig.1) mostra il processo d'integrazione in termini di pipeline di lavorazione di elementi software:

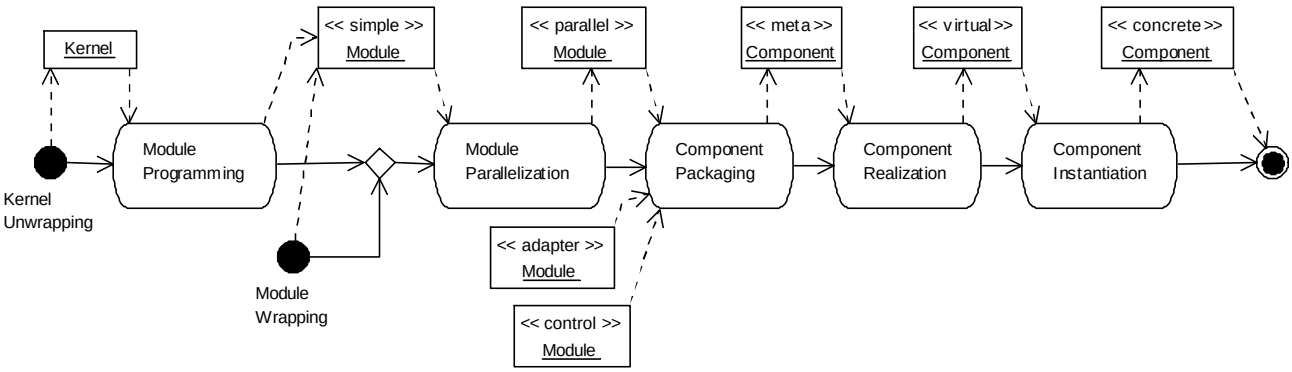


Fig.1 Diagramma di attività del processo d'integrazione

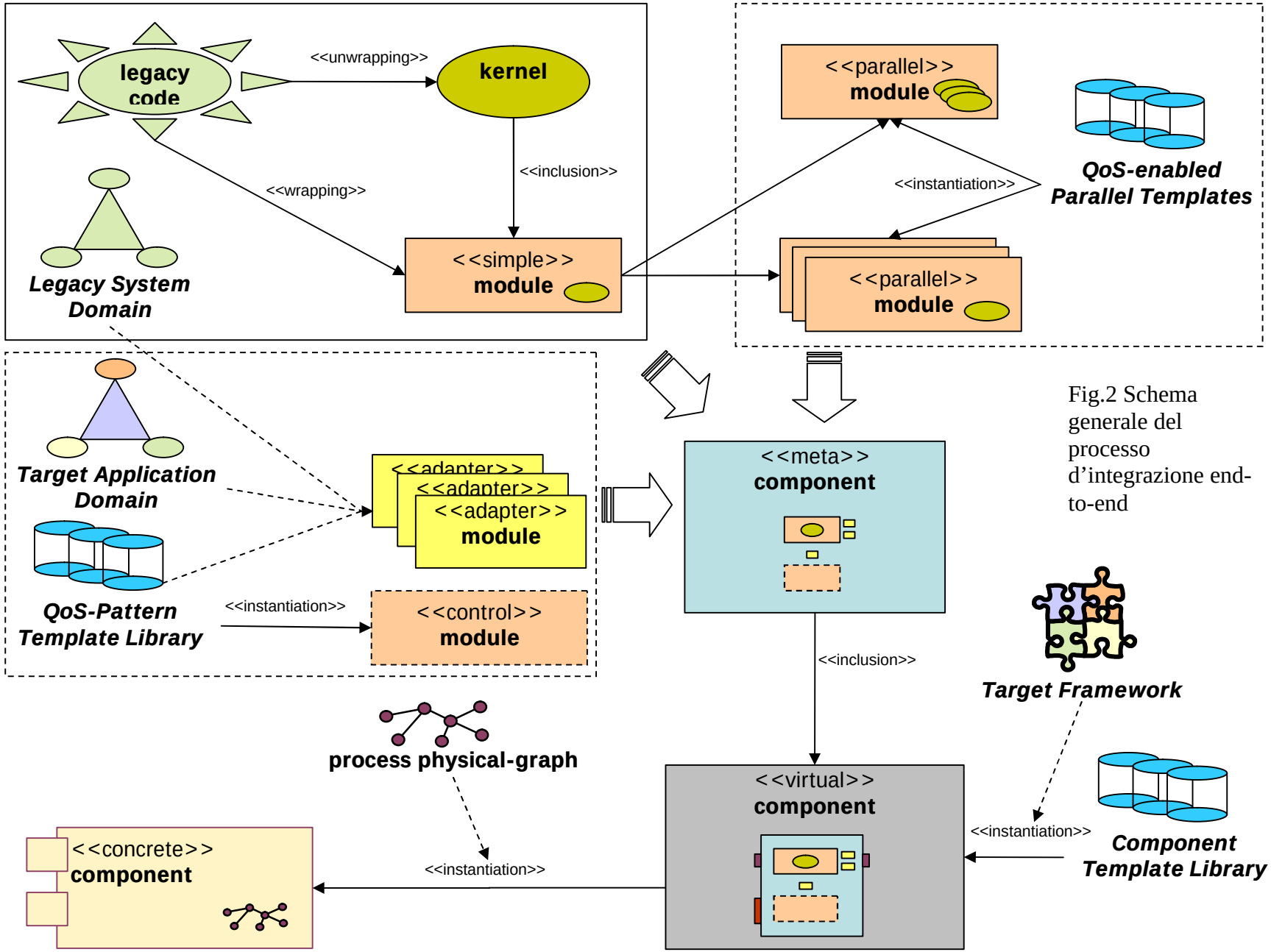


Fig.2 Schema generale del processo d'integrazione end-to-end

Modello di Descrizione di Elementi Software

Il concetto di descrizione di un elemento software sta alla base dell'approccio proposto, essa non si riferisce solamente alla descrizione delle funzionalità del software ma anche ad aspetti non funzionali quali performance, dominio applicativo, tecnologia di partenza. E' importante sottolineare che la meta-descrizione costituisce una sorta di documento di identità di un elemento software che, per trasformazioni successive, fornisce informazioni utili durante tutto il ciclo di vita dell'elemento software [5].

Il modello presentato nella descrizione di un elemento software indirizza tre aspetti fondamentali dell'elemento, quali:

- *Anatomia*
- *Fisiologia*
- *Ontologia*

L'*Anatomia* riguarda i particolari costruttivi ed implementativi dell'elemento software ed è dedicata principalmente ai sistemi di gestione delle risorse. Con il termine ***build*** indichiamo tale aspetto della descrizione che comprende principalmente l'insieme del codice software ***softwareCode*** costitutivo dell'elemento, con le eventuali dipendenze ***dependencies***, ed il grafo dei processi ***abstractGraph***.

La *Fisiologia* riguarda i particolari di funzionalità ed attività dell'elemento software ed è dedicata principalmente ad sistemi di composizione delle risorse. Con il termine ***behaviour*** indichiamo tale aspetto della descrizione che comprende principalmente l'insieme delle interfacce ***interface*** esibite ***provide*** o adottate ***use*** dall'elemento, dei servizi ***service*** realizzati o richiesti nonché delle modalità di attivazione ***activation***. In particolare, la descrizione delle interfacce può riferirsi ad alcuni stereotipi che ne chiarificano ed implicitamente ne definiscono il modello di utilizzo nell'ambito di un dominio o namespace di riferimento. Il servizio rappresenta l'implementazione di una interfaccia nel framework di riferimento, l'attivazione rappresenta la procedura, il meccanismo software o il servizio nel framework di riferimento che permette di avviare l'elemento. Nel seguito si farà indicazione ai seguenti stereotipi di interfacce individuate nelle metodologia di integrazione:

- ***svc***, interfaccia funzionale di servizio
- ***mem***, interfaccia di memoria applicativa strutturata
- ***env***, interfaccia di memoria d'ambiente organizzata secondo un modello associativo

- ***i/o***, interfaccia di memoria sequenziale ad accesso diretto
- ***event***, interfaccia ad eventi strutturata
- ***intro***, interfaccia non-funzionale d'introspezione organizzata secondo un modello associativo

L'*Ontologia* riguarda i particolari interpretativi, semantici e di utilizzo dell'elemento software ed è dedicata principalmente ai sistemi di gestione della conoscenza (KMS) ed ai sistemi di scoperta delle risorse. Con il termine ***semantic*** indichiamo tale aspetto della descrizione che comprende principalmente l'insieme delle definizioni ***definitions***, dei significati ***meanings*** e delle ontologie ***ontologies*** nell'uso dell'elemento.

Elementi Software del Processo di Integrazione

Il presente paragrafo tenta di definire gli aspetti architetturali degli elementi software identificati nella metodologia di integrazione nonché di evidenziarne le proprietà da meta-descrivere.

Il Legacy Code

Con il termine ***legacy code*** denotiamo l'insieme del software in possesso e correntemente utilizzato da una organizzazione, che si desidera integrare in componenti per nuove applicazioni secondo le emergenti tecnologie. E' importante sottolineare che il termine *legacy code* non si riferisce solamente a elementi software di cui si dispone del codice sorgente. I possibili stati di esistenza di un software legacy possono essere:

- **codice sorgente**
- **codice oggetto**
- **codice eseguibile (nativo)**
- **codice dinamicamente collegabile**
- **codice intermedio,** o codice eseguibile non-nativo

Un software legacy in codice sorgente si presenta normalmente come un package organizzato in vari livelli di directory, contenente più file, alcuni sorgenti in senso stretto altri di ausilio alla compilazione. Il codice sorgente è caratterizzato dal linguaggio di programmazione in cui è espresso, nonché dal sistema operativo su cui è portabile. Dal software legacy in sorgente è

possibile identificare, individuare, enucleare ed importare porzioni di codice pre-strutturato in *kernel* computazionali o direttamente in *module* applicativi, secondo i requisiti richiesti.

Un software legacy in codice oggetto si presenta normalmente organizzato in librerie, ossia in package che forniscono in maniera semplice e diretta tutte le informazioni sui singoli elementi contenuti. Il codice oggetto è esclusivamente caratterizzato dall'architettura hardware (CPU) per cui è stato generato. Informazioni aggiuntive possono derivare dal compilatore utilizzato per la sua generazione. Dal software legacy in formato oggetto è possibile solamente identificare, individuare ed isolare porzioni di codice corrispondenti a *kernel* computazionali o wrappabili direttamente in *module* applicativi.

Un software legacy in codice intermedio, eseguibile o dinamicamente collegabile (DLL) è fortemente connesso all'ambiente software (S.O., VES, VM, DLLs, etc) nel quale è stato sviluppato. Il codice intermedio è caratterizzato esclusivamente dal linguaggio in cui è espresso (.Net-Assembly, Java-ByteCode, etc) mentre se il codice è in forma nativa è altresì dipendente dall'architettura hardware per la quale è stato generato. Da tale software legacy è possibile solamente identificare, individuare ed isolare delle parti wrappabili direttamente in *module* applicativi, con l'eventuale dipendenza da runtimer specifici (JVM), nel caso di codice intermedio, o tramite soluzioni ad hoc (JIT,JNI).

Il Kernel

Un ***Kernel*** è un elemento software atomico di computazione stateless, privo di interfacce di I/O e di memoria. Il *kernel* non emette eventuali eccezioni e termina il suo flusso di esecuzione esattamente al ritorno dalla sua chiamata di attivazione.

Un *kernel* ha una complessità dipendente sia da un numero finito di gradi di libertà, ciascuno derivabile da un sotto insieme dei parametri di chiamata, sia, parzialmente, dal valore particolare dei dati da elaborare.

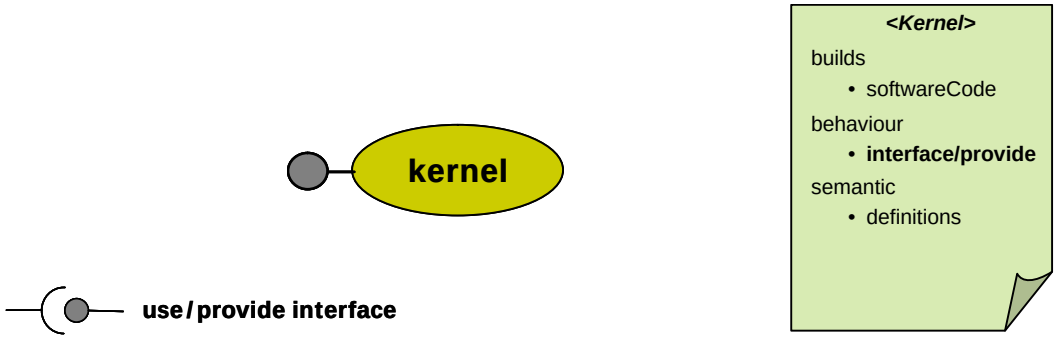


Fig.3 Caratteristiche di un kernel

Il *kernel* è caratterizzato da una *softwareCode* integrante il codice funzionale e può essere corredato da un metodo non-funzionale che esplicita le variabili indipendenti da cui dipende la sua complessità. Tale metodo è eventualmente presente in un ulteriore *softwareCode* allegato.

Il *kernel* presenta un'unica interfaccia *provide* corrispondente al metodo di chiamata. Tale metodo ha una signature del tipo:

```
extern "C" int kernel-call(param-list)
```

dove il qualificatore extern "C" indica che l'interfaccia ha un identificatore univoco, indipendente dal compilatore, *kernel-call* rappresenta il nome del metodo di chiamata e *param-list* è la lista dei parametri richiesti:

```
param-list := param | param, param-list
param := param-type param-name
```

Il *kernel* termina la sua chiamata restituendo un valore intero negativo in caso di errore o non-negativo in caso di esecuzione a buon fine.

I parametri di un *kernel* possono essere di un tipo primitivo (bool, char, int, float, double, …), di un tipo o contenitore standard (string, list, vector, …) o di un tipo utente (struct) inteso come semplice aggregazione di altri tipi. I parametri di un *kernel* possono essere passati per valore o per

riferimento. Nella descrizione semantica per un utilizzo ragionevole in un ipotetico scenario di esercitazione automatica del *kernel*, ciascun parametro è classificato in:

- **variabile**, influisce sulla complessità del *kernel* e necessita di variazione
- **adattabile**, non influisce sulla complessità ma necessita di variazione
- **costante**, non influisce sulla complessità e non necessita di nessuna variazione

I parametri variabili influiscono sulla complessità e necessitano di una variazione da descrivere in termini dei valori assumibili o del range di validità del parametro, insieme ad alcuni possibili valori tipici e/o significativi.

I parametri adattabili non influiscono sulla complessità ma necessitano comunque di una variazione per mantenere la coerenza con alcuni parametri variabili. Tale variazione deve essere descritta in termini di una legge funzionale.

I parametri costanti non influiscono sulla complessità, non necessitano di variazione e vanno descritti semplicemente in termini dei valori tipici e/o significativi.

Il Module

Il ***module*** è un elemento software architetturale di assemblaggio, orientato al soddisfacimento di un requisito funzionale o non-funzionale ben preciso. Un *module* può essere un'unità di servizio (simple/parallel service), un'unità di adattamento (application/relevant data adapter) o un'unità di controllo (control).

Il *module* presenta sia interfacce di servizio che di memoria e di I/O, non emette eventuali eccezioni e termina il suo flusso di esecuzione esattamente al ritorno dalla sua chiamata di attivazione. L'interfacciamento segue delle direttive classificabili in:

- **requirement**, direttive obbligatorie
- **recommendation**, direttive opzionali

Dall'insieme delle direttive di requirement derivano le specifiche **minimal compliance** e l'aderenza a tali specifiche è mandatorio mentre l'insieme delle direttive di requirement e di recommendation definisce le specifiche **advanced compliance** e l'aderenza a tali specifiche favorisce l'integrabilità finale del componente.

Specifiche minimal e advanced compliance

Requirement: Le interfacce di servizio presentate da un modulo, in aderenza alle specifiche minimal compliance, corrispondono esclusivamente a metodi di chiamata ed hanno una signature del tipo:

```
int module-call(param-list)
```

dove *module-call* rappresenta il nome del metodo e *param-list* è la lista dei parametri richiesti.

```
param-list := param | param, param-list
param := param-type param-name
```

Il metodo termina la sua attività restituendo un valore intero negativo in caso di errore o non-negativo in caso di esecuzione a buon fine.

I parametri possono essere di un tipo primitivo (bool, char, int, float, double, …), di un tipo o contenitore standard (string, list, vector, …) o di un tipo utente (struct) inteso come semplice aggregazione di altri tipi. I parametri possono essere passati per valore o per riferimento.

Recommendation: I parametri di input devono essere passati per valore o per riferimento costante, i parametri di input/output per riferimento.

Requirement: Le interfacce esterne adottate da un modulo, in aderenza alle specifiche advanced compliance, corrispondono esclusivamente a metodi di chiamata ed hanno una signature del tipo:

```
int use-call(param-list)
```

dove *use-call* rappresenta il nome del metodo e *param-list* è la lista dei parametri richiesti, in analogia con le interfacce di servizio. Il metodo termina la sua attività restituendo un valore intero negativo in caso di errore o non-negativo in caso di esecuzione a buon fine.

Recommendation: Le interfacce di memoria secondo lo stereotipo *mem*, possono essere:

```
template <datatype> int _get(string uri, datatype &data)
template <datatype> int _put(string uri, [const] datatype &data)
```

L'interfaccia *_get* rappresenta un metodo di chiamata per il recupero di dati strutturati da una risorsa esterna, parametrizzata in funzione del tipo *datatype* dei dati in questione. La risorsa è identificata tramite una **URI/URL** *uri*. Il parametro *data* contiene il riferimento ad un area di memoria su cui conservare i dati recuperati.

L'interfaccia *_put* rappresenta un metodo di chiamata per la scrittura (aggiornamento) di dati strutturati su di una risorsa esterna, parametrizzata in funzione del tipo *datatype* dei dati in

questione. La risorsa è identificata tramite una **URI/URL** *uri*. Il parametro *data* contiene il riferimento all'area di memoria contenente i dati di input (input/output).

Le chiamate restituiscono un valore intero negativo in caso di errore o non negativo in caso di esecuzione a buon fine.

Recommendation: Le interfacce di memoria secondo lo stereotipo *env*, possono essere:

```
int _get-env(string uri, string &value)
int _put-env(string uri, [const] string &value)
```

L'interfaccia ***_get-env*** rappresenta un metodo di chiamata per il recupero di un valore d'ambiente *value* a partire dalla variabile identificata tramite una **URI/URL** *uri*.

L'interfaccia ***_put-env*** rappresenta un metodo di chiamata per la scrittura (aggiornamento) di un valore *value* sulla variabile d'ambiente identificata tramite una **URI/URL** *uri*.

Le chiamate restituiscono un valore intero negativo in caso di errore o non negativo in caso di esecuzione a buon fine.

