



OCTAPY 3: UN SISTEMA DISTRIBUITO E COOPERATIVO ORIENTATO AL WEB SEMANTICO

Paolo Acampa, Mario Mango Furnari, Carmine Ivan Delio Noviello

Octapy è il CMS sviluppato dal CNR creato per realizzazione di sistemi documentali distribuiti e cooperanti, con l'obiettivo di costruire "comunità di collaborazione" a tecnologia web, per la creazione di una rete di conoscenze condivise. Octapy è interamente realizzato a partire dal linguaggio di programmazione Python, e si avvale dell'application server Zope e del CMS Plone quali componenti di pubblicazione ed interfacciamento verso il web. L'architettura del sistema utilizza le recenti tecnologie di "programmazione per componenti" sperimentate col framework CA di Zope 3. Octapy, fornisce specifiche componenti per lo sviluppo di applicazioni cooperative, integrando il protocollo OAI per lo scambio di metadati e fornendo strumenti per la creazione esplicita di relazioni tra i contenuti di uno o più sistemi, organizzati come siti web così da garantire la "identità" dei partecipanti alle comunità virtuali.

Octapy, inoltre, può gestire contenuti di tipo cartografico, così da facilitare l'utilizzo della cartografia come strumento di mediazione e correlazione di contenuti. A questo scopo è stato sviluppato un sottosistema cartografico per la realizzazione di applicazioni che necessitano del trattamento geografico.

In questo articolo sarà illustrata l'architettura della nuova versione del framework Octapy, focalizzando l'attenzione sul modello component-based adottato, illustrando le tecniche e le funzionalità introdotte per aggirare vincoli e limitazioni della piattaforma Zope/Plone. Sarà descritto il formato OCML, XML-based, per la configurazione e descrizione dei contenuti, il modello architetturale e le interfacce di programmazione definite per consentire lo sviluppo di un'architettura estensibile ed interoperabile, con particolare attenzione sull'adozione di standard aperti basati su XML per l'interfacciamento dati e OAI per l'interoperabilità dello scambio di metadati. Inoltre, sarà illustrato come Octapy interfaccia i sistemi cartografici, presentando moduli Python specifici per la cartografia e le limitazioni che questi presentano e che Octapy mira ad aggirare. Infine, sarà illustrato il caso di applicazione del circuito "campaniabeniculturali.it", comunità virtuale promossa dalla Direzione Regionale per i Beni Culturali e Paesaggistici della Campania per la promozione e diffusione delle conoscenze sui beni culturali della Campania. Comunità virtuale in cui circa 50 musei della Regione Campania adoperano con successo il framework Octapy per la creazione della prima ed unica comunità virtuale italiana dedicata al settore dei beni culturali in cui ad ogni soggetto aderente viene garantita l'autonomia gestionale e la propria identità accanto ai vantaggi della cooperazione. Tratto questo visibile nella realizzazione di percorsi e tematismi intermuseali.



I Content Management System e lo sviluppo delle digital library

Parallelamente allo sviluppo e alla diffusione delle tecnologie digitali, in particolar modo delle reti di interconnessione e dalle comunicazioni a banda larga, si è andato sempre più affermando il bisogno da parte di istituzioni, enti, organi governativi e soggetti privati di usufruire di strumenti e tecnologie atte alla memorizzazione, catalogazione, indicizzazione, ricerca e, più in generale, al trattamento di qualsivoglia tipologia di dati, da semplici documenti di testo sino ad arrivare alla catalogazione di tutti i contenuti di un'istituzione, come ad esempio i contenuti di intere biblioteche (siano essi libri o informazioni sulla catalogazione di libri), grandi basi documentali tecnico-scientifiche, i dati relativi alle transazioni economiche effettuate da un istituto bancario. Si parla quindi, in generale, di *gestione di contenuti* come il processo che mira alla definizione di metodologie di trattamento dei dati, alla creazione di eventuali strutture e/o istituzioni preposte a tale trattamento ed infine, ma non ultimo per importanza, alla realizzazione di specifiche tecnologie informatiche idonee ai compiti individuati.

Da una prospettiva di tipo storico, la gestione di contenuti ha subito un lungo processo di crescita ed evoluzione che è coinciso sia con l'aumentare delle potenzialità offerte dalle tecnologie dell'informazione sia con la diffusione a basso costo di reti di comunicazione, che hanno permesso di poter estendere il concetto semplice di "basi documentali". Il primo passo nell'utilizzo delle tecnologie informatiche per la memorizzazione di grandi dati è stato rivolto alla creazione di appositi strumenti software per immagazzinare più istanze di contenuti appartenenti ad un unico modello (o al più ad una famiglia) di dati: con lo sviluppo e la diffusione dei *database* si sono potuti sfruttare i calcolatori elettronici per l'immagazzinamento sistematico di dati, enucleando ed esplicitando le eventuali relazioni che possono sussistere fra essi. Il classico esempio è rappresentato dall'informatizzazione delle biblioteche, grazie alla quale si è resa particolarmente agevole sia la ricerca di testi sia il recupero dei dati addizionali che connotano e denotano un particolare libro (autore, tipologia di libro, codici ISBN, così come recensioni, ecc): i *metadati*. Con il crescente impiego dei metadati, cioè dei "dati che parlano di dati", come strumento di annotazione per arricchire l'informazione principale, si è andato sviluppando il concetto di *collezione digitale*, dove l'elemento principale non è più rappresentato solo dai dati o dalle tipologie di contenuti che si va a gestire, bensì anche dai metadati e dalla loro tipologia, grazie ai quali è possibile l'immagazzinamento e il trattamento di tipi eterogenei di dati e l'interoperabilità con sistemi ed istituzioni differenti. In questo scenario che muta, a cambiare è soprattutto il modo di operare delle istituzioni preposte alla gestione – e spesso alla definizione – delle collezioni digitali, che si trovano a dover cambiare il proprio ruolo da quello di semplici organi preposti al mantenimento e alla catalogazione di contenuti a strutture il cui compito preciso è quello di sviluppare tecniche organizzative, standard di rappresentazione e, per l'appunto, metadati che possono permettere forme di interoperabilità tra istituzioni, con l'unico obiettivo comune di creare una rete virtuale di conoscenza distribuita, in cui ogni nodo di questa rete ha la responsabilità di "farsi interpretare" dagli altri. Adoperando un'unica parola, la *cooperazione* rappresenta l'elemento cardine delle collezioni digitali o, di come attualmente si indica in maniera più

generale, delle *digital library* - [BORG00].

Questa evoluzione nel trattamento dei dati, siano essi digitali o meno, è certamente coincisa con un radicale cambiamento sia sul fronte delle tecnologie informatiche sia su come queste operano per garantire la cooperazione tra sistemi eterogenei. Col passare del tempo, i database sono passati dal ruolo di elementi portanti e centrali dei sistemi di catalogazione a quello di "anello" di una catena ben più complessa in cui altre componenti software e non solo giocano un ruolo determinante: non si parla più di semplici *database systems* ma di sistemi per la gestione di basi documentali o, come attualmente è in uso, di *Content Management System* (CMS). Un CMS è quindi un sistema che permette di organizzare e gestire contenuti, fornendo all'utente *facilities* per svolgere tali compiti e agli sviluppatori soluzioni software che permettano agevolmente la definizione di nuove tipologie di contenuti ed eventualmente estendere il sistema con nuove funzionalità in base ai requisiti (utente, funzionali e di dominio) dell'applicazione specifica che si sta modellando.

Un CMS adoperato per realizzare una *digital library*, nell'accezione che si sta qui considerando e cioè di *sistemi a tecnologia digitale distribuiti e cooperativi per la gestione di contenuti e di metadati*, deve soddisfare tutta una serie di requisiti e vincoli, i cui principali possono essere individuati nei seguenti:

- ✓ *Indipendenza dai formati e sistemi di memorizzazione*: nell'ottica dell'interoperabilità, un CMS per una *digital library* deve essere in grado di astrarre dalla particolare soluzione software (intesa sia come sistema di memorizzazione sia come formato) di immagazzinamento dati, fornendo flessibilità sufficiente a potersi adattare ai vari metodi oggi disponibili e al tempo stesso interfacciandosi verso i più diffusi formati di scambio e tecnologie di rappresentazione di dati, come XML e RDF. Quest'ultimo è largamente adoperato nei sistemi "semantic oriented" in cui modelli di dati basati su ontologie permettono agevolmente l'interfacciamento verso altri sistemi e banche dati.
- ✓ *Gestione delle legacy*: un CMS deve essere in grado di poter gestire eventuali *legacy* che potrebbero presentarsi all'atto dell'integrazione all'interno dell'istituzione che decide di adottare tecnologie per le *digital library*. Il termine *legacy* è qui inteso nell'accezione più ampia: sono da considerarsi *legacy* sia eventuali vincoli sui dati e sulla loro struttura, sia la necessità di dover far coesistere software preesistenti difficilmente sostituibili per motivi non tecnologici. Non sono, infatti, rari i casi in cui i costi di formazione del personale non sono tali da giustificare l'adozione di nuove tecnologie; inoltre, spesso l'adozione di nuove tecnologie software è vissuto dal personale dell'istituzione come ostacolo al lavoro e come "distrazione" dall'obiettivo primario che rimane quello di espletare le attività dell'istituzione e la gestione dei dati.
- ✓ *Vincoli istituzionali*: in molti casi, assieme a requisiti funzionali e di sistema, relativi alla parte di gestione di contenuti e l'interoperabilità verso altri sistemi, nei requisiti di dominio sono presenti veri e propri vincoli istituzionali che debbono essere presi in considerazione nella realizzazione o configurazione del CMS. Problematiche inerenti

normative sulla diffusione dei dati, vincoli di privacy e segretezza o semplici vincoli sulla fruizione da parte del pubblico possono comportare lo sviluppo di funzionalità *ad hoc* per una particolare istituzione. Un CMS quindi affronta anche la problematica del "flusso dei dati" con specifici sottosistemi di *workflow* che devono poter essere adattati alle esigenze specifiche.