Recommendation: Le interfacce di memoria secondo lo stereotipo *i/o*, possono essere:

```
int _open(string uri, string mode)
int _read(int handle, string &buffer)
int _write(int handle, const string &buffer)
int _ctrl(int handle, string cmd)
int _close(int handle)
string _strerror(int errno)
```

L'interfaccia ***_open*** rappresenta un metodo di chiamata per l'inizializzazione di una risorsa logica di I/O. La risorsa è identificata tramite una **URI/URL** *url*. Un parametro *mode* può specificare alcuni attributi dell'inizializzazione (read-only, write-only, append, …). La chiamata restituisce un ***handle*** intero positivo che identifica univocamente la transazione I/O con la risorsa, oppure un valore intero negativo in caso di errore.

L'interfaccia ***_read*** rappresenta un metodo di chiamata per la lettura da una risorsa logica di I/O. La transazione di I/O è identificata tramite il parametro *handle*. Il parametro *buffer* contiene il riferimento ad un area di memoria su cui memorizzare i dati letti.

L’interfaccia *_write* rappresenta un metodo di chiamata per la scrittura su una risorsa logica di I/O. La transazione di I/O è identificata tramite il parametro *handle*. Il parametro *buffer* contiene il riferimento ad un area di memoria da cui recuperare i dati da scrivere.

L’interfaccia *_ctrl* rappresenta un metodo di chiamata per la configurazione di una risorsa logica di I/O. La transazione di I/O è identificata tramite il parametro *handle*. Il parametro *cmd* specifica gli attributi della configurazione.

L’interfaccia *_close* rappresenta un metodo di chiamata per la finalizzazione di una risorsa logica di I/O. La transazione di I/O è identificata tramite il parametro *handle*.

Le chiamate restituiscono un codice intero negativo in caso di errore o un valore non negativo in caso di esecuzione a buon fine.

L’interfaccia *_strerror* rappresenta un metodo di chiamata per il recupero di un messaggio testuale relativo al codice di errore *errno* passato per parametro.

Moduli *service*

Un modulo di servizio, o ***service***, è un’unità di elaborazione applicativa, orientata al soddisfacimento di un requisito funzionale di una determinata applicazione o classe di applicazioni, che può effettuare operazioni di I/O su risorse di memorizzazione esterne o può richiedere funzionalità esterne.

Un module di servizio si distingue in:

- **modulo computazionale**
- **modulo entity**

Un modulo computazionale è un modulo senza stato interno che fornisce metodi per elaborare dei dati applicativi, normalmente attraverso un *kernel*, eventualmente accedendo a risorse di memorizzazione esterne. Tale modulo può essere parallelizzato secondo due strategie: tramite parallelizzazione del kernel o tramite parallelizzazione dell’intero modulo.

Un modulo entity è un modulo con stato interno che fornisce metodi per memorizzare ed accedere ad una memoria applicativa.

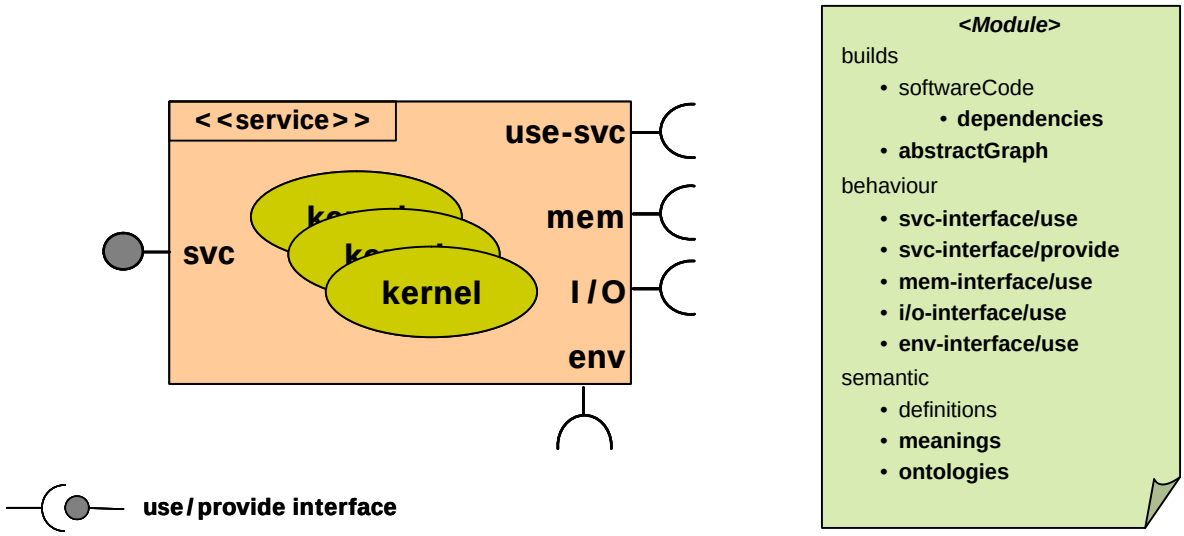


Fig.4 Caratteristiche di un modulo *service*

Un modulo *service* è caratterizzato da:

- un codice software implementativo
- un grafo astratto dei processi, banale, nel caso di moduli seriali (o ***simple-service***), più o meno complesso nel caso di moduli paralleli (o ***parallel-service***)
- interfacce di servizio ed interfacce di memoria. In particolare, un modulo *service* espone metodi di chiamata (svc-interface/provide) mentre accede a servizi esterni (svc-interface/use) ed a memorie di vario tipo (mem, i/o, env, -interface/use)
- definizioni, significati ed ontologie nelle modalità d'utilizzo

Moduli *adapter*

Un modulo di adattamento, o ***adapter***, è un'unità di elaborazione, orientata alla trasformazione, al coordinamento e all'adeguamento di una memoria applicativa tra domini diversi. L'adattamento può riguardare una memoria dati strutturata (DATA-adapter), un canale logico di I/O (VIO-demux) o un memoria di valori d'ambiente (env-adapter).

Un modulo DATA-adapter svolge principalmente una funzione di trasformazione, nei dati e nella semantica di utilizzo, dal dominio applicativo di partenza (legacy-system-domain) verso il nuovo dominio applicativo di riferimento (target-application-domain), e viceversa.

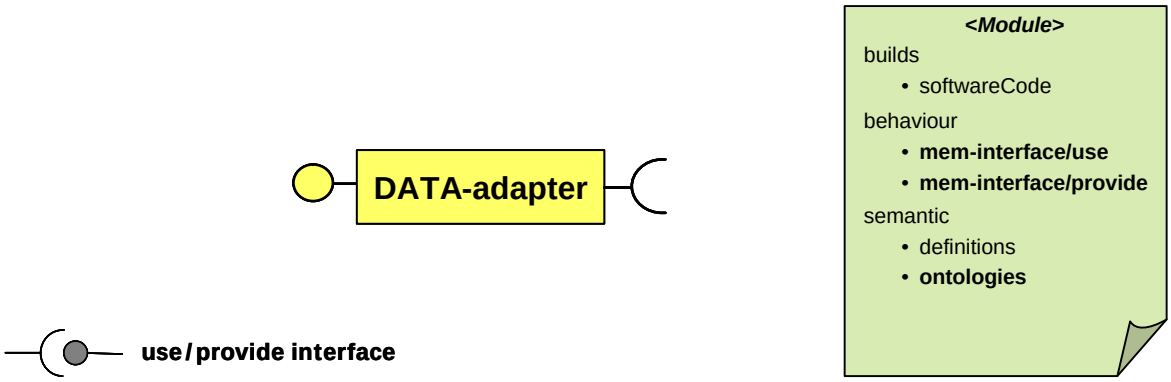


Fig.5 Caratteristiche di un modulo *DATA-adapter*

Un modulo DATA-adapter è caratterizzato da:

- un codice software implementativo
- interfacce di memoria. In particolare, un modulo DATA-adapter espone metodi di chiamata nel legacy-system-domain (mem-interface/provide) ed accede a memorie esterne nel target-application-domain (mem-interface/use)
- definizioni ed ontologie nelle modalità d'utilizzo

Un modulo VIO-demux svolge principalmente una funzione di coordinamento dei canali logici di I/O attraverso un mapping delle risorse tra il dominio applicativo di partenza (legacy-system-domain) ed il nuovo dominio applicativo di riferimento (target-application-domain).

Un modulo VIO-demux è caratterizzato da:

- un codice software implementativo
- interfacce di memoria. In particolare, un modulo VIO-demux espone metodi di I/O per i canali logici nel legacy-system-domain (i/o-interface/provide) ed accede a canali esterni di I/O nel target-application-domain (i/o-interface/use)

- definizioni e significati nelle modalità d'utilizzo

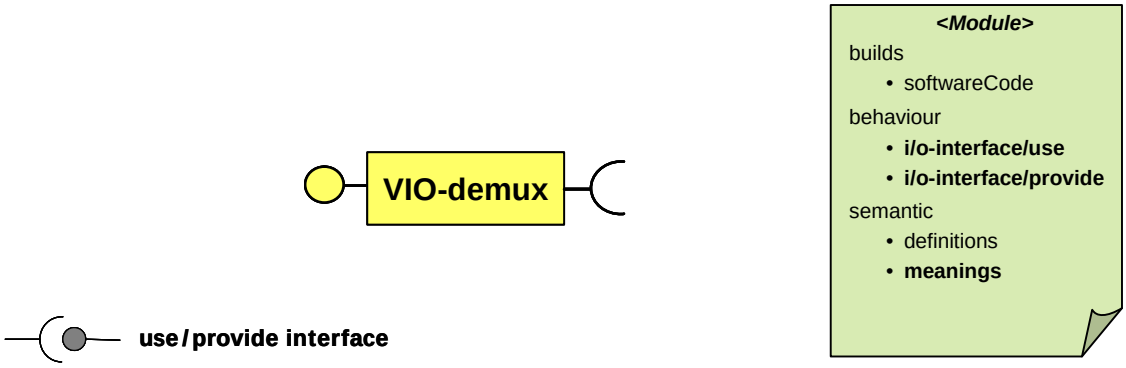


Fig.6 Caratteristiche di un modulo *VIO-demux*

Un modulo env-adapter svolge principalmente una funzione di adeguamento dei valori d'ambiente, e della relativa semantica di utilizzo, dal dominio applicativo di partenza (legacy-system-domain) verso il nuovo dominio applicativo di riferimento (target-application-domain), e viceversa.

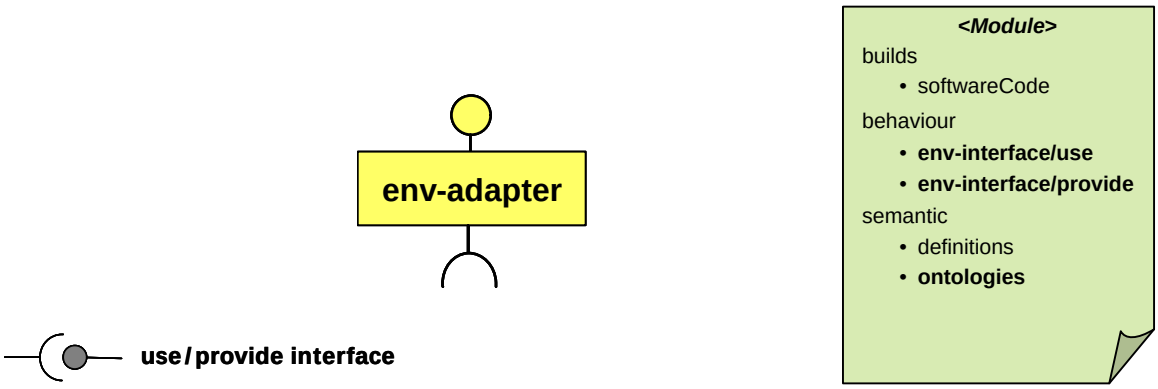


Fig.7 Caratteristiche di un modulo *env-adapter*

Un modulo env-adapter è caratterizzato da:

- un codice software implementativo
- interfacce di memoria. In particolare, un modulo env-adapter espone metodi di accesso a variabili d'ambiente nel legacy-system-domain (env-interface/provide) ed accede alla memoria d'ambiente nel target-application-domain (env-interface/use)
- definizioni ed ontologie nelle modalità d'utilizzo

Moduli *control*

Un modulo di controllo, o ***control***, è un'unità di processamento event-driven, orientata al mantenimento delle variabili d'esecuzione (relevant-data) che intervengono nell'elaborazione effettuata dai moduli di servizio. Un *control* ha un comportamento reattivo modellabile attraverso un macchina a stati e segue dei precisi pattern di coordinamento [6].

Un *control* può contenere un kernel per la valutazione del costo computazionale del modulo di servizio, accessibile attraverso un meccanismo di introspezione [7].

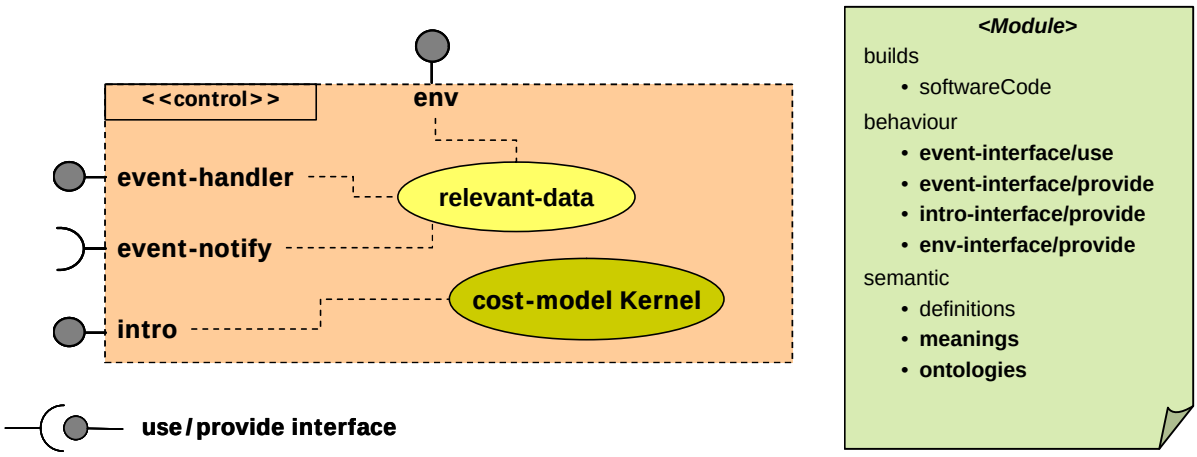


Fig.8 Caratteristiche di un modulo *control*

Un modulo *control* è caratterizzato da:

- un codice software implementativo
- interfacce ad eventi. In particolare, un modulo *control* espone metodi per la sottomissione di eventi (event-interface/provide) mentre accede a risorse esterne per la notifica di eventi (event-interface/use)
- interfacce di introspezione. In particolare, un modulo *control* espone metodi per le chiamate di introspezione dei relevant-data o del costo computazionale
- interfacce di memoria. In particolare, un modulo *control* espone metodi di accesso alle variabili di esecuzione (relevant-data) sotto forma di variabili d'ambiente
- definizioni, significati ed ontologie nelle modalità d'utilizzo

Il Component

Il ***component*** è un elemento software architetturale di composizione conforme alle regole di connessione e di controllo di un framework di riferimento, orientato al soddisfacimento di un funzionalità applicativa secondo criteri di qualità del servizio (QoS).

Il modello di riferimento per il componente tiene conto del processo d'integrazione di codice secondo *kernel* e *module* fin quì presentato. Lo scenario di utilizzo del componente integrato prevede chiamate RPC sincrone (secondo il modello d'interazione solicit/response) effettuate dai componenti di una applicazione distribuita, organizzata secondo un modello a Workflow con gestore (Workflow Manager [8]). In tale ottica, i componenti del modello, derivanti dall'integrazione di module legacy nell'applicazione desiderata, si conformano alle regole di composizione e di controllo di un framework a componenti standard (es. WebServices [2]) con estensioni per il controllo del ciclo di vita [9], la configurazione dinamica (riconfigurazione [10]) e la gestione e della qualità del servizio da parte di un Manager esterno (es. ASSIST Application Manager [11]).

Il *component* è definito a vari livelli di esistenza sulla base degli aspetti applicativi, architetturali e delle risorse fisiche utilizzate. Il *meta-component* rappresenta un'istanza del modello di componente secondo i requisiti applicativi dal punto di vista funzionale, del controllo e della QoS, da completare nel *virtual-component* con il supporto architetturale secondo un framework di riferimento e da mappare nel *concrete-component* sulle risorse fisiche disponibili.

Meta-Component

Il modello di componente presentato prevede tre classi di interfacciamento e due flussi logici di esecuzione (thread) per processo:

- Un'interfacciamento funzionale per le chiamate applicative
- Un'interfacciamento di introspezione per la lettura dei parametri di esecuzione
- Un'interfacciamento ad eventi per il controllo del ciclo di vita e la gestione della qualità del servizio
- Un thread di servizio per lo svolgimento dell'attività funzionale;
- Un thread di controllo per lo svolgimento dell'attività non-funzionale

Il ***meta-component*** si ottiene dall'assemblaggio di moduli di servizio (*simple/parallel service*), per lo svolgimento dell'attività funzionale, di moduli di controllo (*control*) per lo svolgimento dell'attività non-funzionale e di eventuali moduli di adattamento applicativi (*adapter*).