- ✓ *Salvaguardia dei dati e delle identità istituzionali*: un aspetto non di secondaria importanza è rappresentato dalla sicurezza dei dati. Un CMS per una *digital library* deve tener conto della possibilità di cooperazione con altri sistemi, e quindi dell'utilizzo di reti di comunicazione a basso livello di sicurezza, fornendo specifici meccanismi per la protezione dei dati e l'autenticazione delle utenze. Inoltre, altro aspetto configurabile, come forma di "vincolo di istituzione", ma connesso con la privacy dei dati, è la salvaguardia delle identità dei soggetti coinvolti nella creazione di reti di contenuti: molte istituzioni custodiscono patrimoni informativi vasti, che spesso hanno richiesto ingenti sforzi economici, di risorse umane ed intellettive, e quasi sempre mostrano grandi resistenze all'uso indiscriminato e senza controllo delle proprie risorse digitali (spesso si oppongono proprio alla digitalizzazione). Un CMS in questo scenario deve fornire meccanismi di "controllo" della cooperazione, dando ampia possibilità di scelta di quali risorse mettere in condivisione ed eventualmente stabilire regole di condivisione particolareggiate.
- ✓ *Standard web*: oggi, nella quasi totalità dei casi, si parla più genericamente di CMS per indicare sistemi di gestione di contenuti a tecnologia web, che adoperano quindi Internet sia come strumento di accesso e pubblicazione di contenuti, sia come strumento di cooperazione e mediazione. In questo scenario, diventa fondamentale per un CMS il pieno supporto verso i più diffusi standard web di interoperabilità, presentazione, scambio dati ed accessibilità.

Il CMS Octapy nasce con l'obiettivo di fornire uno strumento per la creazione di comunità virtuali distribuite e cooperative a tecnologia web, basate sull'interoperabilità e il rispetto dei vincoli e delle identità culturali ed istituzionali di ogni singolo soggetto aderente alla comunità. Per questo motivo, Octapy soddisfa tutti i requisiti elencati, ed introduce soluzioni software mirate alla realizzazione, configurazione, estensibilità e manutenibilità dei sistemi.

Aspetti tecnologici delle *digital library* e limiti dei CMS

La caratterizzazione di una *digital library*, descritta nel paragrafo precedente, se da un lato fissa dei paletti ben precisi sulle caratteristiche utente che un CMS deve fornire, e quindi sulle funzionalità di base che deve implementare, dall'altro solleva una serie di problematiche tecnologiche non banali, la cui soluzione spesso richiede un'analisi più approfondita del ruolo svolto dai contenuti, o più in generale dai documenti gestiti dal sistema, e sulle soluzioni software che, pur se largamente condivise tra i vari CMS disponibili, pongono troppe limitazioni e restrizioni nella realizzazione delle *digital*

library.

Semplificando le cose al massimo, da un punto di vista squisitamente tecnico un CMS è costituito da un *Content Management Framework* (CMF) che attraverso un'interfaccia di programmazione ben definita consente di realizzare tutte quelle funzionalità che permettono all'utente di gestire i contenuti (inserimento, modifica, cancellazione, pubblicazione, ecc). Tipicamente in un CMF i contenuti sono composti di due parti principali: quella dei dati vera e propria che costituiscono il documento – quasi sempre questa parte stabilisce anche quale forma di memorizzazione, o *storing*, deve essere associata al dato – e una parte di logica di applicazione necessaria per interfacciare il dato alle funzionalità del CMF. In questo senso, il documento all'interno del CMF è una componente software, e tipicamente nei CMF orientati agli oggetti l'implementazione classica è quella di realizzare i contenuti sottoforma di classi (si parla di *classi contenuto*) le cui istanze rappresentano i dati, e arricchire tali classi dei metodi necessari all'integrazione nel CMF. Un CMS, però, non è solo costituito dalla logica di applicazione: per consentire l'interazione con l'utente, un CMS fornisce anche tutte le funzionalità necessarie alla presentazione, e nel caso dei CMS a tecnologia web ci si avvale dei servizi di base di un *application server*. Un esempio è rappresentato dalla piattaforma Zope/Plone, in cui l'*application server* Zope fornisce l'ambiente di esecuzione e pubblicazione su web, il *framework* CMF è la parte di gestione dei contenuti ed infine Plone rappresenta l'elemento di collante e mediazione delle funzionalità di un CMS verso l'utente.

Questo modello architetturale di CMS rappresenta senza ombra di dubbio un approccio sufficientemente generico nel caso in cui ci si trova a dover gestire realtà in cui vi sono pochi vincoli da applicare alla gestione dei dati, una casistica relativamente bassa di tipologie di contenuti e che soprattutto rientrano agevolmente negli "schemi" imposti dal CMS, e un livello di interazione ridotto o nullo e senza meccanismi di cooperazione esplicita. Nel caso di CMS a tecnologia web, quel modello architetturale trova ottimo impiego nella realizzazione di siti web con una base documentale autonoma, con requisiti poco stringenti sull'interazione con l' "esterno", e che quindi possono limitarsi al semplice strumento di "collegamento ipertestuale" per referenziare altri contenuti. Nel caso in cui si vuole adoperare tali strumenti per la realizzazione di *digital library*, e più genericamente di comunità virtuali web, sorgono una serie di problemi che richiedono un cambiamento di prospettiva sulla nozione di documento e sulle modalità di uso che se ne fanno. Quello che emerge è che la riflessione più importante che deve essere effettuata riguarda la "visione" del documento all'interno del CMS, e come questa influisce sull'utilizzo di tali strumenti per lo sviluppo di *digital library*.

In un CMS il documento, in quanto contenuto, svolge il ruolo di entità adibita al mantenimento dei dati, e quindi svolge un ruolo passivo nei confronti delle attività del CMS. Tutto è realizzato in funzione della pubblicazione e della gestione dei contenuti, e di come questi appaiono all'utente finale, senza che il contenuto svolga un ruolo attivo sia nel processo di gestione dei contenuti stessi sia nei meccanismi di funzionamento e cooperazione con altri sistemi. In una *digital library*, invece, il contenuto svolge un ruolo attivo, espone struttura e funzionalità tali da poterlo utilizzare sia come elemento di una collezione di dati (ottenendo la *collezione*

digitale) sia come strumento di mediazione e cooperazione con altri sistemi (il passaggio verso una *digital library* e la costruzione di una rete complessa di conoscenze condivise - **[BORG00]**). In questo senso, il contenuto diventa *documento* e al tempo stesso *interfaccia* per la cooperazione con altri sistemi, centro e periferia dello stesso sistema, nodo e radice di una struttura gerarchica. Si passa quindi ad un documento che ha più modalità di fruizione e rappresentazione: un documento *multivalente* in grado di esprimere la sua struttura e le interazioni con altri documenti e collezioni digitali - **[GLUS05]**.

Alla luce di questo cambio di scenario, deve essere di conseguenza rivisto il ruolo di "aggregazione" dei contenuti. In un CMS *classico* un aggregato è rappresentato da un contenitore di contenuti: l'aggregazione è effettuata in maniera "posizionale", inserendo all'interno i contenuti che si ritiene parte dell'aggregazione. Lo strumento di aggregazione è null'altro che un insieme semplice, dove la responsabilità di "aggregare" è demandata all'utente finale che stabilisce come correlare i contenuti. Nelle *digital library* il concetto di aggregazione si estende, fino a raggiungere quello di "aggregati distribuiti" in cui una serie di documenti cooperano affinché si creino le aggregazioni all'interno della comunità, e queste aggregazioni a loro volta possano essere ricorsivamente aggregate e correlate. Emerge quindi che gli aggregatori sono rappresentabili a loro volta con dei documenti, dotati di funzionalità così da specificare regole e meccanismi di aggregazione, che siano semplici criteri di aggregazione sintattici o posizionali (ad esempio, aggregazione su tutti i documenti di un certo "tipo" o che soddisfano un dato criterio sul contenuto di un campo) sia criteri più complessi basati su vincoli semantici e di interpretazione dei documenti e delle relazioni (ad esempio, aggregazioni basate su ontologie o modelli di interpretazione).

A partire da quest'analisi è possibile enucleare in maniera precisa quali sono gli aspetti critici di un CMS convenzionale che limitano pesantemente l'utilizzo di tali ambienti software per la creazione di comunità virtuali, e nello specifico quelle a tecnologia web:

- ✓ *Limitazioni sul ruolo dei contenuti*: come già accennato, gli attuali CMS benché progettati per gestire contenuti, limitano il ruolo dei contenuti a quello di aggregati di dati, nel tentativo di affrontare il problema nella maniera più generica possibile. Un documento non è altro che una sequenza ordinata di campi di un ben determinato tipo, il cui uso è generalmente limitato agli usi di base offerti dal CMS (visualizzazione, modifica, ordinamento parziale, ecc.). Per una *digital library* è necessario, invece, che un sistema di gestione di contenuti diventi un *sistema orientato ai contenuti* (COS - *Content Oriented System*), in cui sia possibile disporre di più rappresentazioni di un documento e contemporaneamente lo stesso svolga un ruolo attivo nella costruzione delle relazioni tra i contenuti del sistema o di altri sistemi che partecipano ad una rete di CMS cooperanti. In questo senso, un documento diviene *multivalente* ed è allo stesso tempo dato e componente software, componente il cui comportamento non è limitato all'integrazione funzionale e sistematica col CMS ma che punta agli aspetti di integrazione e cooperazione. Inoltre, deve essere possibile fornire modelli di interpretazione di un documento non convenzionali per il CMS in esame, così da favorire allo stesso tempo l'integrazione di realtà apparentemente

disomogenee ma che possono essere correlate. Un esempio di questo aspetto è rappresentato dalle limitazioni che alcuni CMS impongono sul sottoinsieme minimale dei campi che un documento deve possedere, sottoinsieme che spesso non permette la naturale trasposizione di documenti non digitali senza stravolgimenti metodologici o di significato per il tipo di documento che si sta trattando.

✓ *Scarso riuso di componenti content-oriented*: il passaggio da un CMS ad un COS implica la reimplementazione di alcune componenti del CMS e l'introduzione di nuove funzionalità, come ad esempio potrebbero essere quelle di cooperazione e aggregazione distribuita. Questo significa che, in generale, c'è una perdita di genericità da parte di queste nuove componenti, nel momento in cui la struttura e l'uso del documento diventano fondamentali per la loro realizzazione. Quasi tutti gli attuali CMS forniscono librerie di rapida prototipizzazione per i contenuti – gli *Archetype* Plone sono un esempio, così come gli *zope.schema* di Zope 3 – ma questi approcci risultano di scarsa utilità quando si passa ad una ridefinizione del ruolo del documento nel CMS, e diventano addirittura sorgenti di colli di bottiglia nel riuso nel momento in cui si cambiano i modelli di interpretazione dei contenuti. Questo problema è in parte dovuto anche al modello di programmazione di queste librerie: le classi contenuto sono arricchite utilizzando l'ereditarietà multipla e, in taluni CMS con l'eredità dinamica (*mix-in programming*); le funzionalità necessarie ad interagire con l'ambiente del CMF e/o dell'*application server*, limitano fortemente le possibilità di espansione e riuso sia all'interno dello stesso CMS sia al di fuori. Inoltre, non bisogna assolutamente minimizzare gli effetti indotti sui contenuti da aspetti di interazione con l'utente finale: per quanto i CMS tentino di tenere nettamente separati gli aspetti di persistenza da quelli di logica e da quelli di presentazione, troppo spesso si finisce col rendere le componenti accoppiate tra loro, col risultato di limitare ancor di più il riutilizzo di moduli software aggiuntivi – con un'*archetype* si cura la parte di persistenza, logica e presentazione di un *contenttype*. Realizzare software riusabile può, quindi, diventare problematico, e questo pone forti limitazioni allo sviluppo di comunità cooperanti aperte agli standard di interoperabilità ma che vogliono preservare una propria autonomia di gestione ed organizzazione dei dati: se interoperare deve significare adottare standard di organizzazione dei dati rigidi, la stessa interoperabilità diventa una contraddizione in termini. Inoltre, spesso vi sono realtà scientifiche in cui standard apparentemente "rigidi" subiscono interpretazioni molto particolari, col risultato che ogni singola istituzione costruisce ed utilizza propri modelli di interpretazione, spesso in conflitto con gli altri. Tuttavia, una ridefinizione del modello di sviluppo può consentire di aggirare tali ostacoli, senza dover necessariamente aggirare del tutto le funzionalità offerte dal CMS per lo sviluppo di nuove tipologie di contenuti, bensì integrandole ed eventualmente analizzandole con una prospettiva differente. Ancora una volta, il passaggio cruciale consiste nel considerare un contenuto come una componente software a sé stante che rappresenti un *layer* di un'architettura *n-tier*, e non un'istanza particolare del - e nel - livello di gestione dati - [KIRT99].

- ✓ *Difficoltà di deployment di schemi di dati e dati stessi:* l'impossibilità di separare in maniera netta la parte di contenuto da quella di logica e presentazione di un *contenttype* oltre ad avere riflessi sul riuso del codice – difficoltà nel realizzare software che operi indipendentemente dalla strutturazione dei dati, con la tendenza a sviluppare codice per un particolare schema – ha dirette conseguenze sulla possibilità di effettuare il *deployment* in realtà istituzionali dove larga parte dei modelli di dati sono condivisi – e con essi l'interpretazione dei singoli campi – tranne che per un limitato sottoinsieme. Un esempio classico è rappresentato dai campi "didascalia" di una o più immagini delle schede di catalogo adoperate nel settore dei beni culturali: a partire dalla specifica ICCD – **[ICCD]** – per una data tipologia di scheda (Oggetto Artistico, Oggetto Archeologico, ecc), le singole istituzioni possono assegnare significati diversi alla specifica del ruolo di "didascalia", in quanto lo stesso standard non contempla casi di fruizione per un pubblico non tecnico. Questo aspetto, tutt'altro che banale, ha pesanti ripercussioni sul ciclo di *deployment* del software, sia dal punto di vista di organizzazione e sia per quanto riguarda la tempistica del lavoro – tempo necessario per effettuare il *deploy*, mantenimento e aggiornamento di più versioni separate dello stesso pacchetto software (*branch*) – sia dal punto di vista dell'uso delle risorse umane e tecniche: un'operazione, che si palesa come una normale fase di configurazione, deve essere per forza di cose eseguita da personale specializzato (programmatori), cui spetta l'onere ultimo di gestire diverse sottoversioni degli stessi pacchetti software. Un impatto ancor più drammatico si ha sul fronte della cooperazione allorché tutte le istituzioni decidono, in una qualche misura, di voler cooperare: per comunicare il proprio modello di interpretazione per lo schema dei dati diventa una necessità, pena l'impossibilità di contribuire alla costruzione di una base di conoscenza comune e distribuita. L'esempio classico è rappresentato dal progetto *ReMuNa* – **[REMU01]** – e dal successivo progetto *campaniabenculturali.it*, in cui più di 50 realtà museali (è già previsto l'allargamento a 100 unità in un immediato futuro) cooperano nella creazione di una rete di promozione e diffusione delle identità culturali della Campania, pur mantenendo una propria autonomia di gestione ed organizzazione dei dati, degli schemi e dei modelli di interpretazione. Inoltre, queste limitazioni comportano ovvie ripercussioni anche sull'adattabilità del software a realtà preesistenti, impedendo la migrazione alle nuove piattaforme senza dover alterare (e in alcuni casi stravolgere) il modello, l'organizzazione, l'interpretazione e soprattutto l'uso dei dati. Emerge, quindi, con maggiore chiarezza la stringente necessità di considerare i contenuti come una componente del sistema, componente che deve soddisfare sia i vincoli di integrazione all'interno del *framework* del CMS/CMF sia le esigenze di flessibilità di una comunità virtuale.
- ✓ *Scarso supporto alla cooperazione:* la prerogativa principale delle *digital library* è la cooperazione tra soggetti differenti, cooperazione non intesa come condivisione implicita di risorse hardware/software (database comuni e centralizzati), bensì come condivisione di risorse digitali distribuite e patrimonio delle singole istituzioni aderenti alla comunità. In pratica, ad essere distribuiti non sono i supporti di memorizzazione o gli schemi di dati, ma i documenti

di ogni singola istituzione, ed è la loro struttura e il loro subset di metadati a costituire l'interfaccia di mediazione tra le reti di istituti cooperanti. Questo importante aspetto evidenzia che gli attuali CMS non possono concorrere, visto il come sono strutturati, nella costruzione di comunità virtuali. Il motivo principale di ciò è da ricercare nella visione ristretta dei contenuti a semplici collezioni di dati. La necessità è, ancora una volta, quella di possedere documenti *multivalenti*, che possono agevolmente interfacciarsi fra loro con l'ausilio dei più diffusi standard di interoperabilità come l'*Open Archives Initiative* (OAI) per l'interscambio e l'*harvesting* di metadati - **[OAI]** -, specifiche come il *Dublin Core* (DC) per l'interpretazione dei metadati e l'XML, con tutti gli standard correlati (XSD, XSLT, ecc), come strumento per l'interscambio di dati e strutture. Inoltre, fondamentale importanza assume la doppia natura di ogni contenuto, e cioè quella di dato e documento strutturato, così come per gli aggregati. Infine, per i CMS a tecnologia web diventa fondamentale l'adozione di tecnologie di identificazione delle risorse digitali sul web, come il *Digital Object Identifier* (DOI) e l'*Handle System*, che permettono di referenziare in maniera assoluta i documenti attraverso servizi di directory di contenuti, e condividerli tra comunità differenti.