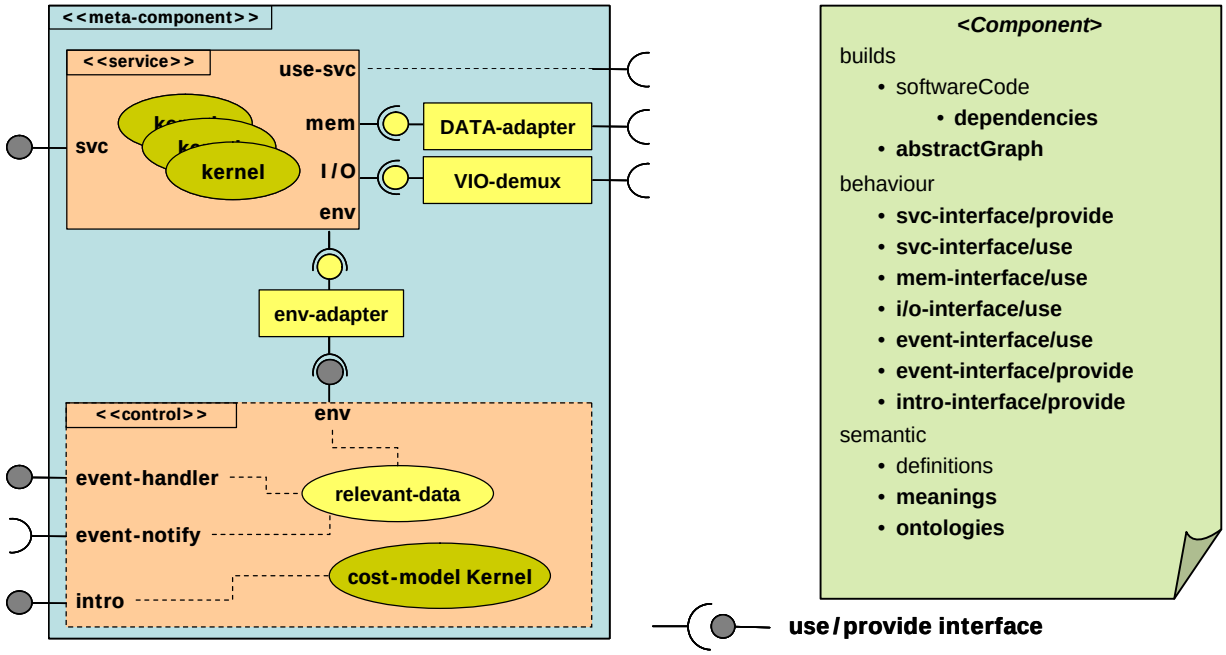


Fig.9 Caratteristiche di un *meta-component*

Le interfacce che né derivano sono l'unione delle interfacce esposte dai singoli moduli costituenti non accoppiate durante il processo di assemblaggio e determinano il livello di integrabilità finale del componente, ossia quelle caratteristiche controllabili e configurabili (I/O, RPC, Grafo dei Processi, Eventi), che incidono sulla flessibilità del pattern di coordinamento applicabile al suo management (Life-Cycle, QoS) attraverso le funzioni del framework.

L'aderenza alle specifiche *minimal compliance* garantisce un livello di integrabilità di base. Tale livello permette la semplice attivazione del componente ed è sufficiente all'applicazione di un pattern per il coordinamento del ciclo di vita.

L'aderenza alle specifiche *advanced compliance* garantisce un livello di integrabilità avanzata. Tale livello permette l'attivazione del componente con configurazione dei relevant-data necessari all'esecuzione, con possibilità di redirezione dei canali di I/O, con controllo delle chiamate esterne (RPC) e con meccanismi di gestione di eventi esterni, ed è necessario per l'applicazione di pattern di coordinamento per la gestione della qualità del servizio [12].

Virtual-Component

Il ***virtual-component*** deriva dalla realizzazione dei meccanismi di comunicazione del *meta-component* secondo le regole del framework di riferimento.

Il modello di componente presentato prevede l'implementazione delle interfacce funzionali attraverso delle porte Solicite/Response e delle interfacce ad eventi attraverso delle porte Subscribe/Notify.

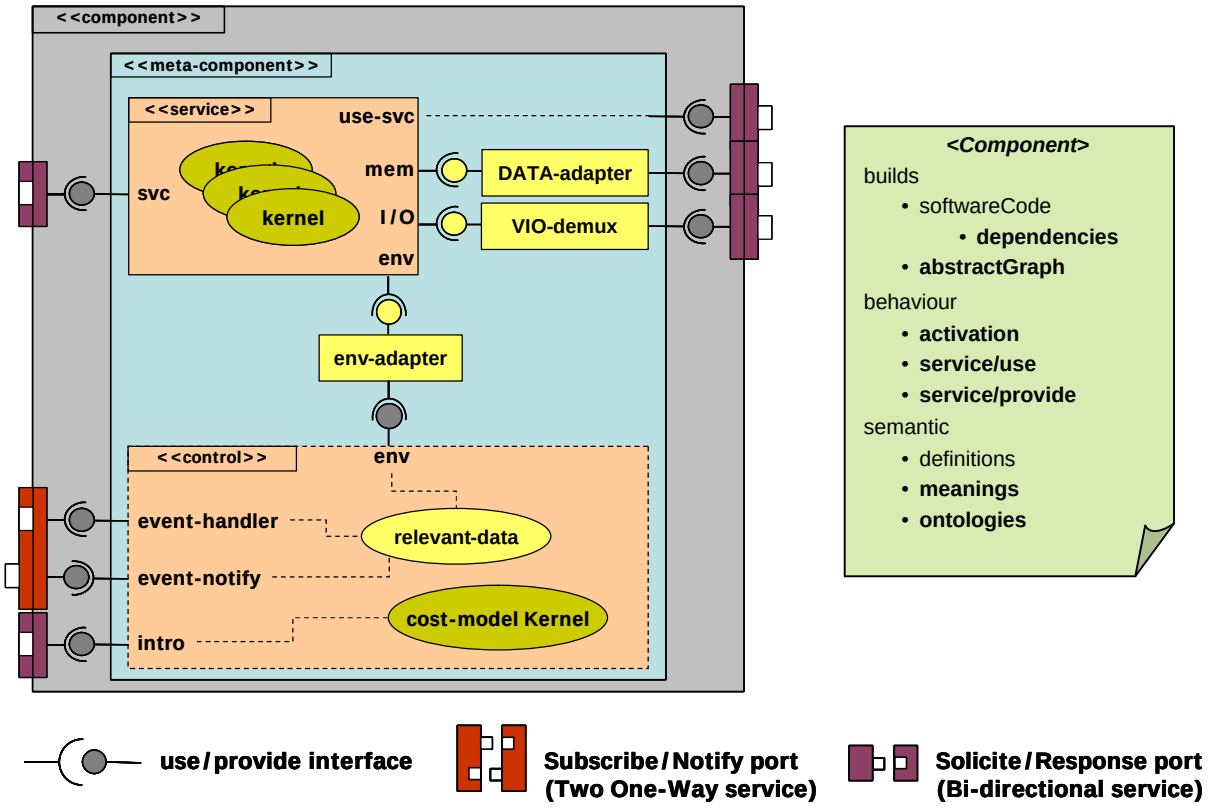


Fig.10 Caratteristiche di un *virtual-component*

Il componente *virtual* è caratterizzato da:

- un codice software implementativo
- un grafo astratto dei processi
- un meccanismo di attivazione, specifico del framework di riferimento
- un insieme di servizi offerti (service/provide) e/o servizi utilizzati (service/use)
- definizioni, significati ed ontologie nelle modalità d'utilizzo

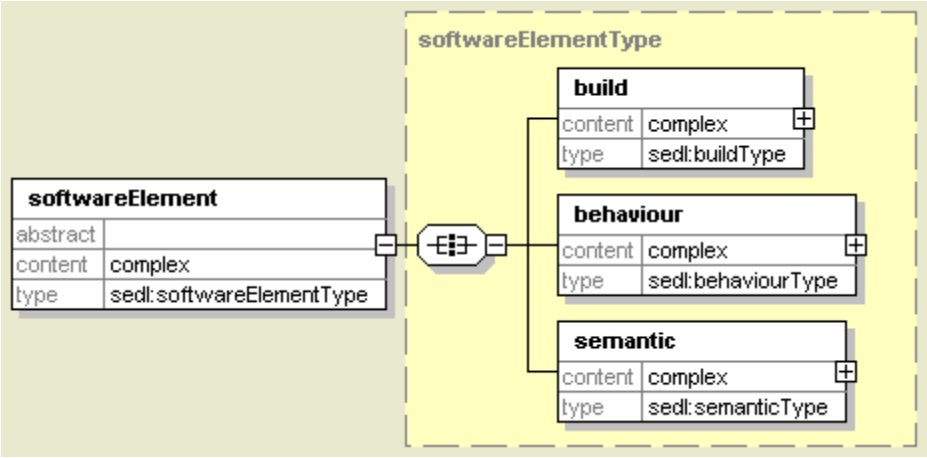
Descrizione di Elementi Software

In questo capitolo è presentato un linguaggio di descrizione per elementi software, secondo il modello discusso nell'ambito dell'integrazione di codice.

Il formalismo utilizzato per il linguaggio è XML [13]. Il formalismo utilizzato per la rappresentazione della sintassi del linguaggio è XML-Schema [14]. E' in corso di studio una rappresentazione della semantica della descrizione in OWL.

SEDL - Software Element Description Language

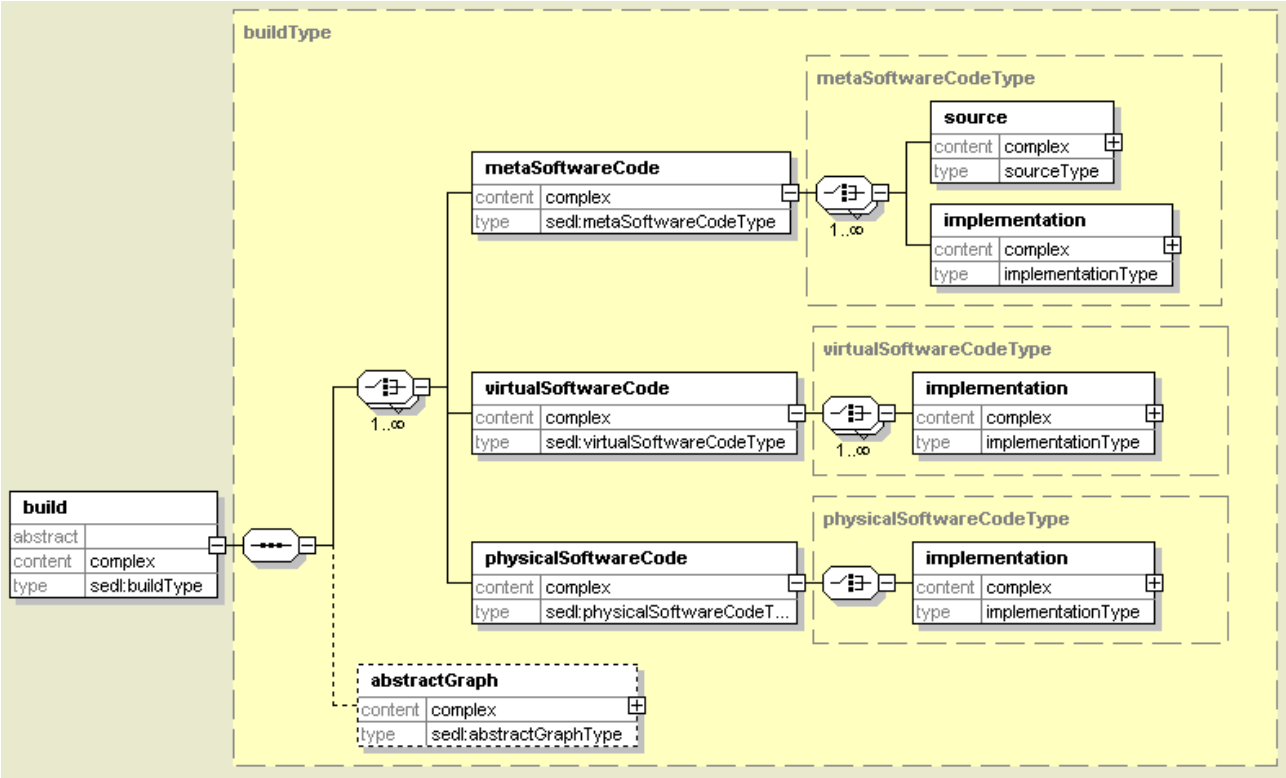
In questo paragrafo è presentato lo schema descrittivo di un documento SEDL/XML. Ogni documento SEDL è costituito da tre sezioni principali contenenti le informazioni relative agli aspetti fondamentali di *Anatomia, Fisiologia* ed *Ontologia* del ***softwareElement***.



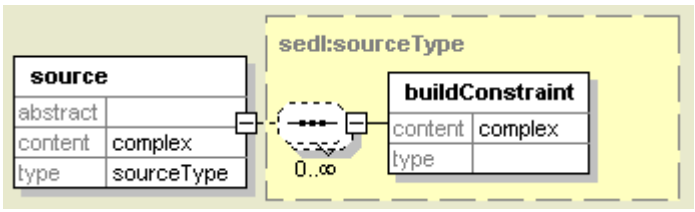
La sezione ***build*** contiene l'insieme del codice software ***softwareCode*** costitutivo dell'elemento software ed il grafo dei processi ***abstractGraph***.

Il *softwareCode* può essere descritto a tre livelli di astrazione:

- il ***physicalSoftwareCode***, contenente una implementazione ***implementation*** specifica del software
- il ***virtualSoftwareContente***, contenente più implementazioni dello stesso software
- il ***metaSoftwareCode***, contenente sorgenti ***source*** ed implementazioni diverse



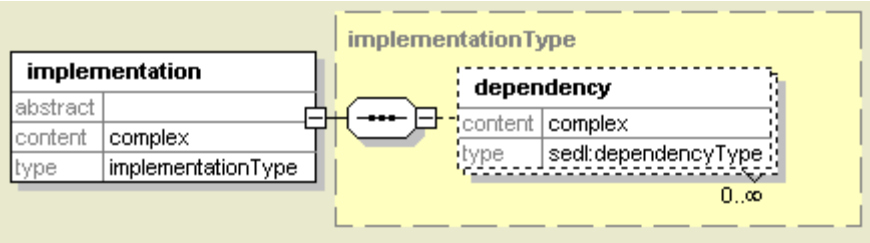
Ogni *source* ha attributi di versione, formato e linguaggio di programmazione e può avere dei vincoli di portabilità verso piattaforme di compilazione.



Attributes of element: **source**			
Name	**Type**	**Use**	**Enum/Pattern**
name	xsd:string	required	
version	xsd:string	optional	
fileLocator	xsd:string	required	
format	xsd:NMTOKEN	required	[Ii]nterface|[Ii]mplementation
language	xsd:NMTOKEN	required	[Cc]([Pp][Pp])?|F|Java

Attributes of element: **buildConstraint**			
Name	**Type**	**Use**	**Enum/Pattern**
compiler	xsd:NMTOKEN	optional	[Gg][Cc][Cc]([23](_[0123456789]+)?)?
machine	xsd:NMTOKEN	optional	([Xx]|(i[3456]))86(-64)?|[Ss][Pp][Aa][Rr][Cc]
os	xsd:NMTOKEN	optional	[Ll]inux_[Rr]ed[Hh]at_7_[123]|[Ll]inux_[Rr]ed[Hh]at_[89]|[Ll]inux_[Ff]edora_[Cc]ore_[123]|[Ss]olaris_[789]

Ogni *implementation* ha attributi di versione, formato, linguaggio di origine, compilatore di costruzione, architettura hardware e sistema operativo di destinazione e può avere delle dipendenze.

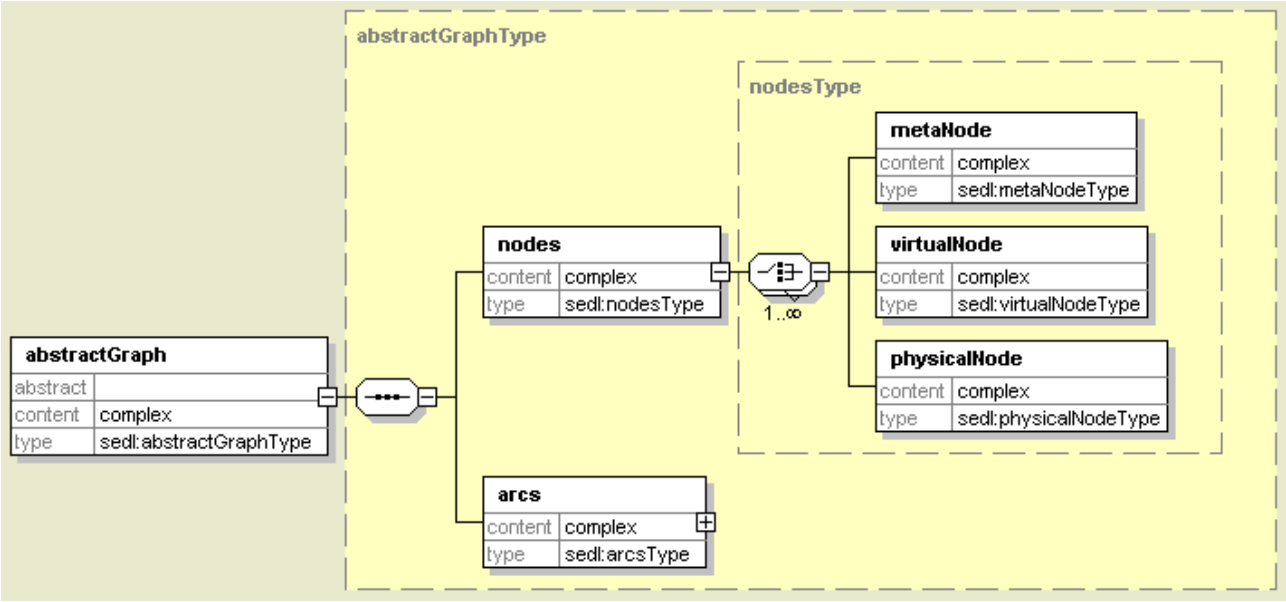


Attributes of element: **implementation**			
Name	**Type**	**Use**	**Enum/Pattern**
name	xsd:string	required	
version	xsd:string	optional	
fileLocator	xsd:string	required	
format	xsd:NMTOKEN	required	[Gg]eneric|[Ee]xecutable|[Dd]ynamic_[Ll]ibrary
language	xsd:NMTOKEN	required	[Gg]eneric|[Cc]([Pp][Pp])?|F|Java
compiler	xsd:NMTOKEN	required	[Gg]eneric|[Gg][Cc][Cc]([23](_[0123456789]+)?)?
machine	xsd:NMTOKEN	required	[Gg]eneric|([Xx]|(i[3456]))86(-64)?|[Ss][Pp][Aa][Rr][Cc]
os	xsd:NMTOKEN	required	[Gg]eneric|[Ll]inux_[Rr]ed[Hh]at_7_[123]|[Ll]inux_[Rr]ed[Hh]at_[89]|[Ll]inux_[Ff]edora_[Cc]ore_[123]|[Ss]olaris_[789]

Attributes of element: **dependency**			
Name	**Type**	**Use**	**Enum/Pattern**
name	xsd:string	required	
version	xsd:string	optional	
fileLocator	xsd:string	required	
format	xsd:NMTOKEN	required	[Gg]eneric|[Ee]xecutable|[Dd]ynamic_[Ll]ibrary

L'*abstractGraph* contiene le informazioni sul grafo dei processi in termini di nodi ***nodes*** ed archi ***arcs***. Un nodo di processo può essere descritto a tre livelli di astrazione:

- il ***physicalNode***, rappresentante un singola risorsa fisica necessaria all'esecuzione del processo su cui mappare un *physicalSoftwareCode* con implementazione adeguata
- il ***virtualNode***, rappresentante un generico nodo multiplo su cui mappare le possibili varianti (implementazioni) di un *virtualSoftwareCode*
- il ***metaNode***, rappresentante un generico nodo multiplo su cui mappare le possibili varianti (sorgenti ed implementazioni) di un *metaSoftwareCode*



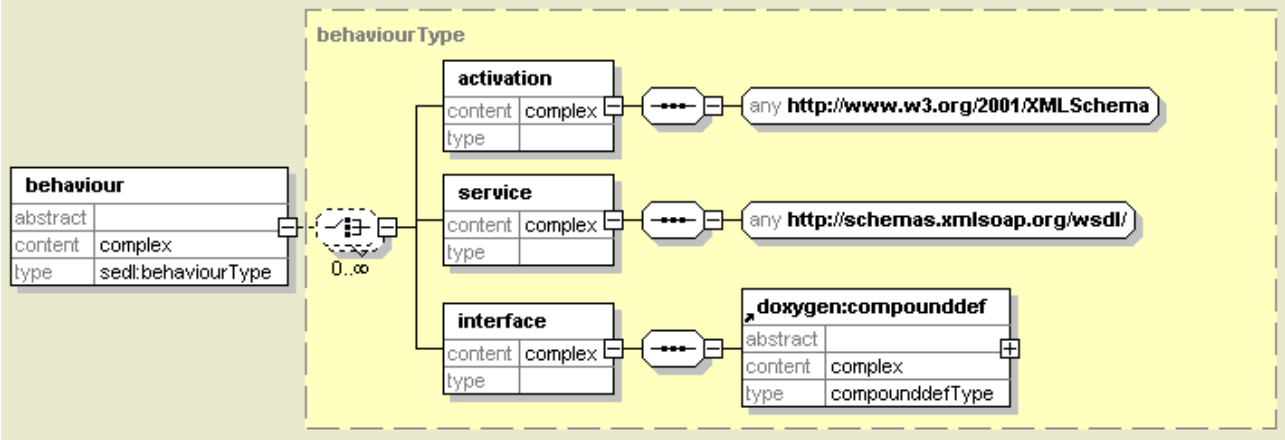
Ad ogni nodo di processo è attribuibile un ruolo di coordinamento. Il *metaNode* ed il *virtualNode* hanno attributi di molteplicità. Il *physicalNode* contiene un riferimento (URI) ad una risorsa fisica.

Attributes of element: **metaNode**			
Name	**Type**	**Use**	**Enum/Pattern**
nodeID	xsd:nonNegativeInteger	required	
metaSoftwareCode	xsd:NCName	required	
role	xsd:NMTOKEN	required	[Gg]eneric|[Mm]aster|[Ss]lave
multiplicityMin	xsd:nonNegativeInteger	required	
multiplicityMax	xsd:positiveInteger	optional	

Attributes of element: **virtualNode**			
Name	**Type**	**Use**	**Enum/Pattern**
nodeID	xsd:nonNegativeInteger	required	
virtualSoftwareCode	xsd:NCName	required	
role	xsd:NMTOKEN	required	[Gg]eneric|[Mm]aster|[Ss]lave
multiplicityMin	xsd:nonNegativeInteger	required	
multiplicityMax	xsd:positiveInteger	optional	

Attributes of element: **physicalNode**			
Name	**Type**	**Use**	**Enum/Pattern**
nodeID	xsd:nonNegativeInteger	required	
physicalSoftwareCode	xsd:NCName	required	
role	xsd:NMTOKEN	required	[Gg]eneric|[Mm]aster|[Ss]lave
resourceURI	xsd:anyURI	required	

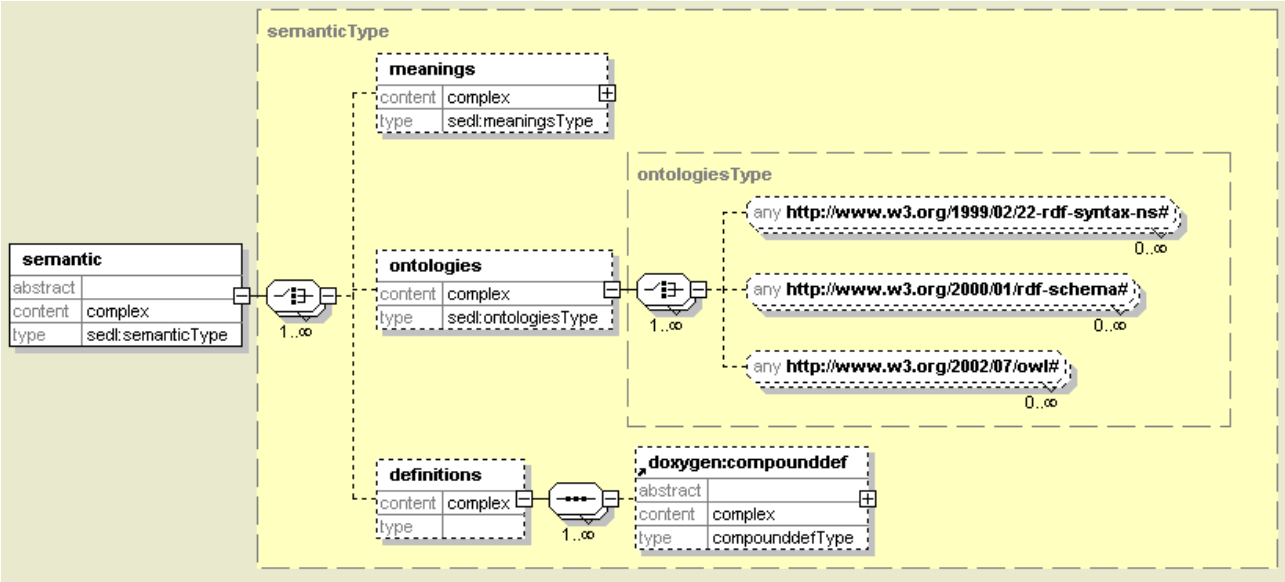
La sezione ***behaviour*** contiene l'insieme delle interfacce ***interface*** esibite (*provide)* o adottate (*use)* dall'elemento software, dei servizi ***service*** realizzati (*provide*)o richiesti (*use*) nonché delle modalità di attivazione ***activation***. Le *interface* sono descritte tramite documenti secondo lo schema **doxygen:compound** [15]. I *service* sono descritti tramite documenti **WSDL** [16]. L'*activation* è descritta tramite documenti in formato **XML-Schema,** contenente le informazioni strutturate su attributi ed elementi della procedura attivazione.



Ad ogni *interface* e *service* è attribuibile uno stereotipo che ne chiarisce ed implicitamente ne definisce il modello di utilizzo nell'ambito di un dominio o namespace *ns* di riferimento.

Attributes of element: **interface**			
Name	**Type**	**Use**	**Enum/Pattern**
kind	xsd:NMTOKEN	required	use provide
stereotype	xsd:string	optional	
ns	xsd:anyURI	optional	

Attributes of element: **service**			
Name	**Type**	**Use**	**Enum/Pattern**
kind	xsd:NMTOKEN	required	use provide
stereotype	xsd:string	optional	
ns	xsd:anyURI	optional	



La sezione ***semantic*** contiene l'insieme delle definizioni ***definitions***, dei significati ***meanings*** e delle ontologie ***ontologies*** nell'uso dell'elemento software. Le *definitions* sono descritte tramite documenti secondo lo schema **doxygen:compound**, con annotazioni tipo JavaDoc [15]. Le *ontologies* sono descritte tramite documenti **OWL** o **RDF** [17]**.**

Tools per l'Ambiente d'Integrazione

In questo capitolo è riportata una breve descrizione dei risultati (parziali) raggiunti nello sviluppo di librerie e tools di supporto all'integrazione di codice, da inquadrare in un panorama più generale che è quello di un Ambiente d'Integrazione di Software Legacy.

Il modello di riferimento per l'Ambiente d'Integrazione prevede lo svolgimento delle varie attività attraverso un sistema esperto di asservimento (integrazione semi-automatica) che guida l'utente nelle varie fasi del processo per mezzo di una interfaccia grafica. In ogni fase, gli elementi software di input e di output sono descritti attraverso il linguaggio SEDL (Software Element Description Language) ad eccezione della prima fase, in cui l'input, il legacy code, è da preparare e da meta-descrivere per la successiva integrazione. Alcune delle fasi sono "quasi-automatiche", ossia l'intervento dell'utente è limitato ad una semplice rifinitura e verifica dei risultati prodotti in automatico, altre sono "semi-automatiche", ossia l'utente è agevolato da schemi prestrutturati (skeleton) di intervento sui risultati parziali prodotti in automatico.

In tale scenario si inseriscono i toolkit presentati per il processo d'integrazione. Il seguente diagramma (fig. 11) né mostra la pertinenza alle varie attività del processo.

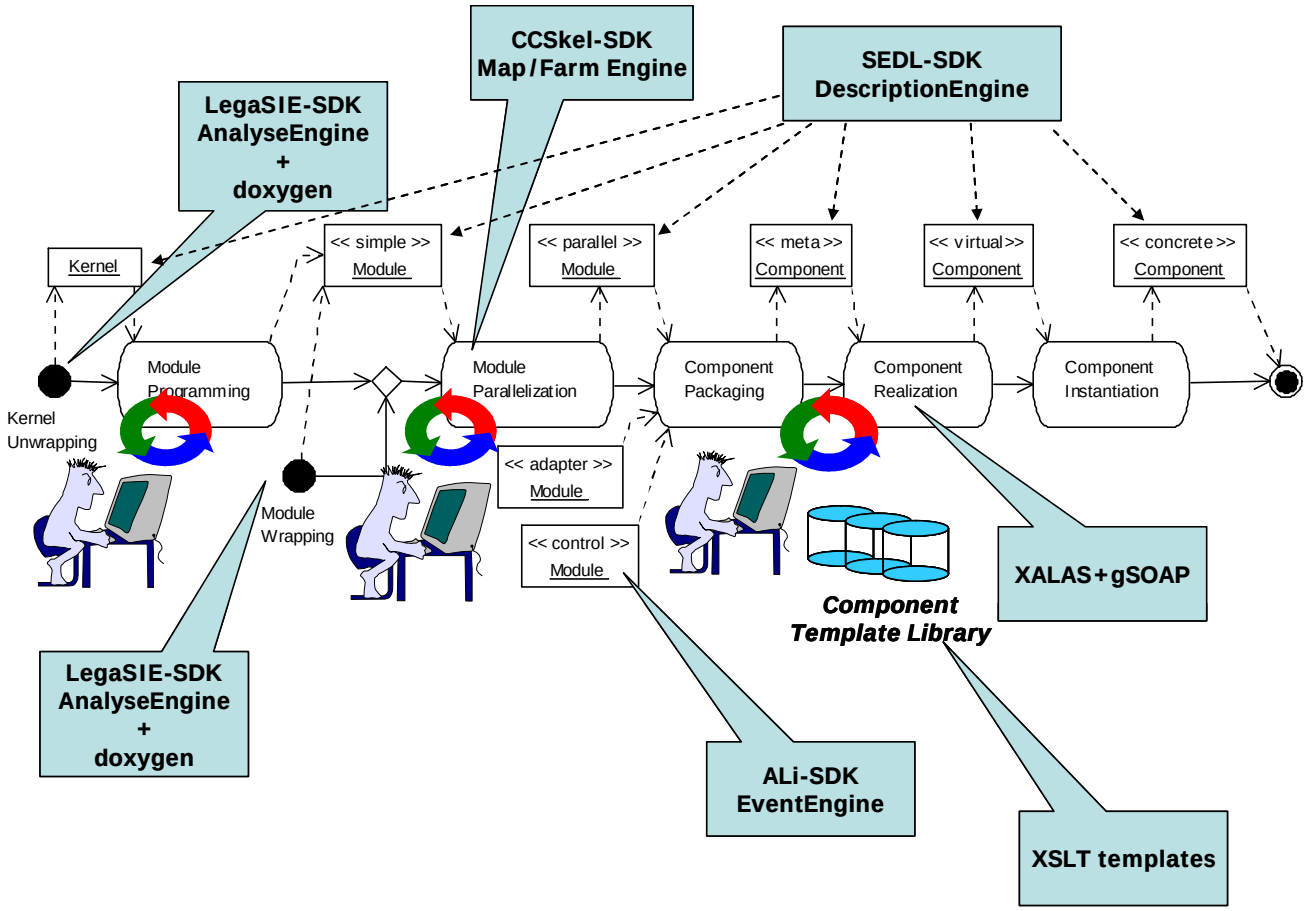


Fig.11 Toolkit per il processo d'integrazione

Il toolkit ***LegaSIE-SDK*** contiene una libreria di metodi per l'estrazione e l'annotazione delle caratteristiche di *Anatomia* e *Fisiologia* di elementi software. L'*AnalyseEngine* permette l'analisi di codice software in sorgente o in oggetto per l'individuazione delle caratteristiche implementative, costruttive e di interfacciamento. In particolare, per l'estrazione delle interfacce di utilizzo è adoperato il tool *doxygen* [10], in maniera immediata, nel caso di code sorgente, ovvero dopo un disassemblaggio dei simboli a partire dal codice oggetto. Il toolkit *LegaSIE* è impiegato in fase di importazione del codice legacy in *kernel* o in *module*.

Il toolkit ***CCSkel-SDK*** contiene una libreria di metodi per la parallelizzazione di moduli legacy. Le classi *MapEngine* e *FarmEngine* forniscono due template parallelizzazione strutturata Farm e Map, sviluppato sinergicamente dallo stesso gruppo nel progetto MIUR 5% [18].

Il toolkit ***Ali-SDK*** contiene una libreria di classi per la realizzazione di moduli di controllo secondo pattern ad eventi per la gestione della QoS [3,19], utilizzato in fase di assemblaggio del *meta-component*.

Il toolkit ***SEDL-SDK*** contiene una libreria di metodi per la manipolazione delle meta-descrizioni di elementi software, utilizzato in ogni fase del processo. La *DescriptionEngine* permette la codifica in linguaggio SEDL di tali meta-descrizioni.

E' in corso di sviluppo un template di componente WebServices da utilizzare nella fase di realizzazione del *virtual-component*. Un insieme di *XSLT-template* [20] definisce la trasformazione di un documento SEDL relativo ad un *meta-component* in una specifica in *gSOAP* [21] delle caratteristiche di una WebServices contenente metodi di servizio per ciascuna delle interfacce funzionali dichiarate nel *meta-component* di partenza. L'utilizzo dell'omonimo strumento *gSOAP* permette, successivamente, l'implementazione della porta SOAP/WebServices [12] da integrare nel *virtual-component*.

Alcuni prototipi di componenti WebServices realizzati ad hoc sono stati testati in GRACE [23].

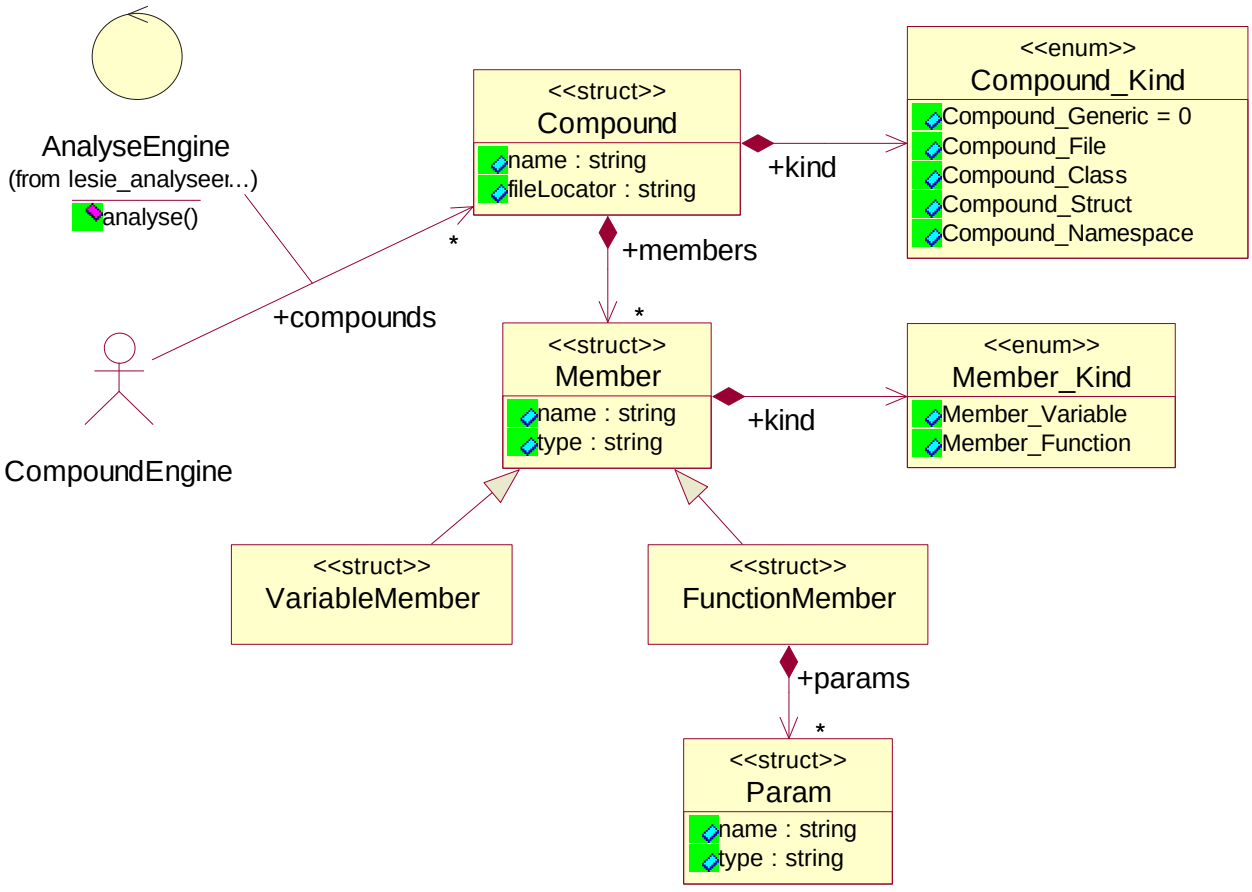
E' in corso di prototipizzazione una piattaforma CAE di supporto all'integrazione di software legacy, denominata GUIDE.

LegaSIE-Toolkit

Il toolkit LegaSIE è una libreria di metodi per l'estrazione e l'annotazione delle caratteristiche di *Anatomia* e *Fisiologia* di elementi software.

La classe principale, ***AnalyseEngine,*** espone metodi per l'analisi di codice software in sorgente o in oggetto e l'individuazione delle caratteristiche implementative, costruttive e di interfacciamento.

Di seguito un diagramma della gerarchia di classi che rappresenta gli elementi di composizione (*Compound*) di un codice software, in termini di file, classi, struct e namespace. Ciascun composto è costituito da un insieme di membri (*Member*) che rappresentano astrazioni di variabili (*VariableMember*) o funzioni (*FunctionMember*). Le funzioni hanno una lista di parametri (*Param*).



Le caratteristiche implementative e di interfacciamento sono direttamente memorizzate tramite le API SEDL.

In Appendice (B) è presentata una documentazione completa delle API LegaSIE.