L'architettura a componenti di Octapy 3

Per soddisfare i requisiti individuati nei paragrafi precedenti, Octapy 3 introduce un *framework component-based* che estende le funzionalità della piattaforma Zope/Plone in un'ottica distribuita, cooperativa e *content oriented*. Da un punto di vista architetturale, Octapy 3 può essere suddiviso in 3 livelli ben distinti, schematizzati in **Figura 1**.

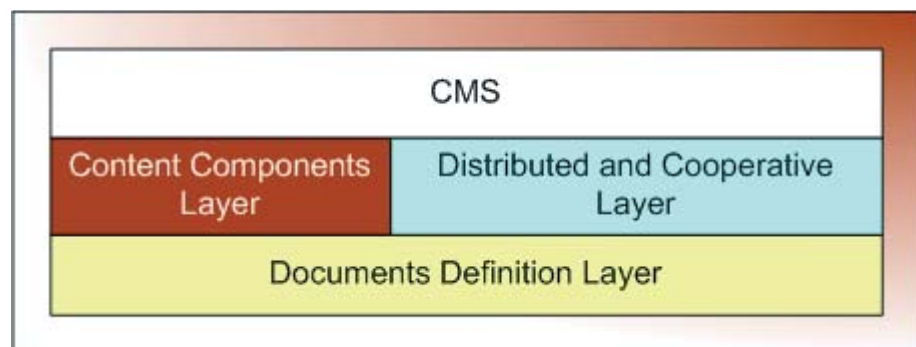


Figura 1 - Schematizzazione dell'architettura a componenti di Octapy 3

- ✓ *Documents Definition Layer*: è il livello che si occupa della definizione dei documenti e al relativo modello di interpretazione. Attraverso questo *layer* sono definiti i nuovi contenuti disponibili per l'utente del CMS, l'interpretazione assegnabile ai singoli campi della struttura del documento, gli eventuali vocabolari associati. Inoltre, i modelli di definizione fungono da elemento di mediazione

tra i sistemi coinvolti nella rete collaborativa: lo scambio dei documenti di definizione dei contenuti permette di conoscere le tipologie di dati possedute da un sistema, oltre alla struttura e al modello di interpretazione. Il *Document Definition Layer* (DDL) è ampiamente utilizzato sia nei moduli di interazione utente sia in alcune componenti della parte distribuita, come l'*OAI subsystem*.

- ✓ *Content Components Layer*: questo *layer* si occupa della generazione delle "componenti contenuto", con le quali è possibile effettuare tutte le operazioni classiche di un CMS (inserimento, modifica, cancellazione, gestione) e attraverso le quali i contenuti sono interfacciati anche con i sottosistemi che si occupano della remotizzazione dei contenuti e della cooperazione tra i sottosistemi. Questo livello assicura l'indipendenza dal particolare modello dei dati, e consente l'espansione e la personalizzazione del CMS in maniera flessibile ed efficiente.
- ✓ *Distributed and Cooperative Layer*: è il *layer* costituito dalle componenti che si occupano della parte distribuita e cooperativa del CMS. I moduli di questo *layer* provvedono alla creazione delle differenti rappresentazioni dei contenuti, supportando il protocollo OAI, standard come il DC, e i modelli di rappresentazione basati su XML. Grazie a questi sottosistemi è possibile aggregare contenuti di altri sistemi, esportare a loro volta contenuti ed aggregati, strutturare e correlare entità differenti, come documenti ed entità cartografiche.
- ✓ *CMS*: è la parte di CMS propriamente detta, in cui viene curata parte della logica di applicazione, la presentazione con l'utente, la gestione dei *workflow*. Questo livello è realizzato adoperando Plone 2.5.x e il prossimo Plone 3.x, fornisce un approccio componentizzato alle viste e, quindi, all'interfaccia utente, con la possibilità di fornire elementi di adattività in base alle tipologie di utenza, i contenuti e gli impieghi del CMS. Coordina, inoltre, le attività col sottosistema cartografico, *Octapy MapServer* (OMS).

Nel seguito di questo documento saranno analizzati nel dettaglio ciascuno dei livelli dell'architettura di Octapy 3, ponendo particolare attenzione sulle scelte metodologiche e progettuali che hanno portato all'attuale definizione del sistema. Infine, sarà presentato un caso reale d'uso del *framework* Octapy 3, quello del circuito www.napolibeniculturali.it.

Introduzione ad OCML

Uno dei requisiti più importanti di un CMS progettato per le *digital library* è quello di consentire una rapida definizione dei contenuti. Definizione che sia al tempo stesso strumento per l'interfacciamento verso altri sistemi e che permetta di definire un modello specifico di interpretazione dei singoli campi, il tutto senza dover ricorrere a competenze strettamente informatiche. Il *framework* Octapy 3 affronta questo requisito introducendo il formato di scambio e configurazione OCML (*Octapy Configuration Markup Language*). L'OCML è un formato di specifica basato su XML, che ne permette la funzione

di mediazione tra sistemi che vogliono cooperare nella costruzione di una rete di sistemi, ad esempio, per conoscere quali contenuti un dato sistema implementa, e quale sia il modello di interpretazione per questi dati è sufficiente saper interpretare opportunamente il formato OCML.

L'OCML è definito utilizzando 3 *namespace*. Un file OCML minimale ha la seguente struttura:

```
<octapy xmlns:view="http://namespaces.remuna.org/octapy-view"  
  xmlns:data="http://namespaces.remuna.org/octapy-data"  
  xmlns:storage="http://namespaces.remuna.org/octapy-storage">  
...  
</octapy>
```

- ✓ *namespace data*: attraverso le direttive di questo *namespace* è possibile definire le tipologie di documenti istanziabili nel sistema, la loro struttura (intesa sia come quali campi e di che tipo sia la loro articolazione in sotto-sezioni), eventuali vocabolari associati. Inoltre, è possibile stabilire relazioni con altri documenti, con criteri differenziati, ad esempio, le relazioni semplici si basano su criteri di padre-figlio o contenitore-contenuto (un documento può essere "sotto-documento di" o "composto-di"); le relazioni complesse sfruttano informazioni di correlazione provenienti da altri sottosistemi, come ad esempio ontologie, per creare reti dinamiche di relazioni fra i documenti. Con le direttive di questo *namespace* è possibile anche specificare quale interpretazione dare alle strutture o ai campi di un documento: l'assegnazione di un'interpretazione si basa su un processo di "etichettatura" dei campi. Le etichette saranno interpretate dai sottosistemi a ciò deputati, e le opportune azioni saranno eseguite. Nel seguito viene mostrato un esempio minimale di utilizzo delle direttive del *namespace data*:

```

<octapy xmlns:view="http://namespaces.remuna.org/octapy-view"
xmlns:data="http://namespaces.remuna.org/octapy-data"
xmlns:storage="http://namespaces.remuna.org/octapy-storage">
  <data:document name="Oggetto Archeologico" id="OggettoArcheologico
    isContainerish="True" subtypes="(RepertoArcheologico,)">
    <data:section name="generale">
      <data:field name="id" type="Int" languageIndependent="True"
        default="1000" mode="r" />
      <data:field name="titolo" type="String" searchable="True"
        languageIndependent="false" required="True"
        default="Titolo"/>
      <data:field name="descrizione" type="Text" searchable="true"
        languageIndependent="false" required="false"
        metadata="{ 'uiuse': 'description' }" />
      <data:field name="inventario" type="int" searchable="true"
        languageIndependent="true" required="false" />
      <data:field name="immagine" type="Image" searchable="false"
        languageIndependent="true" required="true"
        default="img.jpg"/>
    </data:section>
  </data:document>
</octapy>

```

Con questo frammento di codice OCML viene indicato al sistema di generare un nuovo contenuto "Oggetto Archeologico", che ha un relazione "è composto di" con il contenuto "Reperto Archeologico", e che è caratterizzato da una serie di campi. Da notare l'utilizzo dell'attributo *metadata* per il campo "descrizione" che fissa l'interpretazione del campo: in questo modo, se tale contenuto viene esportato verso un altro sistema che non possiede una propria descrizione (o che ha un'interpretazione differente dei suoi campi) quel campo viene correttamente interpretato e gestito.

- ✓ *namespace storage*: le direttive di questo *namespace* consentono di configurare quale schema di *storage* applicare ai contenuti o ad una loro parte. È possibile applicare alcune direttive a livello globale, e altre a livello specifico dei singoli campi: in questo modo è possibile avere contenuti che possono essere composti di parti provenienti da altri sottosistemi, così da gestire meglio particolari *legacy*. Nel seguito viene mostrato un esempio minimale di utilizzo delle direttive del *namespace storage*:

```

<octapy xmlns:view="http://namespaces.remuna.org/octapy-view"
  xmlns:data="http://namespaces.remuna.org/octapy-data"
  xmlns:storage="http://namespaces.remuna.org/octapy-storage">
  <storage:apply
    for="OggettoArcheologico"
    type="Postgres">
    <param name="dbname">marcheo</param>
    <param name="user">username</param>
  </storage:apply>
  <storage:apply
    for="RepertoArcheologico"
    type="bind-marshalling">
    <param name="repository">
      http://marcheo.napolibeniculturali.it
    </param>
    <param name="user">username</param>
  </storage:apply>
</octapy>

```

Da notare lo *storing* di tipo "bind-marshalling", con il quale è possibile aggregare e gestire i contenuti di un sistema remoto, completi di specifica e modello di interpretazione, utilizzando il sottosistema di *marshalling*.

- ✓ *namespace view*: le direttive di questo *namespace* consentono di specificare aspetti dei documenti connessi sia con la presentazione all'utente finale sia per particolari formati di output strutturato dei singoli documenti (*Dublin Core* (DC) e altri formati di rappresentazione/metadattazione). Nel seguito viene mostrato un esempio minimale di utilizzo delle direttive del *namespace view*:

```

<octapy xmlns:view="http://namespaces.remuna.org/octapy-view"
  xmlns:data="http://namespaces.remuna.org/octapy-data"
  xmlns:storage="http://namespaces.remuna.org/octapy-storage">
  <view:widget
    fordata="Text"
    type="StringWidget"
    <view:configure>
      <view:param name="maxlength">255</view:param>
    </view:configure>
  </view:widget>
  <view:dc
    for="OggettoArcheologico"
  </view:dc>
</octapy>

```

In questo frammento di codice OCML viene assegnato uno *StringWidget* a tutti i campi di tipo "Text". Il come sarà gestito questo tipo di *widget* è compito esclusivo del CMS o dei moduli deputati alla presentazione. Da notare che la direttiva *view:dc* indica che il documento

deve presentarsi con una rappresentazione DC compatibile.

La specifica OCML prevede la possibilità di costruire in maniera "incrementale" i file di configurazione, attraverso direttive di inclusione, così da semplificare il processo di creazione dei file, favorirne una migliore organizzazione e manutenibilità. Inoltre, l'OCML nasce come specifica "aperta" nel senso che è possibile aggiungere nuove direttive. Mediante la direttiva "meta" è, infatti, possibile specificare interpreti aggiuntivi per nuove tipologie di direttive. In questo modo, componenti aggiuntive possono sfruttare il livello di definizione dei documenti per esprimere relazioni o semplici configurazioni tra i documenti e la componente stessa. Per la specifica completa del formato OCML si rimanda a [ACAM07].

Il Component Model di Octapy 3

Durante l'analisi dei requisiti tecnologici che un *framework* di gestione documentale per le *digital library* deve soddisfare, è emerso in modo netto che uno dei limiti degli attuali CMS è lo scarso riuso dei moduli software, quando si passa da uno sviluppo di moduli per un sistema di gestione dei contenuti ad un sistema orientato ai contenuti (*Content Oriented System*). Il punto di svolta si ha nel momento in cui i documenti sono concepiti come componenti software a se stanti, e non si parla più di *layer* di dati, ma *layer* di *componenti contenuto*, è cioè di oggetti che forniscono un'interfaccia di programmazione prestabilita, attraverso la quale è possibile costruire le dipendenze con gli altri moduli software.

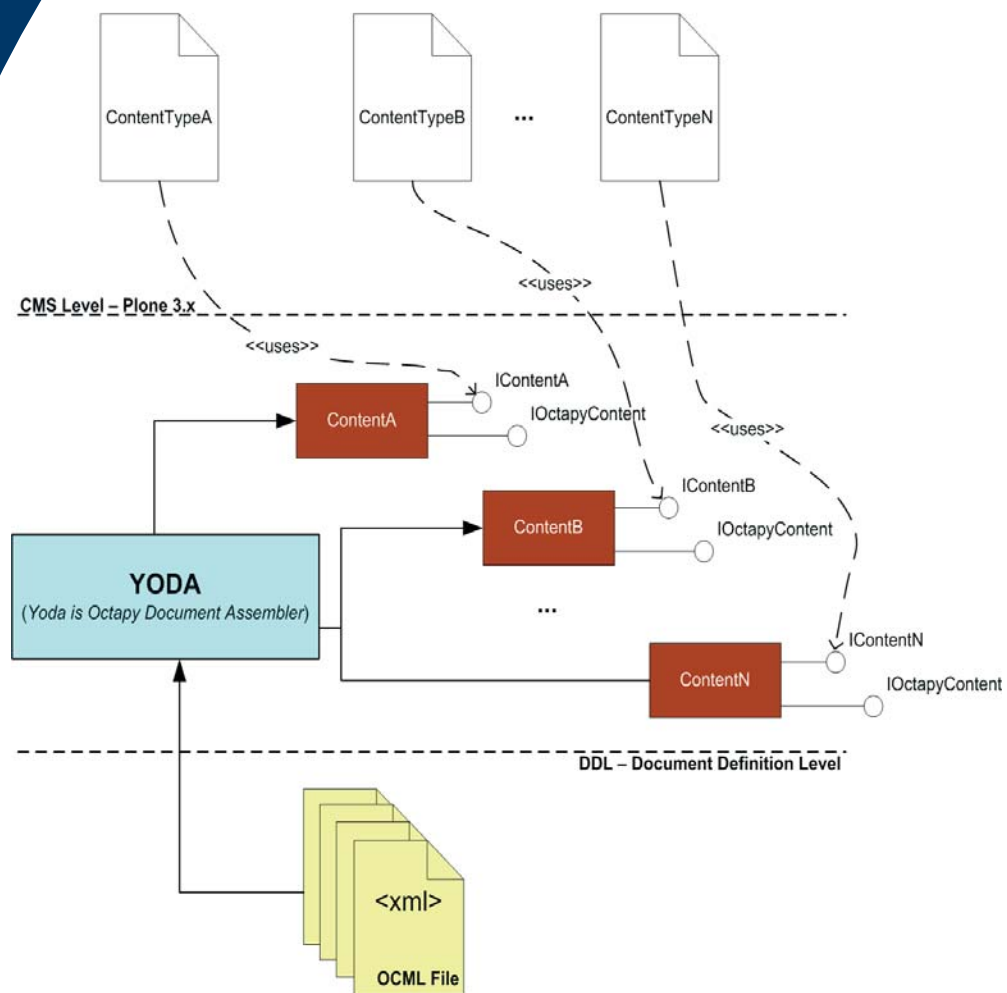


Figura 2 – Schematizzazione del Component Model di Octapy 3

A partire dal formato OCML, Octapy genera le componenti contenuto e le relative interfacce, attraverso le quali tutte le altre componenti (CMS stesso, moduli di interoperabilità - OAI, DC -, ecc) interagiranno con i documenti. Gli stessi documenti, quando dovranno interagire con altri contenuti lo faranno utilizzando le loro interfacce. Al centro di questo processo c'è YODA (acronimo ricorsivo di *Yoda is Octapy Document Assembler*) che assemblando le direttive OCML (sia quelle locali al sistema, sia quelle recuperate da sistemi remoti) genera le componenti contenuto.