SEDL-Toolkit

Il toolkit ***SEDL-SDK*** contiene una libreria di metodi per la manipolazione delle meta-descrizioni di elementi software, utilizzato in ogni fase del processo.

La classe principale, ***DescriptionEngine***, espone metodi per la (de)codifica in linguaggio SEDL di tali meta-descrizioni.

Nella pagina successiva è riportato il diagramma della gerarchia di classi che rappresenta l'anatomia (*Build*) di un *SoftwareElement*, in termini di *SoftwareCode* costitutivo dell'elemento e grafo dei processi *AbstractGraph*.

Il *SoftwareCode* è realizzato tramite:

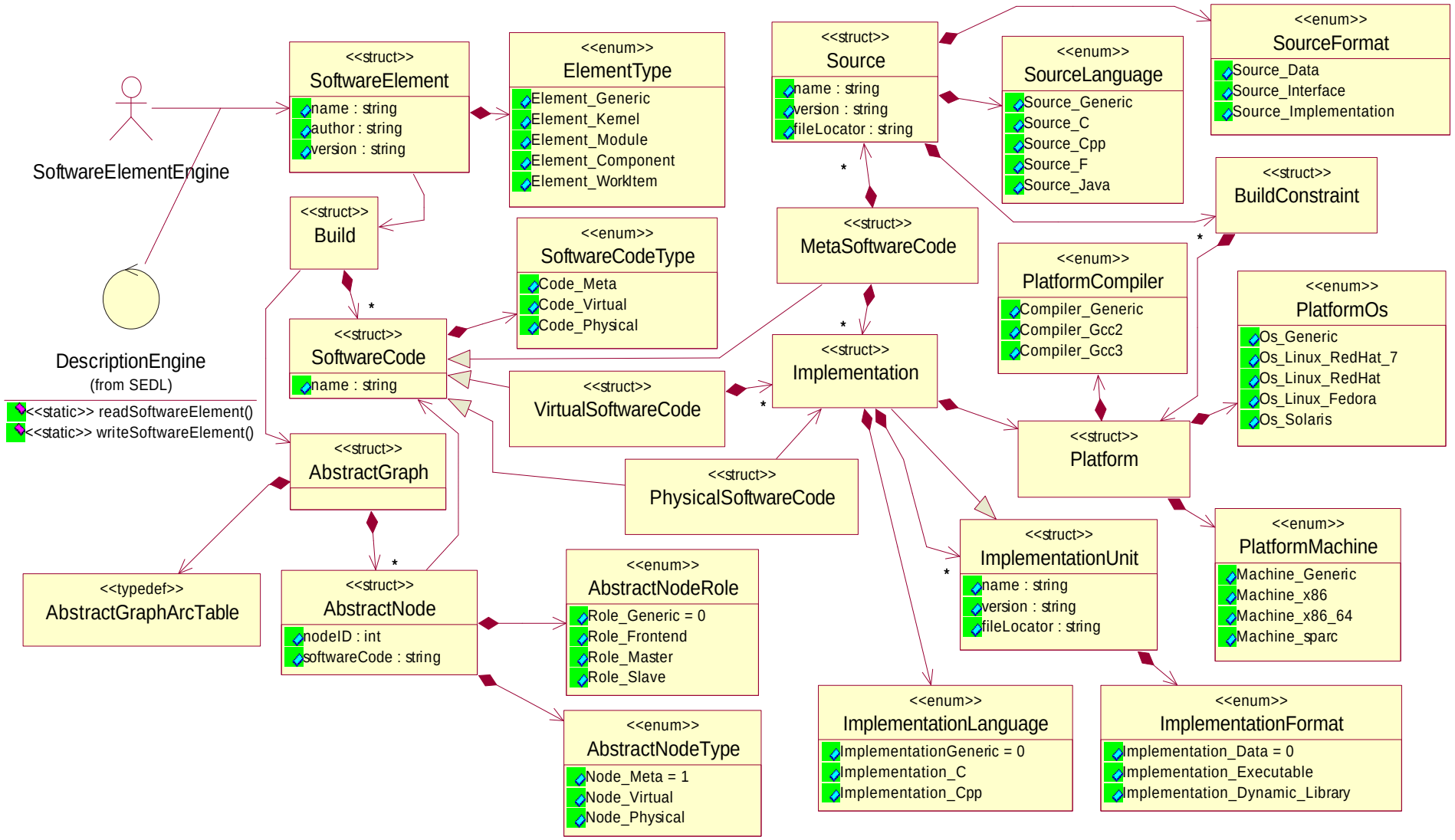
- *PhysicalSoftwareCode*, contenente una implementazione *Implementation*
- *VirtualSoftwareCode*, contenente più *Implementation*
- *MetaSoftwareCode*, contenente sorgenti *Source* ed *Implementation* diverse

Source ed *Implementation* sono rappresentati in termini dei rispettivi formati e linguaggi d'origine (*Source/Implementation Format,Language*) e in termini della piattaforma software di provenienza (*Platform*) a sua volta espressa tramite la triade compilatore, S.O, architettura Hw (*Platform Compiler,Os,Machine*).

Un'*Implementation* realizza una particolare unità implementativa (*ImplementationUnit*) e può avere altre unità implementative sotto forma di dipendenze.

L' *AbstractGraph* è rappresentato in termini di archi e nodi (*AbstractNode*) di categoria Meta/Virtual/Physical collegati ai rispettivi *SoftwareCode.*

In Appendice (A) è presentata una documentazione completa delle API SEDL.


SoftwareElementEngine
DescriptionEngine
(from SEDL)
<<static>> readSoftwareElement()
<<static>> writeSoftwareElement()
<<struct>>
SoftwareElement
name : string
author : string
version : string
<<enum>>
ElementType
Element_Generic
Element_Kernel
Element_Module
Element_Component
Element_WorkItem
<<struct>>
Source
name : string
version : string
fileLocator : string
<<enum>>
SourceLanguage
Source_Generic
Source_C
Source_Cpp
Source_F
Source_Java
<<enum>>
SourceFormat
Source_Data
Source_Interface
Source_Implementation
<<struct>>
Build
<<enum>>
SoftwareCodeType
Code_Meta
Code_Virtual
Code_Physical
<<struct>>
MetaSoftwareCode
<<struct>>
BuildConstraint
<<enum>>
PlatformCompiler
Compiler_Generic
Compiler_Gcc2
Compiler_Gcc3
<<enum>>
PlatformOs
Os_Generic
Os_Linux_RedHat_7
Os_Linux_RedHat
Os_Linux_Fedora
Os_Solaris
<<struct>>
SoftwareCode
name : string
<<struct>>
VirtualSoftwareCode
<<struct>>
Implementation
<<struct>>
Platform
<<struct>>
AbstractGraph
<<struct>>
PhysicalSoftwareCode
<<typedef>>
AbstractGraphArcTable
<<struct>>
AbstractNode
nodeID : int
softwareCode : string
<<enum>>
AbstractNodeRole
Role_Generic = 0
Role_Frontend
Role_Master
Role_Slave
<<enum>>
AbstractNodeType
Node_Meta = 1
Node_Virtual
Node_Physical
<<struct>>
ImplementationUnit
name : string
version : string
fileLocator : string
<<enum>>
PlatformMachine
Machine_Generic
Machine_x86
Machine_x86_64
Machine_sparc
<<enum>>
ImplementationLanguage
ImplementationGeneric = 0
Implementation_C
Implementation_Cpp
<<enum>>
ImplementationFormat
Implementation_Data = 0
Implementation_Executable
Implementation_Dynamic_Library

APPENDICE A: SEDL-Toolkit API Documentation

SEDL-Toolkit Documentazione dei namespace

Riferimenti per il namespace SEDL

Composti

struct AbstractGraph
struct AbstractNode
struct MetaNode
The Abstract Graph Meta Node.

struct VirtualNode
The Abstract Graph Virtual Node.

struct PhysicalNode
The Physical Graph Virtual Node.

struct **Build**
*The Software Element **Build** specification.*

struct ImplementationUnit
struct Implementation
*The **Implementation** of a Software Code.*

struct Platform
*The Software **Platform**.*

struct QoSConstrain
struct QoSVirtualConstrain
The QoS Graph Virtual Constrain.

struct QoSPhysicalConstrain
The QoS Graph Physical Constrain.

struct QoSGraph
struct **QoSNode**
The QoS Graph Node.

struct SoftwareCode
struct MetaSoftwareCode
The Meta Software Code.

struct VirtualSoftwareCode
The Virtual Software Code.

struct PhysicalSoftwareCode
The Physical Software Code.

struct SoftwareElement
The Software Element specification.

struct BuildConstraint
*The **Source BuildConstraint**.*

struct **Source**
*The **Source** Code description.*

class DescriptionEngine

Ridefinizioni di tipo (typedefs)

typedef map< int, AbstractNode * > AbstractNodeTable
The Abstract Graph Node Table.

typedef map< int, map< int, AbstractGraphArc * > > **AbstractGraphArcTable**
The Abstract Graph Arc Table.

typedef map< string, SoftwareCode * > SoftwareCodeTable
The Software Code Table.

typedef map< string, ImplementationUnit * > ImplementationUnitTable
*The **Implementation** Unit Table.*

typedef map< int, **QoSNode** * > **QoSNodeTable**
The QoS Graph Node Table.

typedef map< int, map< int, QoSGraphArc * > > **QoSGraphArcTable**
The QoS Graph Arc Table.

typedef map< string, **Source** * > **SourceTable**
*The Software Code **Source** Table.*

typedef map< string, Implementation * > ImplementationTable
*The Software Code **Implementation** Table.*

typedef vector< SourceBuildConstraint * > **BuildConstraintTable**
*The **Source BuildConstraint** Table.*

Tipi enumerati (enum)

enum AbstractNodeRole { Role_Generic = 0, Role_Frontend, Role_Master, Role_Slave }
The Abstract Graph Node Role.

enum AbstractNodeType { Node_Meta = 1, Node_Virtual, Node_Physical }
The Abstract Graph Node Type.

enum ImplementationFormat { Implementation_Data = 0, Implementation_Executable, Implementation_Dynamic_Library }
The Implementation Format.

enum ImplementationLanguage { ImplementationGeneric = 0, Implementation_C, Implementation_Cpp }
The Implementation Native Language.

enum PlatformCompiler { Compiler_Generic = 0, Compiler_Gcc2, Compiler_Gcc3 }
The Platform Compiler.

enum PlatformMachine { Machine_Generic = 0, Machine_x86, Machine_x86_64, Machine_sparc }

The Platform Architecure.

enum PlatformOs { Os_Generic = 0, Os_Linux_RedHat_7, Os_Linux_RedHat, Os_Linux_Fedora, Os_Solaris }

The Platform Operating System.

enum QoSConstrainType { Constrain_Virtual = 1, Constrain_Physical }

The QoS Constrain Type.

enum SoftwareCodeType { Code_Meta = 1, Code_Virtual, Code_Physical }

The Software Code type.

enum ElementType { Element_Generic = 0, Element_Kernel, Element_Module, Element_Component, Element_WorkItem }

The Software Element type.

enum SourceFormat { Source_Data = 0, Source_Interface, Source_Implementation }

The Source Format.

enum SourceLanguage { Source_Generic = 0, Source_C, Source_Cpp, Source_F, Source_Java }

The Source Language.

Funzioni

template<typename AbstractGraph> void **release** (**AbstractGraph** *)

Destroy.

template<typename AbstractNode> void **release** (**AbstractNode** *)

Destroy.

template<typename Build> void **release** (**Build** *)

Destroy.

template<typename Implementation> void **release** (**Implementation** *)

Destroy.

template<typename QoSVirtualConstrain> **QoSVirtualConstrain** * **create** (int multiplicity)

Create Virtual Constrain.

template<typename QoSPhysicalConstrain> **QoSPhysicalConstrain** * **create** (const string &resourceURI)

Create Physical Constrain.

template<typename QoSConstrain> void **release** (**QoSConstrain** *)

Destroy.

template<typename QoSGraph> void **release** (**QoSGraph** *)

Destroy.

template<typename QoSNode> **QoSNode** * **create** (int abs_nodeID, float minRequiredPower, float maxRequiredPower=0.0, const string &requiredPowerUnit="", **QoSConstrain** *constrain=0)

Create.

template<typename QoSNode> void **release** (**QoSNode** *)

Destroy.

template<typename SoftwareCode> void **release** (**SoftwareCode** *)

Destroy.

template<typename SoftwareElement> void **release** (**SoftwareElement** *)

Destroy.

Documentazione delle ridefinizioni di tipo (typedefs)

typedef map<int, map<int, AbstractGraphArc *> > SEDL::AbstractGraphArcTable

The Abstract Graph Arc Table.

typedef map<int, AbstractNode *> SEDL::AbstractNodeTable

The Abstract Graph Node Table.

typedef vector<SourceBuildConstraint *> SEDL::BuildConstraintTable

The **Source BuildConstraint** Table.

typedef map<string, Implementation *> SEDL::ImplementationTable

The Software Code **Implementation** Table.

typedef map<string, ImplementationUnit *> SEDL::ImplementationUnitTable

The **Implementation** Unit Table.

typedef map<int, map<int, QoSGraphArc *> > SEDL::QoSGraphArcTable

The QoS Graph Arc Table.

typedef map<int, QoSNode *> SEDL::QoSNodeTable

The QoS Graph Node Table.

typedef map<string, SoftwareCode *> SEDL::SoftwareCodeTable

The Software Code Table.

typedef map<string, Source *> SEDL::SourceTable

The Software Code **Source** Table.

Documentazione dei tipi enumerati

enum SEDL::AbstractNodeRole

The Abstract Graph Node Role.

Valori dei tipi enumerati:

Role_Generic
Role_Frontend
Role_Master
Role_Slave

enum SEDL::AbstractNodeType

The Abstract Graph Node Type.

Valori dei tipi enumerati:

Node_Meta
Node_Virtual
Node_Physical

enum SEDL::ElementType

The Software Element type.

Valori dei tipi enumerati:

Element_Generic Generic/Other element.

Element_Kernel
Element_Module
Element_Component
Element_WorkItem

enum SEDL::ImplementationFormat

The **Implementation** Format.

Valori dei tipi enumerati:

Implementation_Data Generic data/conf.

Implementation_Executable Executable.

Implementation_Dynamic_Library Dynamic library (i.e. DLL or shared object).

enum SEDL::ImplementationLanguage

The **Implementation** Native Language.

Valori dei tipi enumerati:

ImplementationGeneric Generic/Other.

Implementation_C C standard.

Implementation_Cpp C++.

enum SEDL::PlatformCompiler

The **Platform** Compiler.

Valori dei tipi enumerati:

Compiler_Generic Generic/Other.

Compiler_Gcc2 GNU C/C++ Compiler v2.xx.

Compiler_Gcc3 GNU C/C++ Compiler v3.xx.

enum SEDL::PlatformMachine

The **Platform** Architecure.

Valori dei tipi enumerati:

Machine_Generic Generic/Other.

Machine_x86 Intel/x86 or compatible.

Machine_x86_64 Intel/x86 64 bit.

Machine_sparc SUN/sparc/ultra or compatible.

enum SEDL::PlatformOs

The **Platform** Operating System.

Valori dei tipi enumerati:

Os_Generic Generic/Other.

Os_Linux_RedHat_7 Redhat Linux 7.x.

Os_Linux_RedHat Redhat Linux 8,9,x.

Os_Linux_Fedora Fedore Linux 1,2,3,x.

Os_Solaris SUN/Solaris 7,8,9,x.

enum SEDL::QoSConstrainType

The QoS Constrain Type.

Valori dei tipi enumerati:

Constrain_Virtual Virtual QoS Constrain.

Constrain_Physical Physical QoS Constrain.

enum SEDL::SoftwareCodeType

The Software Code type.

Valori dei tipi enumerati:

Code_Meta
Code_Virtual
Code_Physical

enum SEDL::SourceFormat

The **Source** Format.

Valori dei tipi enumerati:

Source_Data Generic source data.

Source_Interface Interface (API).

Source_Implementation **Implementation**.

enum SEDL::SourceLanguage

The **Source** Language.

Valori dei tipi enumerati:

Source_Generic Generic/Other.

Source_C C standard.

Source_Cpp C++.

Source_F Fortran.

Source_Java Java.

Documentazione delle funzioni

template<typename QoSNode> QoSNode * SEDL::create< QoSNode > (int *abs_nodeID*, float *minRequiredPower*, float *maxRequiredPower* = `0.0`, const string & *requiredPowerUnit* = `""`, QoSConstrain * *constrain* = `0`)

Create.

template<typename QoSPhysicalConstrain> QoSPhysicalConstrain * SEDL::create< QoSPhysicalConstrain > (const string & *resourceURI*)

Create Physical Constrain.

template<typename QoSVirtualConstrain> QoSVirtualConstrain * SEDL::create< QoSVirtualConstrain > (int *multiplicity*)

Create Virtual Constrain.

template<typename SoftwareElement> void SEDL::release< SoftwareElement > (SoftwareElement *)

Destroy.

template<typename SoftwareCode> void SEDL::release< SoftwareCode > (SoftwareCode *)

Destroy.

template<typename QoSNode> void SEDL::release< QoSNode > (QoSNode *)

Destroy.

template<typename QoSGraph> void SEDL::release< QoSGraph > (QoSGraph *)

Destroy.

template<typename QoSConstrain> void SEDL::release< QoSConstrain > (QoSConstrain *)

Destroy.

template<typename Implementation> void SEDL::release< Implementation > (Implementation *)

Destroy.

template<typename Build> void SEDL::release< Build > (Build *)

Destroy.

template<typename AbstractNode> void SEDL::release< AbstractNode > (AbstractNode *)

Destroy.

template<typename AbstractGraph> void SEDL::release< AbstractGraph > (AbstractGraph *)

Destroy.

Riferimenti per il namespace std

Riferimenti per il namespace xmlns_sedl

SEDL-Toolkit Documentazione delle classi

Riferimenti per la struct SEDL::AbstractGraph

`#include <AbstractGraph.h>`

Attributi pubblici

AbstractNodeTable nodes
The graph nodes.

AbstractGraphArcTable arcs
The graph arcs (not yet implemented).

Descrizione Dettagliata

The Abstract Graph Contains Abstract Nodes/Arcs sets.

Documentazione dei dati membri

AbstractGraphArcTable SEDL::AbstractGraph::arcs

The graph arcs (not yet implemented).

AbstractNodeTable SEDL::AbstractGraph::nodes

The graph nodes.

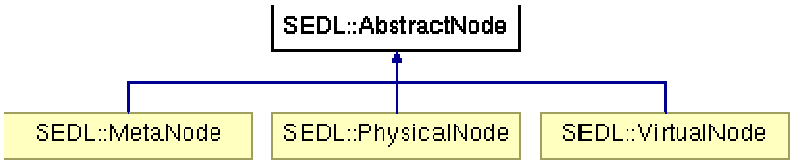
La documentazione per questa struct è stata generata a partire dal seguente file:

- **AbstractGraph.h**

Riferimenti per la struct SEDL::AbstractNode

`#include <AbstractNode.h>`

Diagramma delle classi per SEDL::AbstractNode



Attributi pubblici

int nodeID

The node unique identifier.