Come è possibile osservare dalla **Figura 2**, ogni tipologia di contenuto espone sempre almeno 2 interfacce: la prima è *IOctapyContent*, comune a tutti i contenuti; la seconda è l'interfaccia specifica del particolare contenuto, che espone lo schema di dati, il modello di interpretazione, le relazioni con altri contenuti e gli eventuali metodi di gestione. Al centro del processo di interfacciamento verso una componente contenuto c'è l'interfaccia *IOctapyContent*, un modulo software che si interfaccia con un contenuto può sfruttare l'immutabilità di questa interfaccia per conoscere la reale interfaccia (o le interfacce) del contenuto (questo processo di *lookup*

è analogo a quello introdotto in altri *framework* per componenti, come il COM di Microsoft, in cui l'interfaccia *IUnknown* permette di ottenere le interfacce implementate tramite il metodo *queryInterface()* – [EDDO98]), ed adoperarla opportunamente. L'interfaccia *IOctapyContent* espone una serie di metodi, in particolar modo *getContentInterface()*, che restituisce l'interfaccia del contenuto. Il seguente codice Python mostra come è possibile in maniera semplice e generica interfacciarsi ad un contenuto:

```
# Il seguente pseudo-codice mostra come sia possibile
# interfacciarsi con le classi contenuto in maniera generica e flessibile
# sfruttando il component-model di Octapy 3
from Products.Octapy.interfaces import IOctapyContent

# getContent() è una funzione che si ipotizza restituire un contenuto
content = getContent()
# Il component-model assicura che un contenuto possieda l'interfaccia
# IOctapyContent
if content.directlyProvides(IOctapyContent):
    iface = content.getContentInterface()
    print "Inspecting contenttype: ", iface.getTaggedValue("contenttype")
    for attr in iface:
        if "uiuse" in iface[attr].getTaggedValues():
            #Stampa gli attributi a cui è associato uno specifico modello
            #di interpretazione per l'uso nel layer di presentazione
            uiuse = iface[attr].getTaggedValue("uiuse")
            value = iface[attr].value
            print "Special attr '%s' with uiuse '%s': %s" % (attr, uiuse, value)
        else:
            value = iface[attr].value
            type = iface[attr].getTaggedValue('datatype')
            print "Attr '%s': %s(%s)" % (attr, value, type)
```

Ad esempio, per un contenuto istanza di *OggettoArcheologico*, definito nel file OCML di esempio del precedente paragrafo, si ottiene il seguente *output*:

```
Inspecting contenttype: OggettoArcheologico
Attr 'id': 124(Int)
Attr 'titolo': Statua di pescatore negro da Messina(String)
Special attr 'descrizione' with uiuse 'description': La statua raffigura
un...
Attr 'inventario': 94323234(Int)
Attr 'immagine': <byte data>(Image)
```

Questo esempio mette in evidenza aspetti non banali. Innanzitutto, il *framework* consente di astrarre dal tipo di memorizzazione dei documenti (DB, file, ODB, ecc), dalla loro localizzazione (contenuti locali o remoti), dalla loro organizzazione e rappresentazione. Inoltre, attraverso l'utilizzo di funzioni generiche, come *getTaggedValue()*, è possibile ricavare annotazioni provenienti direttamente dal formato OCML (attributo "metadata"). L'interpretazione di queste annotazioni sarà poi demandata ai sottosistemi

specifici che dovranno gestirle: ad esempio, la direttiva 'uiuse' sarà interpretata dal livello di presentazione; la direttiva 'dc', invece, sarà adoperata dal sottosistema di gestione del *Dublin Core*, come meccanismo di *mapping* dei metadati.

```
...  
<data:field name="descrizione_breve" type="Text"  
  metadata="{ 'dc': 'title' }"/>  
<data:field name="autore_scheda" type="Text"  
  metadata="{ 'dc': 'author' }"/>  
...
```

Infine, attraverso l'interfaccia della componente contenuto è possibile risalire sia ai suoi attributi (*field*) sia al tipo e ad altre informazioni accessorie.

Allo stesso modo con cui è stata definita l'interfaccia di base *IOctapyContent* dei contenuti all'interno del sistema, così è stata definita un'apposita interfaccia per gli aggregatori, *IOctapyContainer*. Attraverso questa interfaccia è possibile ricevere le istanze degli oggetti contenuto, siano essi locali sia remoti.

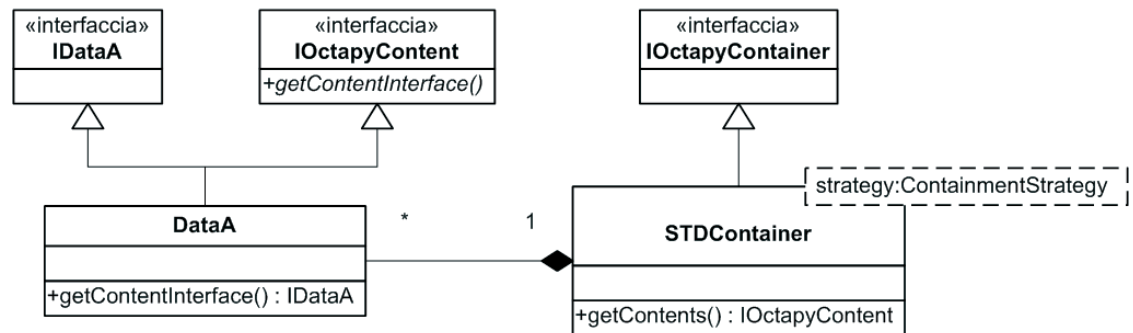


Figura 3 – Il modello ad oggetti e le relazioni tra *IOctapyContainer* e *IOctapyContent*

I *container* possono essere parametrizzati in accordo con una strategia di aggregazione – [GAMM95], così da aggregare contenuti sia in base a meccanismi di tipo posizionale o basato sui riferimenti, sia in base a criteri di aggregazione semantici, attraverso l'uso di ontologie. La specifica "strategia" di aggregazione provvederà ad inviare una query al sottosistema ontologico, il quale restituirà i contenuti risultanti – [AIEL06]. In **Figura 3** è riportato il diagramma delle classi e delle interfacce coinvolte.

Dal punto di vista implementativo, Octapy 3 sfrutta i servizi della *Component Architecture* (CA) introdotta con Zope 3, adoperando i meccanismi di *adapting* per gestire la corretta compatibilità verso il software preesistente, in particolar modo Plone e CMF – [WEIT07], [NOVI07].

Supporto al distribuito e cooperativo

Octapy CMS nasce con l'obiettivo primario di fornire funzionalità per lo sviluppo di comunità virtuali a tecnologia web, e quindi fornisce strumenti specifici per la cooperazione tra sistemi autonomi ed indipendenti. In Octapy ogni contenuto può essere reso distribuito, e quindi acquisibile da qualsiasi altro sistema partecipante ad una rete di sistemi "trusted". Questo è possibile grazie al fatto che in Octapy ogni documento è fornito di più rappresentazioni digitali, che vanno da quelle *user-oriented* per la fruizione da parte dell'utente finale, a quelle strutturate in XML per essere interfacciate ed interpretate da sistemi software locali o remoti. Il sottosistema di gestione distribuita e cooperativa consente di gestire su un server locale qualsiasi contenuto remoto come se si stesse operando in locale. Opportune routine di *marshalling* si incaricano dell'interpretazione dei documenti XML alla base dello scambio di dati, generando oggetti *proxy* che implementano la stessa interfaccia dell'oggetto remoto, astruendo, quindi, dalle problematiche di interpretazione e comunicazione. I documenti XML scambiati forniscono, pertanto, tutte le informazioni necessarie all'individuazione del contenuto remoto, incluso i modelli di interpretazione. In questo modo, è sempre possibile adattare i modelli nel caso differiscano, considerato che il sistema non utilizza metadati prefissati, ma utilizza annotazioni con le relative interpretazioni. Ad esempio, se in remoto, il campo descrizione è rappresentato da un campo diverso da quello locale, utilizzando la direttiva *'uiuse'* è possibile adattarlo senza alcuna conseguenza sul codice. Il seguente frammento di codice mostra come sono facilmente ottenibili i contenuti esportati da un dato server, e come si può sfruttare il fatto che il contenuto porta con se il modello di interpretazione (si noti come due istituti museali interpretano in maniera differente il campo *descrizione* di un *Oggetto Artistico*):

```
>>> from Products.Octapy.utilities import discoverContent
>>> from Products.Octapy.interfaces import IOctapyContainer, IOctapyProxy
#Effettua la richiesta remota dei contenuti esportati
#Restituisce l'elenco dei container
>>> remoteContents = discoverContent("http://capodim.napolibeniculturali.it")
>>> remoteContents[0].directlyProvides(IOctapyContainer)
True
>>> contents = remoteContents[0].getContents()
>>> for c in contents:
...     if c.directlyProvides(IOctapyContent): #E' un contenuto
...         print c.getContentInterface().getTaggedValue("contenttype")

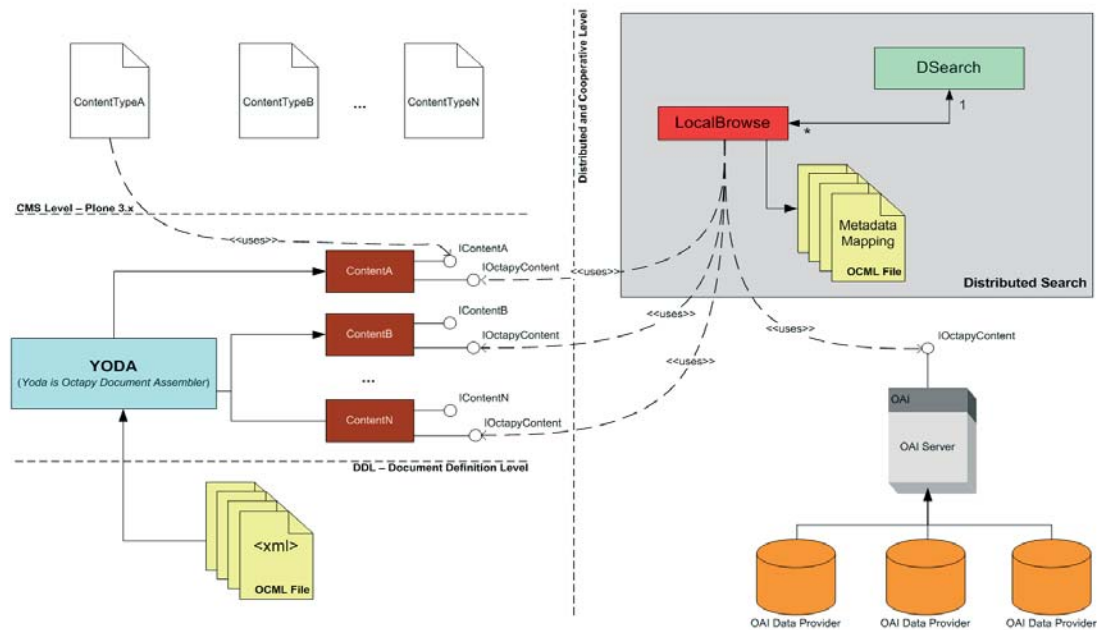
OggettoArtistico
OggettoArtistico
.....
>>> c.directlyProvides(IOctapyProxy)
True
#Il contenuto è esportato in sola lettura
>>> c.mode
'r'
>>> iface = c.getContentInterface()
#Per il Museo di Capodimonte, il campo descrizione
#dell'OggettoArtistico è 'oss'
>>> iface['oss'].getTaggedValue('uiuse')
'description'
```

```

>>> remoteContents = discoverContent("http://preale.
napolibeniculturali.it")
.....
>>> iface = c.getContentInterface()
#Per il Museo di Palazzo Reale, il campo descrizione
#dell'OggettoArtistico è 'descrizione_completa'
>>> iface['descrizione_completa'].getTaggedValue('uiuse')
'description'

```

Octapy, però, non si limita all'interoperabilità tra i sistemi che adottano la stessa piattaforma software (o lo stesso modello di scambio di dati), ma mira a garantire l'interoperabilità verso altri sistemi, adottando modelli di esportazione dei dati e protocolli ritenuti *standard de facto*, come DC e OAI. In **Figura 4** è riportato l'interazione del sottosistema di ricerca distribuita e metadatozione con la restante parte del *framework* Octapy.



Octapy 3 Component Architecture Overview
OAI sub-system

Figura 4 – Il Component Model di Octapy 3 e il sottosistema di gestione dei metadati OAI-based

Octapy, infine, fornisce pieno supporto al protocollo *Handle System* (meccanismo di riferimento universale su web dei contenuti), e supporta pienamente il *mapping* verso il modello dei metadati proposti dal progetto europeo *MICHAEL* – [MICH01].

La componente cartografica: il modulo OMS

Il framework Octapy fornisce uno specifico sottosistema per la gestione cartografica: il modulo *Octapy MapServer*

(OMS). Il modulo *OMS* è un modulo di gestione cartografica che fornisce servizi e strumenti per la memorizzazione strutturata di mappe, la generazione di rappresentazioni delle stesse (ad esempio immagini) e la gestione di contenuti associati. Il modulo *OMS* è interamente realizzato con il linguaggio di programmazione Python, ed espone un'interfaccia di programmazione (API) ad oggetti, attraverso cui è possibile accedere alle funzionalità offerte ed eventualmente estenderlo in base alle proprie esigenze. Il modulo *OMS* è stato progettato nell'ottica della massima flessibilità, e con l'obiettivo di fornire una molteplicità di rappresentazioni differenti a partire da un'unica memorizzazione strutturata. Inoltre, per particolari esigenze, è possibile realizzare gestori personalizzati per la generazione della rappresentazione di mappe. *OMS* fornisce, inoltre, specifici strumenti per semplificare l'integrazione del modulo in ambienti *client/server*, permettendo di accedere alla maggior parte delle funzionalità attraverso il protocollo di trasporto HTTP e adottando l'XML come formato di scambio dati.

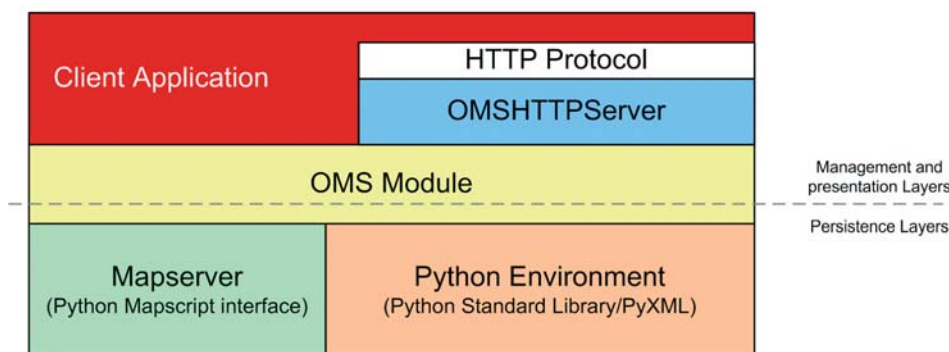


Figura 5 – L'architettura del modulo OMS

Il modulo *OMS*, da un punto di vista architetturale, può essere logicamente suddiviso in due livelli: uno dedicato alla gestione della persistenza delle mappe e uno dedicato alla gestione e rappresentazione delle memorizzazioni strutturate di mappe. Come è possibile osservare dalla **Figura 5** il modulo *OMS* demanda parte di questi compiti a librerie terzi, e in particolar modo tutta la parte di gestione della persistenza e della generazione di immagini grafiche di mappe al modulo *MapServer*, prodotto *Open Source* sviluppato dall'Università del Minnesota e disponibile in **[MAPS01]**. Il modulo *OMS* fornisce più di 10 possibilità per la persistenza delle mappe, tra cui spiccano il formato *ShapeFile*, standard *de facto* per la memorizzazione strutturata di mappe, *PostgreSQL* (attraverso l'estensione *PostGIS*), *Oracle* (tramite *OracleSpatial*) e *MySQL*. Inoltre, attraverso la nozione di *layer*, è possibile accedere contemporaneamente a più fonti dati in un'unica mappa, o istanziare *layer* dinamici provenienti da contenuti presenti nel CMS Octapy.

Il modulo *OMS* consente di generare più rappresentazioni a partire dalla stessa mappa, e fornisce un *framework* specifico per l'implementazione di gestori della rappresentazione personalizzati. Il modulo *OMS*, in configurazione standard, fornisce i seguenti gestori di rappresentazione:

- ✓ *Gestore output grafico*: è il gestore che provvede

alla generazione grafica (sotto forma di immagine) di una mappa. Allo stato attuale sono implementati i seguenti formati di immagine: GIF, JPEG, WBMP, PNG.

- ✓ *Gestore output formato XRef*: XRef è un formato di scambio basato su XML che nasce per l'impiego della libreria OMS con il modulo *OMSHTTPServer*, server che risponde in base a chiamate HTTP. L'obiettivo del formato XRef è quello di agevolare la realizzazione dei livelli di presentazione nelle applicazioni progettate secondo la logica *3-tier*, rafforzando la separazione tra il codice di presentazione e quello di gestione cartografica. In generale, il formato XRef si dimostra idoneo in tutte quelle situazioni in cui è necessario separare e astrarre dall'interfaccia di programmazione del modulo OMS. Per maggiori informazioni sul formato XRef si rimanda al paragrafo "Il formato XRef" di **[FURN05a]**.
- ✓ *Gestore output formato XML*: il modulo OMS consente di generare la rappresentazione in formato XML della memorizzazione strutturata di una mappa. Questa funzionalità consente di accedere in maniera trasparente e portabile alle informazioni contenute all'interno dei *mapfile*, i file contenenti le informazioni relative ad una mappa. Il formato XML consente alle applicazioni *client* di ignorare i dettagli implementativi connessi con il particolare tipo di fonte dati adoperata (*ShapeFile*, *PostGIS*, ecc.) e allo stesso tempo di disporre di informazioni utili per l'interfacciamento con il modulo di gestione cartografica. La rappresentazione in XML prevede, opzionalmente, la generazione di alcune informazioni in formato GML 3.1.1. Per maggiori informazioni sul formato XML si rimanda al paragrafo "Il formato XML" in **[FURN05a]**.

Grazie alla possibilità di rappresentare in XML le informazioni cartografiche, con il modulo OMS i dati cartografici diventano documenti digitali, dotati di una precisa struttura e, quindi, diventano elementi di una collezione digitale. In questo modo è possibile far diventare i dati cartografici "contenuto" gestibile da un CMS, favorendo il processo di gestione (le mappe e i contenuti associati sono gestiti via web, senza l'ausilio di strumenti proprietari terzi) e pubblicazione. Maggiori informazioni sul modulo OMS possono essere reperite in **[OMS]**.