AbstractNodeRole role

The node role (i.e. Master/Slave).

AbstractNodeType type

The node type (i.e. Meta/Virtual/Physical).

string softwareCode

Descrizione Dettagliata

The Abstract Graph Node This is base class for Abstract Graph Meta/Virtual/Physical Node

---

Documentazione dei dati membri

int SEDL::AbstractNode::nodeID

The node unique identifier.

AbstractNodeRole SEDL::AbstractNode::role

The node role (i.e. Master/Slave).

string SEDL::AbstractNode::softwareCode

The related Software Code reference name

Vedi anche:

SoftwareCode

AbstractNodeType SEDL::AbstractNode::type

The node type (i.e. Meta/Virtual/Physical).

La documentazione per questa struct è stata generata a partire dal seguente file:

- **AbstractNode.h**

Riferimenti per la struct SEDL::Build

The Software Element **Build** specification.

```
#include <Build.h>
```

Attributi pubblici

SoftwareCodeTable softwareCode

The build software codes.

AbstractGraph * abstractGraph

The build abstract graph (if any).

Descrizione Dettagliata

The Software Element **Build** specification.

Documentazione dei dati membri

AbstractGraph* SEDL::Build::abstractGraph

The build abstract graph (if any).

SoftwareCodeTable SEDL::Build::softwareCode

The build software codes.

La documentazione per questa struct è stata generata a partire dal seguente file:

- **Build.h**

Riferimenti per la struct SEDL::BuildConstraint

The **Source BuildConstraint**.

```
#include <Source.h>
```

Attributi pubblici

Platform platform
The build-constraint software platform.

Descrizione Dettagliata

The **Source BuildConstraint**.

Documentazione dei dati membri

Platform SEDL::BuildConstraint::platform

The build-constraint software platform.

La documentazione per questa struct è stata generata a partire dal seguente file:

- **Source.h**

Riferimenti per la classe SEDL::DescriptionEngine

`#include <DescriptionEngine.h>`

Membri pubblici statici

static struct **SoftwareElement** * **readSoftwareElement** (const string &fileLocator) throw (string)
static struct **QoSGraph** * **getQoSGraph** (const string &text) throw (string)
static string **putQoSGraph** (struct **QoSGraph** *qosGraph) throw (string)

Documentazione delle funzioni membro

QoSGraph * DescriptionEngine::getQoSGraph (const string & *text*) throw (string) `[static]`

string DescriptionEngine::putQoSGraph (struct QoSGraph * *qosGraph*) throw (string) `[static]`

SoftwareElement * DescriptionEngine::readSoftwareElement (const string & *fileLocator*) throw (string) `[static]`

La documentazione per questa classe è stata generata a partire dai seguenti file:

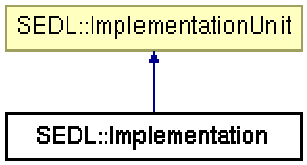
- **DescriptionEngine.h**
- **DescriptionEngine.cpp**

Riferimenti per la struct SEDL::Implementation

The **Implementation** of a Software Code.

`#include <Implementation.h>`

Diagramma delle classi per SEDL::Implementation



Attributi pubblici

ImplementationLanguage language

The implementation language.

Platform platform

The implementation software platform.

ImplementationUnitTable dependency

The implementation dependency (i.e. support-data,support-library,conf-file).

Descrizione Dettagliata

The **Implementation** of a Software Code.

Documentazione dei dati membri

ImplementationUnitTable SEDL::Implementation::dependency

The implementation dependency (i.e. support-data,support-library,conf-file).

ImplementationLanguage SEDL::Implementation::language

The implementation language.

Platform SEDL::Implementation::platform

The implementation software platform.

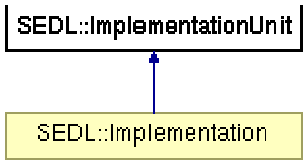
La documentazione per questa struct è stata generata a partire dal seguente file:

- **Implementation.h**

Riferimenti per la struct SEDL::ImplementationUnit

`#include <Implementation.h>`

Diagramma delle classi per SEDL::ImplementationUnit



Attributi pubblici

string **name**
the unit name

string **version**
the unit version (if any)

string fileLocator
the unit file path

ImplementationFormat format
the unit file format

Descrizione Dettagliata

The **Implementation** Unit This is an implementation file description

Documentazione dei dati membri

string SEDL::ImplementationUnit::fileLocator

the unit file path

ImplementationFormat SEDL::ImplementationUnit::format

the unit file format

string SEDL::ImplementationUnit::name

the unit name

string SEDL::ImplementationUnit::version

the unit version (if any)

La documentazione per questa struct è stata generata a partire dal seguente file:

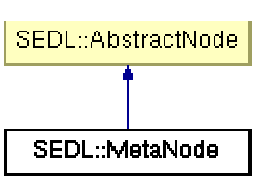
- **Implementation.h**

Riferimenti per la struct SEDL::MetaNode

The Abstract Graph Meta Node.

`#include <AbstractNode.h>`

Diagramma delle classi per SEDL::MetaNode



Attributi pubblici

const MetaSoftwareCode * metaSoftwareCode
int multiplicityMin

The node minimun multiplicity.

int multiplicityMax
The node maximum multiplicity (0=unbound).

Descrizione Dettagliata

The Abstract Graph Meta Node.

Documentazione dei dati membri

const MetaSoftwareCode* SEDL::MetaNode::metaSoftwareCode

The Meta Software Code reference (may be null)

Vedi anche:

MetaSoftwareCode

int SEDL::MetaNode::multiplicityMax

The node maximum multiplicity (0=unbound).

int SEDL::MetaNode::multiplicityMin

The node minimun multiplicity.

La documentazione per questa struct è stata generata a partire dal seguente file:

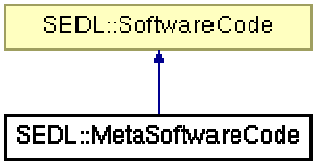
- **AbstractNode.h**

Riferimenti per la struct SEDL::MetaSoftwareCode

The Meta Software Code.

```
#include <SoftwareCode.h>
```

Diagramma delle classi per SEDL::MetaSoftwareCode



Attributi pubblici

SourceTable source
The software code sources (not yet implemented).

ImplementationTable implementation
The software code implementations.

Descrizione Dettagliata

The Meta Software Code.

Documentazione dei dati membri

ImplementationTable SEDL::MetaSoftwareCode::implementation

The software code implementations.

SourceTable SEDL::MetaSoftwareCode::source

The software code sources (not yet implemented).

La documentazione per questa struct è stata generata a partire dal seguente file:

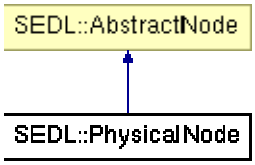
- **SoftwareCode.h**

Riferimenti per la struct SEDL::PhysicalNode

The Physical Graph Virtual Node.

```
#include <AbstractNode.h>
```

Diagramma delle classi per SEDL::PhysicalNode



Attributi pubblici

const PhysicalSoftwareCode * physicalSoftwareCode
string resourceURI
The node mapping resource URI or description URL.

Descrizione Dettagliata

The Physical Graph Virtual Node.

Documentazione dei dati membri

const PhysicalSoftwareCode* SEDL::PhysicalNode::physicalSoftwareCode

The Physical Software Code reference (may be null)

Vedi anche:

PhysicalSoftwareCode

string SEDL::PhysicalNode::resourceURI

The node mapping resource URI or description URL.

La documentazione per questa struct è stata generata a partire dal seguente file:

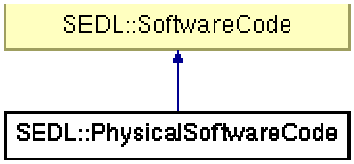
- **AbstractNode.h**

Riferimenti per la struct SEDL::PhysicalSoftwareCode

The Physical Software Code.

`#include <SoftwareCode.h>`

Diagramma delle classi per SEDL::PhysicalSoftwareCode



Attributi pubblici

Implementation * implementation

The software code implementation.

Descrizione Dettagliata

The Physical Software Code.

Documentazione dei dati membri

Implementation* SEDL::PhysicalSoftwareCode::implementation

The software code implementation.

La documentazione per questa struct è stata generata a partire dal seguente file:

- **SoftwareCode.h**

Riferimenti per la struct SEDL::Platform

The Software **Platform**.

`#include <Platform.h>`

Attributi pubblici

PlatformCompiler compiler
The software platform compiler.

PlatformMachine machine
The software platform architecture.

PlatformOs os
The software platform operating system.

Descrizione Dettagliata

The Software **Platform**.

Documentazione dei dati membri

PlatformCompiler SEDL::Platform::compiler

The software platform compiler.

PlatformMachine SEDL::Platform::machine

The software platform architecture.

PlatformOs SEDL::Platform::os

The software platform operating system.

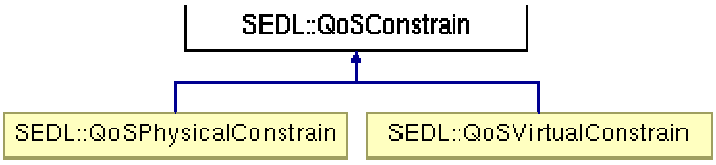
La documentazione per questa struct è stata generata a partire dal seguente file:

- **Platform.h**

Riferimenti per la struct SEDL::QoSConstrain

`#include <QoSConstrain.h>`

Diagramma delle classi per SEDL::QoSConstrain



Attributi pubblici

QoSConstrainType type
The constrain type.

Descrizione Dettagliata

The QoS Graph Constrain This is base class for QoS Graph Virtual/Physical Constrain

---

Documentazione dei dati membri

QoSConstrainType SEDL::QoSConstrain::type

The constrain type.

La documentazione per questa struct è stata generata a partire dal seguente file:

- **QoSConstrain.h**

Riferimenti per la struct SEDL::QoSGraph

`#include <QoSGraph.h>`

Attributi pubblici

QoSNodeTable nodes

The graph nodes.

QoSGraphArcTable arcs
The graph arcs (not yet implemented).

Descrizione Dettagliata

The QoS Graph Contains QoS Nodes/Arcs sets.

Documentazione dei dati membri

QoSGraphArcTable SEDL::QoSGraph::arcs

The graph arcs (not yet implemented).

QoSNodeTable SEDL::QoSGraph::nodes

The graph nodes.

La documentazione per questa struct è stata generata a partire dal seguente file:

- **QoSGraph.h**

Riferimenti per la struct SEDL::QoSNode

The QoS Graph Node.
`#include <QoSNode.h>`

Attributi pubblici

int abs_nodeID
float minRequiredPower
The minimum required node power.

float maxRequiredPower
The maximum required node power (0=unbound).

string requiredPowerUnit
The required node power unit (if any).

QoSConstrain * constrain
The node constrain (if any).

Descrizione Dettagliata

The QoS Graph Node.

Documentazione dei dati membri

int SEDL::QoSNode::abs_nodeID

The related Abstract Node id

Vedi anche:

AbstractNode

QoSConstrain* SEDL::QoSNode::constrain

The node constrain (if any).

float SEDL::QoSNode::maxRequiredPower

The maximum required node power (0=unbound).

float SEDL::QoSNode::minRequiredPower

The minimum required node power.

string SEDL::QoSNode::requiredPowerUnit

The required node power unit (if any).

La documentazione per questa struct è stata generata a partire dal seguente file:

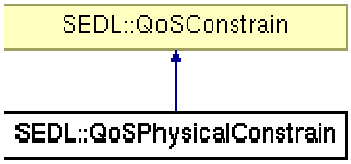
- **QoSNode.h**

Riferimenti per la struct SEDL::QoSPhysicalConstrain

The QoS Graph Physical Constrain.

`#include <QoSConstrain.h>`

Diagramma delle classi per SEDL::QoSPhysicalConstrain



Attributi pubblici

string resourceURI

The constrain resource URI or description URL.

Descrizione Dettagliata

The QoS Graph Physical Constrain.

Documentazione dei dati membri

string SEDL::QoSPhysicalConstrain::resourceURI

The constrain resource URI or description URL.

La documentazione per questa struct è stata generata a partire dal seguente file:

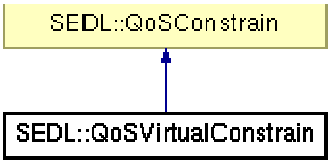
- **QoSConstrain.h**

Riferimenti per la struct SEDL::QoSVirtualConstrain

The QoS Graph Virtual Constrain.

`#include <QoSConstrain.h>`

Diagramma delle classi per SEDL::QoSVirtualConstrain



Attributi pubblici

int multiplicity
The constrain multiplicity.

Descrizione Dettagliata

The QoS Graph Virtual Constrain.

Documentazione dei dati membri

int SEDL::QoSVirtualConstrain::multiplicity

The constrain multiplicity.

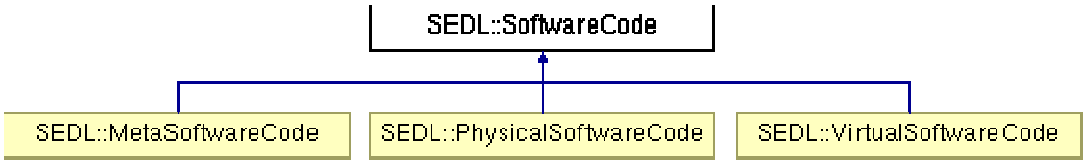
La documentazione per questa struct è stata generata a partire dal seguente file:

- **QoSConstrain.h**

Riferimenti per la struct SEDL::SoftwareCode

`#include <SoftwareCode.h>`

Diagramma delle classi per SEDL::SoftwareCode



Attributi pubblici

string **name**
The software code name.

SoftwareCodeType type
The software code type.

Descrizione Dettagliata

The Software Code description. This is base class for Meta/Virtual/Physical Software Code.

Documentazione dei dati membri

string SEDL::SoftwareCode::name

The software code name.

SoftwareCodeType SEDL::SoftwareCode::type

The software code type.

La documentazione per questa struct è stata generata a partire dal seguente file:

- **SoftwareCode.h**

Riferimenti per la struct SEDL::SoftwareElement

The Software Element specification.

`#include <SoftwareElement.h>`

Attributi pubblici

ElementType type

The software element type.

string **name**

The software element name.

string **author**

The software element author (if any).

string **version**

The software element version (if any).

Build * build

The software element build.

Behaviour * **behaviour**

The software element behaviour (not yet implemented).

Semantic * **semantic**

The software element semantic (not yet implemented).

Descrizione Dettagliata

The Software Element specification.

Documentazione dei dati membri

string SEDL::SoftwareElement::author

The software element author (if any).

Behaviour* SEDL::SoftwareElement::behaviour

The software element behaviour (not yet implemented).

Build* SEDL::SoftwareElement::build

The software element build.

string SEDL::SoftwareElement::name

The software element name.

Semantic* SEDL::SoftwareElement::semantic

The software element semantic (not yet implemented).

ElementType SEDL::SoftwareElement::type

The software element type.

string SEDL::SoftwareElement::version

The software element version (if any).

La documentazione per questa struct è stata generata a partire dal seguente file:

- **SoftwareElement.h**

Riferimenti per la struct SEDL::Source

The **Source** Code description.

```
#include <Source.h>
```

Attributi pubblici

string **name**
the source name

string **version**
the source version (if any)

string fileLocator
the source file path

SourceFormat format
the source file format

SourceLanguage language
the source lanugage

BuildConstraintTable constraints
The source build-constraints.

Descrizione Dettagliata

The **Source** Code description.

Documentazione dei dati membri

BuildConstraintTable SEDL::Source::constraints

The source build-constraints.

string SEDL::Source::fileLocator

the source file path

SourceFormat SEDL::Source::format

the source file format

SourceLanguage SEDL::Source::language

the source lanugage

string SEDL::Source::name

the source name

string SEDL::Source::version

the source version (if any)

La documentazione per questa struct è stata generata a partire dal seguente file:

- **Source.h**

Riferimenti per la classe toUpper

Membri pubblici

toUpper (const string &s)

Documentazione dei costruttori e dei distruttori

toUpper::toUpper (const string & *s*) `[inline]`

La documentazione per questa classe è stata generata a partire dal seguente file:

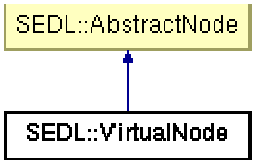
- **DescriptionAdapter.cpp**

Riferimenti per la struct SEDL::VirtualNode

The Abstract Graph Virtual Node.

`#include <AbstractNode.h>`

Diagramma delle classi per SEDL::VirtualNode



Attributi pubblici

const VirtualSoftwareCode * virtualSoftwareCode

int multiplicityMin

The node minimun multiplicity.

int multiplicityMax

The node maximum multiplicity (0=unbound).

Descrizione Dettagliata

The Abstract Graph Virtual Node.

Documentazione dei dati membri

int SEDL::VirtualNode::multiplicityMax

The node maximum multiplicity (0=unbound).

int SEDL::VirtualNode::multiplicityMin

The node minimun multiplicity.

const VirtualSoftwareCode* SEDL::VirtualNode::virtualSoftwareCode

The Virtual Software Code reference (may be null)

Vedi anche:

VirtualSoftwareCode

La documentazione per questa struct è stata generata a partire dal seguente file:

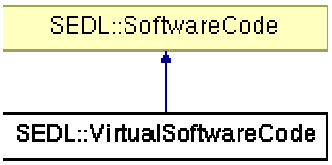
- **AbstractNode.h**

Riferimenti per la struct SEDL::VirtualSoftwareCode

The Virtual Software Code.

`#include <SoftwareCode.h>`

Diagramma delle classi per SEDL::VirtualSoftwareCode



Attributi pubblici

ImplementationTable implementation

The software code implementations.

Descrizione Dettagliata

The Virtual Software Code.

Documentazione dei dati membri

ImplementationTable SEDL::VirtualSoftwareCode::implementation

The software code implementations.

La documentazione per questa struct è stata generata a partire dal seguente file:

- **SoftwareCode.h**

SEDL-Toolkit Documentazione dei file

Riferimenti per il file AbstractGraph.cpp

```
#include <AbstractGraph.h>
#include <AbstractNode.h>
```

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date

Wed Nov 9 2005

Revision

1.1.0

Implementation of Abstract Graph.

Riferimenti per il file AbstractGraph.h

```
#include <map>
```

Namespace

namespace **SEDL**
namespace **std**

Composti

struct SEDL::AbstractGraph

Ridefinizioni di tipo (typedefs)

typedef map< int, AbstractNode * > **SEDL::AbstractNodeTable**

The Abstract Graph Node Table.

typedef map< int, map< int, AbstractGraphArc * > > **SEDL::AbstractGraphArcTable**

The Abstract Graph Arc Table.

Funzioni

template<typename AbstractGraph> void **SEDL::release** (AbstractGraph *)
Destroy.

Descrizione Dettagliata

Autore:
Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date
Wed Nov 9 2005

Revision
1.1.0

Definition of Abstract Graph.

Riferimenti per il file AbstractNode.cpp

```
#include <AbstractNode.h>
```

Descrizione Dettagliata

Autore:
Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date
Wed Nov 9 2005

Revision
1.1.0

Implementation of Abstract Node.

Riferimenti per il file AbstractNode.h

```
#include <string>
```

Namespace

namespace **SEDL**

Composti

struct SEDL::AbstractNode
struct SEDL::MetaNode
The Abstract Graph Meta Node.

struct SEDL::VirtualNode
The Abstract Graph Virtual Node.

struct SEDL::PhysicalNode
The Physical Graph Virtual Node.

Tipi enumerati (enum)

enum SEDL::AbstractNodeRole { SEDL::Role_Generic = 0, SEDL::Role_Frontend, SEDL::Role_Master, SEDL::Role_Slave }
The Abstract Graph Node Role.

enum SEDL::AbstractNodeType { SEDL::Node_Meta = 1, SEDL::Node_Virtual, SEDL::Node_Physical }
The Abstract Graph Node Type.

Funzioni

template<typename AbstractNode> void **SEDL::release** (AbstractNode *)
Destroy.

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date

Wed Nov 9 2005

Revision

1.1.0

Definition of Abstract Node.

Riferimenti per il file Build.cpp

```
#include <Build.h>
#include <SoftwareCode.h>
#include <AbstractGraph.h>
```

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date

Wed Nov 9 2005

Revision

1.1.0

Implementation of Build.

Riferimenti per il file Build.h

```
#include <map>
#include <string>
```

Namespace

namespace **SEDL**

Composti

struct SEDL::Build
*The Software Element **Build** specification.*

Ridefinizioni di tipo (typedefs)

typedef map< string, SoftwareCode * > **SEDL::SoftwareCodeTable**
The Software Code Table.

Funzioni

template<typename Build> void **SEDL::release** (Build *)
Destroy.

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date

Wed Nov 9 2005

Revision

1.1.0

Definition of Build.

Riferimenti per il file Description.cpp

```
#include <Description.h>
```

Namespace

namespace **xmlns_sedl**

Funzioni

template<> QoSGraph::node * **xmlns_sedl::create< QoSGraph::node >** (int abs_nodeID, float minRequiredPower, float maxRequiredPower, const string &RequiredPowerUnit)

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date

Wed Nov 9 2005

Revision

1.1.0

Implementation of XML-SEDL namespace.

Documentazione delle funzioni

template<> QoSGraph::node* xmlns_sedl::create< QoSGraph::node > (int *abs_nodeID*, float *minRequiredPower*, float *maxRequiredPower*, const string & *RequiredPowerUnit*)

Riferimenti per il file DescriptionAdapter.cpp

```
#include "Description.h"
#include "DescriptionAdapter.h"
#include <SoftwareElement.h>
#include <Build.h>
#include <SoftwareCode.h>
#include <Implementation.h>
#include <AbstractGraph.h>
#include <AbstractNode.h>
```

```
#include <QoSGraph.h>
#include <QoSNode.h>
#include <QoSConstrain.h>
#include <ctype.h>
```

Composti

class toUpper

Funzioni

template<> QoSConstrain * DescriptionAdapter::get< SEDL::QoSConstrain, xmlns_sedl::physicalConstrain > (const xmlns_sedl::physicalConstrain *xml)
template<> QoSConstrain * DescriptionAdapter::get< SEDL::QoSConstrain, xmlns_sedl::virtualConstrain > (const xmlns_sedl::virtualConstrain *xml)
template<> QoSNode * DescriptionAdapter::get< SEDL::QoSNode, xmlns_sedl::QoSGraph::node > (const xmlns_sedl::QoSGraph::node *xml)
template<> QoSGraph * DescriptionAdapter::get< SEDL::QoSGraph, xmlns_sedl::QoSGraph > (const xmlns_sedl::QoSGraph *xml)
template<> AbstractNode * DescriptionAdapter::get< SEDL::AbstractNode, xmlns_sedl::virtualNode > (const xmlns_sedl::virtualNode *xml)
template<> AbstractGraph * DescriptionAdapter::get< SEDL::AbstractGraph, xmlns_sedl::abstractGraph > (const xmlns_sedl::abstractGraph *xml)
template<> ImplementationUnit * DescriptionAdapter::get< SEDL::ImplementationUnit, xmlns_sedl::dependency > (const xmlns_sedl::dependency *xml)
template<> Implementation * DescriptionAdapter::get< SEDL::Implementation, xmlns_sedl::implementation > (const xmlns_sedl::implementation *xml)
template<> SoftwareCode * DescriptionAdapter::get< SEDL::SoftwareCode, xmlns_sedl::virtualSoftwareCode > (const xmlns_sedl::virtualSoftwareCode *xml)
template<> Build * DescriptionAdapter::get< SEDL::Build, xmlns_sedl::build > (const xmlns_sedl::build *xml)
template<> SoftwareElement * DescriptionAdapter::get< SEDL::SoftwareElement, xmlns_sedl::softwareElement > (const xmlns_sedl::softwareElement *xml)

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Wed Nov 9 2005

Revision

1.1.0

Implementation of Description Adaptation layer.

Documentazione delle funzioni

template<> AbstractGraph* DescriptionAdapter::get< SEDL::AbstractGraph, xmlns_sedl::abstractGraph > (const xmlns_sedl::abstractGraph * *xml*)

template<> AbstractNode* DescriptionAdapter::get< SEDL::AbstractNode, xmlns_sedl::virtualNode > (const xmlns_sedl::virtualNode * *xml*)

template<> Build* DescriptionAdapter::get< SEDL::Build, xmlns_sedl::build > (const xmlns_sedl::build * *xml*)

template<> Implementation* DescriptionAdapter::get< SEDL::Implementation, xmlns_sedl::implementation > (const xmlns_sedl::implementation * *xml*)

template<> ImplementationUnit* DescriptionAdapter::get< SEDL::ImplementationUnit, xmlns_sedl::dependency > (const xmlns_sedl::dependency * *xml*)

template<> QoSConstrain* DescriptionAdapter::get< SEDL::QoSConstrain, xmlns_sedl::physicalConstrain > (const xmlns_sedl::physicalConstrain * *xml*)

template<> QoSConstrain* DescriptionAdapter::get< SEDL::QoSConstrain, xmlns_sedl::virtualConstrain > (const xmlns_sedl::virtualConstrain * *xml*)

template<> QoSGraph* DescriptionAdapter::get< SEDL::QoSGraph, xmlns_sedl::QoSGraph > (const xmlns_sedl::QoSGraph * *xml*)

template<> QoSNode* DescriptionAdapter::get< SEDL::QoSNode, xmlns_sedl::QoSGraph::node > (const xmlns_sedl::QoSGraph::node * *xml*)

template<> SoftwareCode* DescriptionAdapter::get< SEDL::SoftwareCode, xmlns_sedl::virtualSoftwareCode > (const xmlns_sedl::virtualSoftwareCode * *xml*)

template<> SoftwareElement* DescriptionAdapter::get< SEDL::SoftwareElement, xmlns_sedl::softwareElement > (const xmlns_sedl::softwareElement * *xml*)

Riferimenti per il file DescriptionBuilder.cpp

```
#include "Description.h"
#include "DescriptionBuilder.h"
#include <QoSGraph.h>
#include <QoSNode.h>
#include <QoSConstrain.h>
```

Funzioni

template<> xmlns_sedl::QoSGraph * DescriptionBuilder::get< xmlns_sedl::QoSGraph, SEDL::QoSGraph > (QoSGraph *qg)

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date

Wed Nov 9 2005

Revision

1.1.0

Implementation of Description Presentation.

Documentazione delle funzioni

template<> xmlns_sedl::QoSGraph* DescriptionBuilder::get< xmlns_sedl::QoSGraph, SEDL::QoSGraph > (QoSGraph * *qg*)

Riferimenti per il file DescriptionEngine.cpp

```
#include <AL.h>
#include <AL_XMLElement.h>
#include <AL_XMLParser.h>
#include "Description.h"
#include "DescriptionEngine.h"
#include "DescriptionAdapter.h"
#include "DescriptionBuilder.h"
#include <SoftwareElement.h>
#include <Build.h>
#include <SoftwareCode.h>
#include <Implementation.h>
#include <AbstractGraph.h>
#include <AbstractNode.h>
#include <QoSGraph.h>
#include <QoSNode.h>
#include <QoSConstrain.h>
```

Funzioni

static void **print** (AL_XMLElement *tree, string tabs="")

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date

Wed Nov 9 2005

Revision

1.0.2

Implementation of Description Engine.

Documentazione delle funzioni

static void print (AL_XMLElement * *tree*, string *tabs* = "") [static]

Riferimenti per il file DescriptionEngine.h

```
#include <string>
```

Namespace

namespace **SEDL**

Composti

class SEDL::DescriptionEngine

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date

Tue Jun 28 2005

Revision

1.0.1

Definition of Description Engine.

Riferimenti per il file Implementation.cpp

```
#include <Implementation.h>
```

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date

Wed Nov 9 2005

Revision

1.1.0

Implementation of Implementation.

Riferimenti per il file Implementation.h

```
#include <map>
#include <string>
#include <Platform.h>
```

Namespace

namespace **SEDL**

Composti

struct SEDL::ImplementationUnit
struct SEDL::Implementation
*The **Implementation** of a Software Code.*

Ridefinizioni di tipo (typedefs)

typedef map< string, ImplementationUnit * > **SEDL::ImplementationUnitTable**
*The **Implementation** Unit Table.*

Tipi enumerati (enum)

enum SEDL::ImplementationFormat { SEDL::Implementation_Data = 0, SEDL::Implementation_Executable, SEDL::Implementation_Dynamic_Library }
The Implementation Format.

enum SEDL::ImplementationLanguage { SEDL::ImplementationGeneric = 0, SEDL::Implementation_C, SEDL::Implementation_Cpp }

The Implementation Native Language.

Funzioni

template<typename Implementation> void **SEDL::release** (Implementation *)

Destroy.

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Wed Nov 9 2005

Revision

1.1.0

Definition of Implementation.

Riferimenti per il file Platform.h

```
#include <string>
```

Namespace

namespace **SEDL**

Composti

struct SEDL::Platform

*The Software **Platform**.*

Tipi enumerati (enum)

enum SEDL::PlatformCompiler { SEDL::Compiler_Generic = 0, SEDL::Compiler_Gcc2, SEDL::Compiler_Gcc3 }

The Platform Compiler.

enum SEDL::PlatformMachine { SEDL::Machine_Generic = 0, SEDL::Machine_x86, SEDL::Machine_x86_64, SEDL::Machine_sparc }

The Platform Architecure.

enum SEDL::PlatformOs { SEDL::Os_Generic = 0, SEDL::Os_Linux_RedHat_7, SEDL::Os_Linux_RedHat, SEDL::Os_Linux_Fedora, SEDL::Os_Solaris }

The Platform Operating System.

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Wed Nov 9 2005

Revision

1.1.0

Definition of Software Platform.

Riferimenti per il file QoSConstrain.cpp

```
#include <QoSConstrain.h>
```

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Wed Nov 9 2005

Revision

1.1.1

Implementation of Quality-Of-Service (QoS) Graph Constrain.

Riferimenti per il file QoSConstrain.h

```
#include <string>
```

Namespace

namespace **SEDL**

Composti

struct SEDL::QoSConstrain
struct SEDL::QoSVirtualConstrain

The QoS Graph Virtual Constrain.

struct SEDL::QoSPhysicalConstrain

The QoS Graph Physical Constrain.

Tipi enumerati (enum)

enum SEDL::QoSConstrainType { SEDL::Constrain_Virtual = 1, SEDL::Constrain_Physical }

The QoS Constrain Type.

Funzioni

template<typename QoSVirtualConstrain> QoSVirtualConstrain * **SEDL::create** (int multiplicity)

Create Virtual Constrain.

template<typename QoSPhysicalConstrain> QoSPhysicalConstrain * **SEDL::create** (const string &resourceURI)

Create Physical Constrain.

template<typename QoSConstrain> void **SEDL::release** (QoSConstrain *)

Destroy.

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Wed Nov 9 2005

Revision

1.1.1

Definition of Quality-Of-Service (QoS) Graph Constrain.

Riferimenti per il file QoSGraph.cpp

```
#include <QoSGraph.h>
#include <QoSNode.h>
```

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Wed Nov 9 2005

Revision

1.1.1

Implementation of Quality-Of-Service (QoS) Graph.

Riferimenti per il file QoSGraph.h

`#include <map>`

Namespace

namespace **SEDL**

Composti

struct SEDL::QoSGraph

Ridefinizioni di tipo (typedefs)

typedef map< int, QoSNode * > **SEDL::QoSNodeTable**
The QoS Graph Node Table.

typedef map< int, map< int, QoSGraphArc * > > **SEDL::QoSGraphArcTable**
The QoS Graph Arc Table.

Funzioni

template<typename QoSGraph> void **SEDL::release** (QoSGraph *)
Destroy.

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Wed Nov 9 2005

Revision

1.1.1

Definition of Quality-Of-Service (QoS) Graph.

Riferimenti per il file QoSNode.cpp

```
#include <QoSNode.h>
#include <QoSConstrain.h>
```

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date

Wed Nov 9 2005

Revision

1.1.1

Implementation of Quality-Of-Service (QoS) Graph Node.

Riferimenti per il file QoSNode.h

```
#include <string>
```

Namespace

namespace **SEDL**

Composti

struct SEDL::QoSNode
The QoS Graph Node.

Funzioni

template<typename QoSNode> QoSNode * **SEDL::create** (int abs_nodeID, float minRequiredPower, float maxRequiredPower=0.0, const string &requiredPowerUnit="", QoSConstrain *constrain=0)
Create.

template<typename QoSNode> void **SEDL::release** (QoSNode *)
Destroy.

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date

Wed Nov 9 2005

Revision

1.1.1

Definition of Quality-Of-Service (QoS) Graph Node.

Riferimenti per il file SoftwareCode.cpp

```
#include <SoftwareCode.h>
#include <Implementation.h>
```

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date

Wed Nov 9 2005

Revision

1.1.0

Implementation of ...

Riferimenti per il file SoftwareCode.h

```
#include <map>
```

```
#include <string>
```

Namespace

namespace **SEDL**

Composti

struct SEDL::SoftwareCode
struct SEDL::MetaSoftwareCode
The Meta Software Code.

struct SEDL::VirtualSoftwareCode
The Virtual Software Code.

struct SEDL::PhysicalSoftwareCode
The Physical Software Code.

Ridefinizioni di tipo (typedefs)

typedef map< string, Source * > **SEDL::SourceTable**
*The Software Code **Source** Table.*

typedef map< string, Implementation * > **SEDL::ImplementationTable**
*The Software Code **Implementation** Table.*

Tipi enumerati (enum)

enum SEDL::SoftwareCodeType { SEDL::Code_Meta = 1, SEDL::Code_Virtual, SEDL::Code_Physical }
The Software Code type.

Funzioni

template<typename SoftwareCode> void **SEDL::release** (SoftwareCode *)
Destroy.

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date

Wed Nov 9 2005

Revision

1.1.0

Definition of Software Code.

Riferimenti per il file SoftwareElement.cpp

```
#include <SoftwareElement.h>
#include <Build.h>
```

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date

Wed Nov 9 2005

Revision

1.1.0

Implementation of Software Element.

Riferimenti per il file SoftwareElement.h

```
#include <string>
```

Namespace

namespace **SEDL**

Composti

struct SEDL::SoftwareElement
The Software Element specification.

Tipi enumerati (enum)

enum SEDL::ElementType { SEDL::Element_Generic = 0, SEDL::Element_Kernel, SEDL::Element_Module, SEDL::Element_Component, SEDL::Element_WorkItem }
The Software Element type.

Funzioni

template<typename SoftwareElement> void **SEDL::release** (SoftwareElement *)
Destroy.

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Wed Nov 9 2005

Revision

1.1.0

Definition of Software Element.

Riferimenti per il file Source.h

```
#include <string>
#include <vector>
#include <Platform.h>
```

Namespace

namespace **SEDL**

Composti

struct SEDL::BuildConstraint
*The **Source BuildConstraint**.*

struct SEDL::Source
*The **Source** Code description.*

Ridefinizioni di tipo (typedefs)

typedef vector< SourceBuildConstraint * > **SEDL::BuildConstraintTable**
*The **Source BuildConstraint** Table.*

Tipi enumerati (enum)

enum SEDL::SourceFormat { SEDL::Source_Data = 0, SEDL::Source_Interface, SEDL::Source_Implementation }
The Source Format.

enum SEDL::SourceLanguage { SEDL::Source_Generic = 0, SEDL::Source_C, SEDL::Source_Cpp, SEDL::Source_F, SEDL::Source_Java }
The Source Language.

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Wed Nov 9 2005

Revision

1.1.0

Definition of Source Code.

APPENDICE B: LegaSIE-Toolkit API Documentation

LegaSIE-Toolkit Documentazione dei namespace

Riferimenti per il namespace LegaSIE

Composti

class AnalyseCpp
class AnalyseEngine
class AnalyseObj
struct Compound
*The **Compound** element.*

struct **Member**
*The **Member** element.*

struct VariableMember
*The Variable **Member** element.*

struct FunctionMember
*The Function **Member** element.*

struct **Param**
*The **Param** element.*

Ridefinizioni di tipo (typedefs)

typedef multimap< string, **Member** * > **MemberTable**
*The **Compound Member** Table.*

typedef vector< **Param** * > **ParamTable**
*The Function **Param** Table.*

Tipi enumerati (enum)

enum Compound_Kind { Compound_Generic = 0, Compound_File, Compound_Class, Compound_Struct, Compound_Namespace }
enum Member_Kind { Member_Variable, Member_Function }

Funzioni

template<> **Compound** * **create< Compound >** (**Compound_Kind** kind, const string &name, const string &fileLocator, const string &sourceLocator)

Create.

template<> void release< Compound > (Compound *)

Destroy.

template<class type> **type** * **create** (...)

Default Constructor.

template<class type> void **release** (**type** *obj)

Default Destructor.

template<> **VariableMember** * **create< VariableMember >** (const string &name, const string &**type**)

Create.

template<> **FunctionMember** * **create< FunctionMember >** (const string &name, const string &**type**)

Create.

template<> void release< Member > (Member *)

Destroy.

template<> **Param** * **create< Param >** (const string &name, const string &**type**)

Create.

template<> void **release< Param >** (**Param** *)

Destroy.

Documentazione delle ridefinizioni di tipo (typedefs)

typedef multimap<string, Member *> LegaSIE::MemberTable

The **Compound Member** Table.

typedef vector<Param *> LegaSIE::ParamTable

The Function **Param** Table.