L'esperienza del circuito www.napolibeniculturali.it

Il CMS Octapy è frutto di anni di esperienza scientifica e tecnica maturata nel settore dei Beni Culturali campani, che ha fornito un banco prova sufficientemente significativo per verificare le funzionalità del CMS e le basi scientifiche ed informatiche su cui si fonda. Con il progetto ReMuNa (Rete dei Musei Napoletani) – **[REMU01]**, infatti, è stato possibile far cooperare in rete 18 delle più significative istituzioni culturali e museali di Napoli (tra cui *Archivio di Stato di Napoli*, *Museo Archeologico Nazionale di Napoli*, *Museo di Capodimonte*, *Certosa* e *Museo di San Martino*, *Museo Diego Aragona Pignatelli Cortes*, *Museo di Palazzo Reale*, *Pinacoteca del Pio Monte della Misericordia*, *Museo del Tesoro di San Gennaro*, *Parco della Tomba di Virgilio*, *Museo dell'opera di Santa Chiara*) con un'unica piattaforma software e

sotto un unico modello di interoperabilità, così da formare un circuito di siti web ognuno dotato di una propria autonomia di gestione, ma costituenti una rete di conoscenze comuni. Successivamente, i risultati tecnico scientifici del progetto ReMuNa sono stati acquisiti dalla Direzione Regionale per i Beni Culturali e Paesaggistici della Campania, che ha lanciato il progetto *campaniabenculturali.it*, che vede la creazione di una federazione di circuiti – ognuno associato ad una provincia della Campania – che porteranno entro la fine del 2007 e gli inizi del 2008 all’istituzione di un circuito di oltre 100 siti web, in regime di cooperazione. Al momento, sono in pubblicazione più di 50 siti del circuito *napolibenculturali.it*, che cooperano per la creazione di percorsi tematici trasversali alle istituzioni museali: con l’ausilio degli aggregatori e dei contenuti distribuiti, i musei cooperano alla creazione di percorsi “virtuali”, i cui contenuti sono aggregati da una tematica comune. Il circuito *napolibenculturali.it* aderisce inoltre al progetto europeo Michael, iniziativa finanziata dalla Comunità Europea dedicata allo sviluppo di nuove tecnologie per la fruizione di contenuti culturali. Il Ministero per i Beni e le Attività Culturali partecipa a questo progetto, e la versione italiana del portale Michael è disponibile in **[MICH02]**. Questo progetto ha prodotto un documento di specifica dei metadati che si intende utilizzare, ed è consultabile in **[MICH01]**.

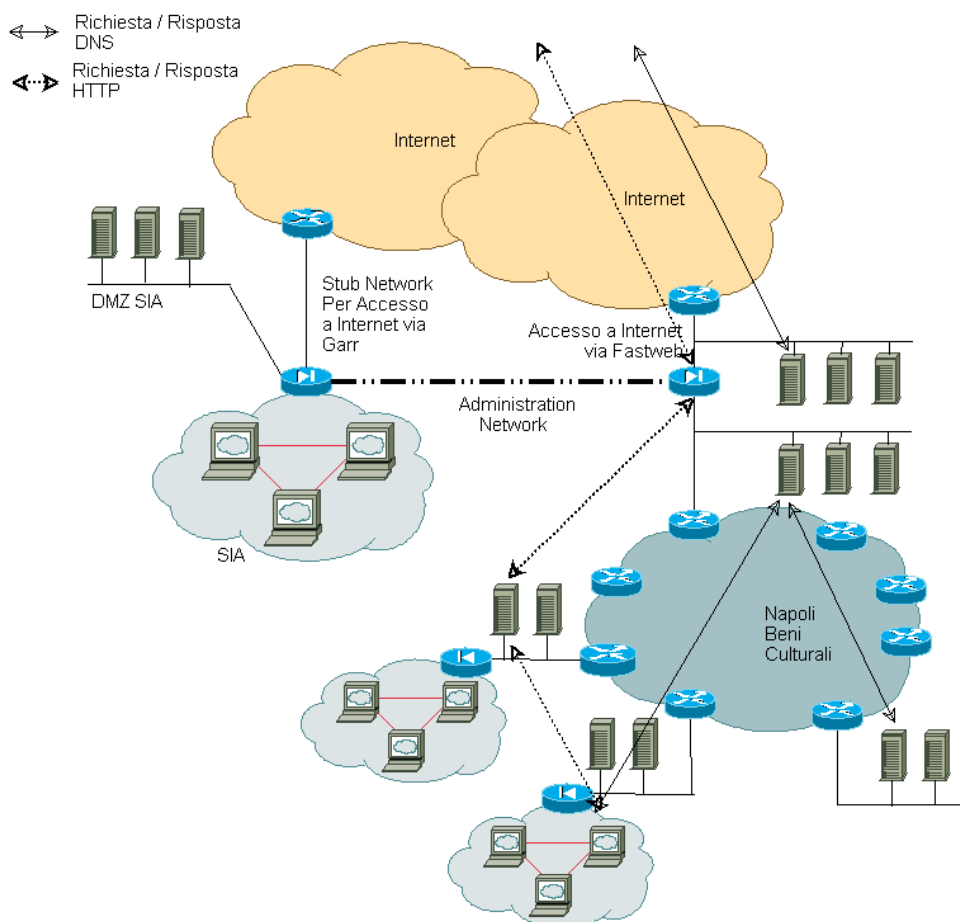


Figura 6 – L’infrastruttura di rete del circuito *napolibenculturali.it*

L’infrastruttura di rete e servizi rappresenta lo strato

operativo dei sistemi distribuiti. L'infrastruttura sulla quale si poggiano i servizi di rete del circuito *napolibeniculturali.it* è una *Virtual Private Network* (VPN) a tecnologia MPLS alla quale hanno accesso le varie sedi attraverso linee a 2Mbps o a 10Mbps, si veda **Figura 6**. Ogni Ente, Soprintendenza, Museo e Istituto aderente al circuito *napolibeniculturali.it* è dotato dell'infrastruttura di rete locale per la connessione dei dispositivi, del punto di accesso alla WAN, di uno o più server per ospitare il sistema di gestione dei contenuti, di sistemi di rete per l'accesso remoto ai server. Per alcune delle loro sedi, è stata integrata la rete locale degli enti con la VPN così che essi possono usufruire dei servizi offerti dal circuito *napolibeniculturali.it* e l'accesso ad internet. Oltre all'accesso a *napolibeniculturali.it*, questi dispositivi consentono l'accesso delle reti locali degli enti ai servizi del Ministero dei Beni Culturali e la gestione di accessi di tipo Intranet tra più sedi distribuite sul territorio ma appartenenti allo stesso ente.

Riferimenti e Bibliografia

- [ACAM07] P. Acampa, C. Noviello, *Specifica OCML*, Technical Sheet Octapy CMS
- [AIEL06] Aiello A., M. Mango Furnari M., Massarotti A., Brandi S., Caputo V., Barone V., *An experimental Ontology Server for an Information Grid environment*, Int. Journal on Parallel Programming, Dec. 2006
- [BORG00] C. L. Borgaman, *From Gutenberg to the global information infrastructure*, MIT Press
- [EDDO98] G. Eddon, H. Eddon, *Distributed COM*, Microsoft Press
- [FURN05] M. Mango Furnari, C. Noviello, *The integration of cartographic information into a Content Management System*, Proceeding of Internet Imaging VII Conference
- [FURN05b] M. Mango Furnari, C. Noviello, *Introduction to Octapy MapServer*, <http://oms.remuna.org>
- [GAMM95] E. Gamma et al., *Design Patterns – Elements of Reusable Object-Oriented Software*, Addison Wesley
- [GLUS05] R. J. Glushko, *Document Engineering*, MIT Press
- [ICCD] *Specifica ICCD*, <http://iccd.beniculturali.it>
- [KIRT99] M. Kirtland, *Designing Component Based Applications*, Microsoft Press
- [MAPS01] *MapServer Project Page*, <http://mapserver.gis.umn.edu/>
- [MICH01] *Progetto MICHAEL*, <http://www.michael-culture.org>
- [MICH02] *Progetto MICHAEL del Ministero per i Beni e le Attività Culturali*, <http://michael.beniculturali.it>
- [NOVI07] C. Noviello, *Il Component-Model di Octapy 3*, Technical Sheet Octapy CMS
- [OAI] *Open Archives Initiative*, <http://www.openarchives.org>
- [OMS] *Octapy MapServer*, <http://oms.remuna.org>
- [REMU01] *Progetto ReMuNa*, <http://www.remuna.org> e <http://www.remuna.org/progetto>
- [VLIS98] J. Vlissides, *Pattern Hatching – Design Patterns Applied*, Addison Wesley
- [WEIT07] P. von Weitershausen, *Web Component Development with Zope 3 - 2nd Edition*, Springer



<http://octapycms.remuna.org>



©Copyright Istituto di Cibernetica "E.Caianiello"
©Consiglio Nazionale delle Ricerche

Octapy, Octapy CMS, OMS, Octapymus sono copyright del CNR
Plone è un copyright della Plone Foundation
Zope è un copyright della Zope Corporation
Tutti i nomi e i marchi citati sono copyright dei rispettivi autori