Documentazione dei tipi enumerati

enum LegaSIE::Compound_Kind

Valori dei tipi enumerati:

Compound_Generic
Compound_File
Compound_Class
Compound_Struct
Compound_Namespace

enum LegaSIE::Member_Kind

Valori dei tipi enumerati:

Member_Variable
Member_Function

Documentazione delle funzioni

template<class type> type* LegaSIE::create (...) [inline]

Default Constructor.

template<> Compound* LegaSIE::create< Compound > (Compound_Kind *kind*, const string & *name*, const string & *fileLocator*, const string & *sourceLocator*)

Create.

template<> FunctionMember* LegaSIE::create< FunctionMember > (const string & *name*, const string & *type*)

Create.

template<> Param* LegaSIE::create< Param > (const string & *name*, const string & *type*)

Create.

template<> VariableMember* LegaSIE::create< VariableMember > (const string & *name*, const string & *type*)

Create.

template<class type> void LegaSIE::release (type * *obj*) `[inline]`

Default Destructor.

template<> void LegaSIE::release< Compound > (Compound *)

Destroy.

template<> void LegaSIE::release< Member > (Member *)

Destroy.

template<> void LegaSIE::release< Param > (Param *)

Destroy.

LegaSIE-Toolkit Documentazione delle classi

Riferimenti per la classe LegaSIE::AnalyseCpp

`#include <AnalyseCpp.h>`

Membri pubblici statici

static int **analyse** (const string &tmpdir, const string &fileLocator, vector< **Compound** * > &compounds) throw (string)
Analyse c/c++ file and return found elements.

Documentazione delle funzioni membro

static int LegaSIE::AnalyseCpp::analyse (const string & *tmpdir*, const string & *fileLocator*, vector< Compound * > & *compounds*) throw (string) [static]

Analyse c/c++ file and return found elements.

La documentazione per questa classe è stata generata a partire dal seguente file:

- **AnalyseCpp.h**

Riferimenti per la classe LegaSIE::AnalyseEngine

`#include <AnalyseEngine.h>`

Membri pubblici statici

static int **analyse** (const string &tmpdir, const string &fileLocator, map< string, **Compound** * > &compounds) throw (string)

Documentazione delle funzioni membro

static int LegaSIE::AnalyseEngine::analyse (const string & *tmpdir*, const string & *fileLocator*, map< string, Compound * > & *compounds*) throw (string) [static]

Analyse input file and merge found elements.

Parametri:

tmpdir working dir
fileLocator the input file URL
compounds the element list updated

Valori di ritorno:

0=all_done,-1=not_applicable

La documentazione per questa classe è stata generata a partire dal seguente file:

- **AnalyseEngine.h**

Riferimenti per la classe LegaSIE::AnalyseObj

`#include <AnalyseObj.h>`

Membri pubblici statici

static int **analyse** (const string &tmpdir, const string &fileLocator, vector< **Compound** * > &compounds) throw (string)

Analyse object file and return found elements.

Documentazione delle funzioni membro

static int LegaSIE::AnalyseObj::analyse (const string & *tmpdir*, const string & *fileLocator*, vector< Compound * > & *compounds*) throw (string) `[static]`

Analyse object file and return found elements.

La documentazione per questa classe è stata generata a partire dal seguente file:

- **AnalyseObj.h**

Riferimenti per la struct LegaSIE::Compound

The **Compound** element.

`#include <Compound.h>`

Attributi pubblici

string **name**

*The **Compound** name.*

Compound_Kind kind
*The **Compound** kind.*

MemberTable members
*The **Compound** members.*

string fileLocator
*The **Compound** location.*

string sourceLocator
*The **Compound** source.*

Descrizione Dettagliata

The **Compound** element.

Documentazione dei dati membri

string LegaSIE::Compound::fileLocator

The **Compound** location.

Compound_Kind LegaSIE::Compound::kind

The **Compound** kind.

MemberTable LegaSIE::Compound::members

The **Compound** members.

string LegaSIE::Compound::name

The **Compound** name.

string LegaSIE::Compound::sourceLocator

The **Compound** source.

La documentazione per questa struct è stata generata a partire dal seguente file:

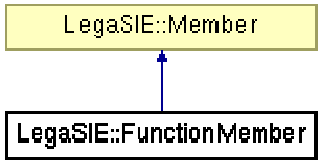
- **Compound.h**

Riferimenti per la struct LegaSIE::FunctionMember

The Function **Member** element.

`#include <Member.h>`

Diagramma delle classi per LegaSIE::FunctionMember



Attributi pubblici

ParamTable params

The Function params (if any).

Descrizione Dettagliata

The Function **Member** element.

Documentazione dei dati membri

ParamTable LegaSIE::FunctionMember::params

The Function params (if any).

La documentazione per questa struct è stata generata a partire dal seguente file:

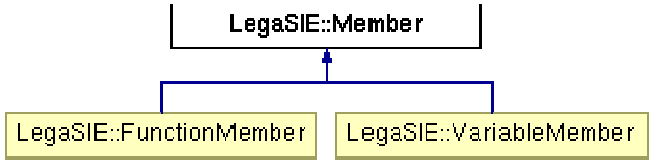
- **Member.h**

Riferimenti per la struct LegaSIE::Member

The **Member** element.

`#include <Member.h>`

Diagramma delle classi per LegaSIE::Member



Attributi pubblici

string **name**
*The **Member** name.*

string **type**
*The **Member** type.*

Member_Kind kind
*The **Member** kind.*

Descrizione Dettagliata

The **Member** element.

Documentazione dei dati membri

Member_Kind LegaSIE::Member::kind

The **Member** kind.

string LegaSIE::Member::name

The **Member** name.

string LegaSIE::Member::type

The **Member** type.

La documentazione per questa struct è stata generata a partire dal seguente file:

- **Member.h**

Riferimenti per la struct LegaSIE::Param

The **Param** element.

```
#include <Param.h>
```

Attributi pubblici

string **name**
*The **Param** name (if any).*

string **type**
*The **Param** type.*

Descrizione Dettagliata

The **Param** element.

Documentazione dei dati membri

string LegaSIE::Param::name

The **Param** name (if any).

string LegaSIE::Param::type

The **Param** type.

La documentazione per questa struct è stata generata a partire dal seguente file:

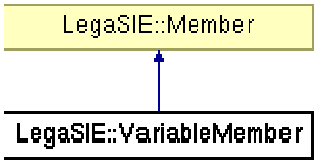
- **Param.h**

Riferimenti per la struct LegaSIE::VariableMember

The Variable **Member** element.

`#include <Member.h>`

Diagramma delle classi per LegaSIE::VariableMember



Descrizione Dettagliata

The Variable **Member** element.

La documentazione per questa struct è stata generata a partire dal seguente file:

- **Member.h**

LegaSIE-Toolkit Documentazione dei file

Riferimenti per il file AnalyseCpp.cpp

```
#include "AnalyseCpp.h"
#include <sys/wait.h>
#include <sys/types.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <AL.h>
```

```
#include <AL_XMLParser.h>
#include <Compound.h>
#include <Member.h>
#include <Param.h>
#include "doxygen.h"
#include <output.h>
```

Namespace

namespace **LeSIE**

Definizioni

#define **DBG**(args...) if (doDebug()) OUT(args); else

Funzioni

static bool **isQuiet** ()
static bool **doDebug** ()
static void **print** (AL_XMLElement *tree, string tabs="")

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Tue Sep 20 2005

Revision

1.0.1

Implementation of C++ Analyse module.

Documentazione delle definizioni

#define DBG(args...) if (doDebug()) OUT(args); else

Documentazione delle funzioni

static bool doDebug () `[static]`

static bool isQuiet () `[static]`

static void print (AL_XMLElement * *tree*, string *tabs* = "") `[static]`

Riferimenti per il file AnalyseCpp.h

```
#include <string>
#include <vector>
```

Namespace

namespace **LegaSIE**
namespace **std**

Composti

class LegaSIE::AnalyseCpp

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Tue Sep 20 2005

Revision

1.0.1

Definition of C++ Analyse module.

Riferimenti per il file AnalyseEngine.cpp

```
#include "AnalyseEngine.h"
#include "AnalyseCpp.h"
#include "AnalyseObj.h"
#include <Compound.h>
```

Variabili

struct {
  const char * **type**
    analyse module type

  int(* **call**)(const string &, const string &, vector< Compound * > &)
    analyse module call

} AnalyseTable []

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Tue Sep 20 2005

Revision

1.0.1

Implementation of Analyse Engine.

Documentazione delle variabili

struct { ... } AnalyseTable[] `[static]`

int(* call)(const string &, const string &, vector< Compound * > &)

analyse module call

const char* type

analyse module type

Riferimenti per il file AnalyseEngine.h

```
#include <string>
#include <map>
```

Namespace

namespace **LegaSIE**

Composti

class LegaSIE::AnalyseEngine

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Tue Sep 20 2005

Revision

1.0.1

Definition of Analyse Engine.

Riferimenti per il file AnalyseObj.cpp

```
#include "AnalyseObj.h"
#include <sys/wait.h>
#include <sys/types.h>
#include <errno.h>
#include <libgen.h>
#include <stdlib.h>
```

```
#include <stdio.h>
#include <AL.h>
#include <Compound.h>
#include <Member.h>
#include <Param.h>
#include "TypeParser.h"
#include <output.h>
```

Definizioni

#define **DBG**(args...) if (doDebug()) OUT(args); else

Funzioni

static bool **doDebug** ()

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Tue Oct 4 2005

Revision

1.0.1

Implementation of (Shared) Object Analyse module.

Documentazione delle definizioni

#define DBG(args...) if (doDebug()) OUT(args); else

Documentazione delle funzioni

static bool doDebug () `[static]`

Riferimenti per il file AnalyseObj.h

```
#include <string>
#include <vector>
```

Namespace

namespace **LegaSIE**

Composti

class LegaSIE::AnalyseObj

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Tue Oct 4 2005

Revision

1.0.1

Definition of (Shared) Object Analyse module.

Riferimenti per il file Compound.cpp

```
#include <Compound.h>
#include <Member.h>
```

Funzioni

template<> Compound * **create< Compound >** (**Compound_Kind** kind, const string &name, const string &fileLocator, const string &sourceLocator)
template<> void **release< Compound >** (Compound *compound)

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date

Wed Sep 28 2005

Revision

1.0.1

Implementation of Compound element.

Documentazione delle funzioni

template<> Compound* create< Compound > (Compound_Kind *kind*, const string & *name*, const string & *fileLocator*, const string & *sourceLocator*)

template<> void release< Compound > (Compound * *compound*)

Riferimenti per il file Compound.h

```
#include <string>
#include <map>
#include <LegaSIE.h>
```

Namespace

namespace **LegaSIE**

Composti

struct LegaSIE::Compound
*The **Compound** element.*

Ridefinizioni di tipo (typedefs)

typedef multimap< string, Member * > **LegaSIE::MemberTable**
*The **Compound Member** Table.*

Tipi enumerati (enum)

enum LegaSIE::Compound_Kind { LegaSIE::Compound_Generic = 0, LegaSIE::Compound_File, LegaSIE::Compound_Class, LegaSIE::Compound_Struct, LegaSIE::Compound_Namespace }

Funzioni

template<> Compound * **LegaSIE::create< Compound >** (**Compound_Kind** kind, const string &name, const string &fileLocator, const string &sourceLocator)

Create.

template<> void **LegaSIE::release< Compound >** (Compound *)

Destroy.

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Tue Sep 20 2005

Revision

1.0.1

Definition of Compound element.

Riferimenti per il file LeSIE.h

Namespace

namespace **LegaSIE**

Funzioni

template<class type> **type * LegaSIE::create** (...)

Default Constructor.

template<class type> void **LegaSIE::release** (**type** *obj)

Default Destructor.

Riferimenti per il file Member.cpp

```
#include <Member.h>
#include <Param.h>
```

Funzioni

template<> VariableMember * **create< VariableMember >** (const string &name, const string &**type**)

Create.

template<> FunctionMember * **create< FunctionMember >** (const string &name, const string &**type**)

Create.

template<> void **release< Member >** (Member *member)

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Thu Sep 29 2005

Revision

1.0.1

Implementation of Member element.

Documentazione delle funzioni

template<> FunctionMember* create< FunctionMember > (const string & *name*, const string & *type*)

Create.

template<> VariableMember* create< VariableMember > (const string & *name*, const string & *type*)

Create.

template<> void release< Member > (Member * *member*)

Riferimenti per il file Member.h

```
#include <string>
#include <vector>
#include <LegaSIE.h>
```

Namespace

namespace **LegaSIE**

Composti

struct LegaSIE::Member
*The **Member** element.*

struct LegaSIE::VariableMember
*The Variable **Member** element.*

struct LegaSIE::FunctionMember
*The Function **Member** element.*

Ridefinizioni di tipo (typedefs)

typedef vector< Param * > **LegaSIE::ParamTable**
*The Function **Param** Table.*

Tipi enumerati (enum)

enum LegaSIE::Member_Kind { LegaSIE::Member_Variable, LegaSIE::Member_Function }

Funzioni

template<> VariableMember * **LegaSIE::create< VariableMember >** (const string &name, const string &**type**)
Create.

template<> FunctionMember * **LegaSIE::create< FunctionMember >** (const string &name, const string &**type**)
Create.

template<> void **LegaSIE::release< Member >** (Member *)
Destroy.

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Tue Sep 20 2005

Revision

1.0.1

Definition of Member element.

Riferimenti per il file Param.cpp

```
#include <Param.h>
```

Funzioni

template<> Param * **create< Param >** (const string &name, const string &**type**)
Create.

template<> void **release< Param >** (Param *param)

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,gasiino@medialab.pa.icar.cnr.it

Date

Thu Sep 29 2005

Revision

1.0.1

Implementation of Param element.

Documentazione delle funzioni

template<> Param* create< Param > (const string & *name*, const string & *type*)

Create.

template<> void release< Param > (Param * *param*)

Riferimenti per il file Param.h

```
#include <string>
#include <LegaSIE.h>
```

Namespace

namespace **LegaSIE**

Composti

struct LegaSIE::Param
*The **Param** element.*

Funzioni

template<> Param * **LegaSIE::create< Param >** (const string &name, const string &**type**)
Create.

template<> void **LegaSIE::release< Param >** (Param *)
Destroy.

Descrizione Dettagliata

Autore:

Fabio Collura, Gabriele Siino (C) 2005 colf,`gasiino@medialab.pa.icar.cnr.it`

Date

Tue Sep 20 2005

Revision

1.0.1

Definition of Param element

REFERENCES - RIFERIMENTI BIBLIOGRAFICI

[1] A.Machì, S. Lombardo, G. Siino, M. Tripiciano. “Integrazione di librerie scientifiche nell’ambiente di programmazione a componenti GRID.IT”“ Technical Report ICAR-CNR Dept. Palermo RT-ICAR-PA-03-11 Dicembre 2003

[2] M. Vanneschi: The programming model of ASSIST, an environment for parallel and distributed portable applications. Parallel Computing 28(12): 1709-1732 (2002)

[3] Web Services Activity http://www.w3.org/2002/ws/

[4] A. Machì, F. Collura, S. Lombardo. “Legacy Software Integrating Environment. Metodologie, patterns e tools per l’integrazione nell’ambiente di programmazione Grid.it“ Technical Report ICAR-CNR Dept. Palermo RT-ICAR-PA-04-15 Dicembre 2004

[5] G. Siino: “Descrizione di Elementi Software in contesto di ambiente a componenti ad alte prestazioni su griglia computazionale”. Tesi di Laurea in Ingegneria Informatica Univ. Palermo AA. 2004-2005

[6] A. Machì, F. Collura, , S. Lombardo. “Modellazione UML dello Skeleton di coordinamento di un Componente Parallelo Master-Slave e di patterns per il controllo esterno della sua performance“ Technical Report ICAR-CNR Dept. Palermo RT-ICAR-PA-05-03 Marzo 2005

[7] V. Morici. “Un sistema intelligente pr la previsione del tempo di esecuzione di componenti di libreria”Tesi di Laurea in Ingegneria Informatica Univ. Palermo AA. 2003-2004

[8] Workflow Management Coalition, The Workflow Reference Model (WFMC-TC-1003, 19-Jan-95, 1.1), http://www.wfmc.org

[9] A. Machì, F. Collura, S. Lombardo: “Dependable Execution of Workflow Activities on a Virtual Private Grid Middleware. V. S. Sunderam et als. (Eds.) Computational Science ICCS 2005 LNCS 3516 pp.267-274, 2005 Springer-Verlag 2005

[10] M. Aldinucci, A. Petrocelli, E. Pistoletti, M. Torquati, M. Vanneschi, L. Veraldi, and C. Zoccolo, Dynamic reconfiguration of grid-aware applications in ASSIST, in 11th Intl Euro-Par 2005: Parallel and Distributed Computing, LNCS, Lisboa, Portugal, August 2005

[11] M. Aldinucci, S. Campa, M. Coppola, M. Danelutto, D. Laforenza, D. Puppin, L. Scarponi, M. Vanneschi, C. Zoccolo "Components for high performance Grid programming in the Grid.it Project". In Proc. Of Intl. Workshop on Component Models and Systems for Grid Applications

[12] S. Lombardo, V. Graziano, A. Machì. “Euristiche per il Controllo della QoS di componenti grid-enabled (modelli e patterns)” Technical Report ICAR-CNR Dept. Palermo RT-ICAR-PA-05-12 Novembre 2005

[13] Extensible Markup Language (XML) http://www.w3.org/XML

[14] XML-Schema http://www.w3.org/XML/Schema

[15] Dimitri van Heesch. Doxygen. www.doxygen.org

[16] Web Services Description Language (WSDL) v,1.1 http://www.w3.org/TR/wsdl

[17] OWL Web Ontology Language http://www.w3.org/TR/owl-features, Resource Description Framework (RDF) http://www.w3.org/RDF

[18] A. Machì, F. Collura: “Skeleton di componenti master-slave per la parallelizzazione di moduli legacy” Technical Report ICAR-CNR Dept. Palermo RT-ICAR-PA-05-02 Marzo 2005

[19] F.Collura. “Una piattaforma di supporto CAE al restauro supervisionato di filmati digitali” Tesi di Laurea in Ingegneria Informatica Univ. Palermo AA. 2004-2005

[20] XSL Transformations (XSLT) v,1.0 http://www.w3.org/TR/xslt

[21] Robert van Engelen “gSOAP: C/C++ Web Services and Clients” http://www.cs.fsu.edu/~engelen/soap.html

[22] SOAP v,1.1 specification: http://www.w3.org/TR/soap

[23] S. Lombardo, A. Machì “Grid-Aware serviCEs administrator: un server di Web-services, operante sotto vincoli di Qualità di Servizio. “ Technical Report ICAR-CNR Dept. Palermo RT-ICAR-PA-05-13 Dicembre 2005