Model Driven Development of Content Management Applications

Prasad M. Deshpande[+] Brendan McNichols[++] Michael Richmond[+]
Savitha Srinivasan[+] Vladimir Zbarsky[+]
{ prasadd, btmcnich, mrichmon, savitha, vzbarsky}@us.ibm.com

ABSTRACT

Building applications on databases or content management systems involves various stages such as schema design, business logic design and UI design. We explore a model driven design approach, in which the application building process is initiated by building a model. The model includes various aspects such as the data model and business rules definitions. Once the model is defined, other components can be generated automatically from it. These components are then used to build the rest of the application. We have developed an integrated, extensible platform for supporting the different roles in this process: Data Architect, Application Developer & Web Interface Designer. The data architect role is supported by tooling that supports visual repository schema design and data modeling in a disconnected mode. The application developer and UI designer role is supported by tooling that allows business and process logic to be assembled at a high level of abstraction, potentially without the need for specialized coding skills. The challenge addressed by this approach is to bridge the gaps found at the boundaries of development tooling and allow for a smooth transition across the various roles, thus enabling rapid development of content management based enterprise applications.

1. INTRODUCTION

The imperative in business today is to respond to market-place changes in real-time and be agile. Building applications and updating them to reflect business process changes is a complex and time consuming task. A typical project may involve several different developers performing various development roles, which leads to challenges in maintaining consistency across the project. An integrated tooling platform that supports these roles can help to address these challenges by common environment and language for developers. The critical aspects of the application such as the schema definitions, business rules definitions, workflow definitions can be maintained in a common project and used to support the different roles.

The way business applications are developed is evolving. Many business process applications are being developed at higher levels of abstraction with a growing focus on assembling "service-oriented" components and less on code crafting than traditional development. These "service-oriented" components allow business and process logic to be assembled at a higher level, or meta-layer, potentially without the need for specialized coding skills.

This paper presents our model driven development architecture for Content Management (CM) together with the tooling which supports this architecture. We focus on two key roles in this paper: the data Architect and the Web UI designer. For the data architect role, we have built a visual, diagram oriented tool for designing the data model based on the Eclipse Modeling Framework (EMF). For the Web UI Designer role, we provide tooling to compose the UI by putting together UI components in a drag and drop fashion. The tools are implemented as plugins to Eclipse. The extendible plugin architecture of Eclipse makes this platform very powerful and enables building of additional tools that can use the existing plugins. Thus, we can support a model driven design approach so that different components that are dependent on the data model can have their corresponding tooling interface with the data modeling plugin. The infrastructure we describe in this paper is developed for the IBM DB2 Content Manager, though the ideas are applicable to building applications on other data stores as well.

The remainder of section 1 provides an overview of Content Management and the data modeling aspects of a Content Manager application. In section 2 we give a brief overview of the CM data model and the architecture for IBM's Content Manager repository. Section 3 describes the model driven development architecture, highlighting two key development roles that are the focus of our tooling. Section 4 describes our implementation of the MDD framework. Section 5 describes our tooling support for visual, diagram based design of the application data model. Section 6 describes our tool support for web interface design and assembling services-oriented applications. Sections 7 presents earlier related work and Section 8 gives a summary of our work.

1.1 Content Management

Content management (CM) is defined as software that builds, organizes, manages, and stores collections of digital works in any medium or format. It refers to the process of handling various types of structured and unstructured information, including images and documents that may contain billing data, customer service information, or other types of content. It also refers to the

ADVANCES IN DATA MANAGEMENT 2005
Jayant Haritsa, T.M. Vijayaraman (Editors)

[+]IBM Almaden Research Center, 650 Harry Rd
San Jose, CA 95120
[++] IBM Rational, 8383 158th Ave, NE
Redmond, WA 98052

process of capturing, storing, sorting, codifying, integrating, updating and protecting any and all information. Studies estimate that more than 75% of enterprise data is unstructured and document-related [9]. Key technologies in the content management market include traditional Document Management, Web Content Management, Digital Asset Management, and Records Management. Today's users of content management are in document-heavy industries, where document management is essential, often for regulatory or compliance reasons. A real-time enterprise needs content management so it can create, access, and transfer information, as needed, to meet the enterprise's business goals. To summarize, today's content comprises of many different forms of unstructured data that must be managed:

- Dynamic Web content—business data in relational databases and personalized information
- Business documents—ranging from contracts and invoices to forms and e-mail that facilitate internal back-office processes and enable direct external communication with customers, partners, and suppliers
- Rich media—such as digital audio and video which is rapidly transforming areas of training, education, marketing and customer relationship management in many industries
- Records Management -- is being driven by government and industry regulations to effectively document processes, audit trails and data retention
- Team Collaboration Content - Web collaboration sessions or threads, Webcast content, and instant messages that are rapidly becoming an important information asset

While outwardly dissimilar, all of these forms of enterprise content have similar management needs. To be truly useful, a content management solution must address requirements for mass storage, search and access, personalization, integration with business applications, access and version control and rapid delivery over the Internet. From a data point of view, the key difference from traditional relational databases is the support for unstructured content. Content management systems manage data that has both structured (attributes) and unstructured components (content), and maintain the association between them.

1.2 CM Application Development

A web based content management application is comprised of three distinct elements:

- Backend Data Modeling
- Mid tier application logic (JSP/Servlets or EJBs), and
- Web based UI frameworks.

A CM application can have very complex data modeling requirements. These requirements include support for different types of content and their relationships, such as links and folders. This situation can quickly lead to a system development process that is difficult to manage. Customers frequently express a need for tools that make it easier to model their data and build custom CM applications. Once the schema is designed, it needs to be shared with other application builders who will code the mid-tier logic and the UI. Thus documentation of the schema becomes an important consideration. The mid-tier application logic encodes the business logic for the application. This includes the process for create, update and deletion of documents, search and retrieval of the documents and workflow for routing the documents. The UI is the presentation layer using which users interact with the system.

2. DB2 CONTENT MANAGER

In this section, we will give a brief introduction to the IBM DB2 Content Manager on which our solution is based.

2.1 DB2 Content Manager Architecture

IBM's content management technology directly leverages the DB2 Universal Database to store "structured" data—which fits into the columns and rows of traditional databases and complements that with supporting repositories which manage "unstructured" data or content. DB2 Content Manager uses a triangular architecture, to offer functional advantages. Client applications (running either in end-user desktops or mid-tier application servers) use a single object-oriented API to invoke all DB2 Content Manager services which are divided across a single library server and one or more resource managers. The library server manages the content meta-data and is responsible for access control to all of the content, interfacing with one or more resource managers. Resource managers manage and store the individual content documents such as a pdf document, image or video file. Both the library server and resource manager may utilize the Lightweight Directory Access Protocol (LDAP) service for user management and access control. All access to the library server is handled via the database query language SQL, due to the library server code being co-resident with the database engine code. Query results returned to the client from DB2 Content Manager include object tokens that act as locators for requested content that the user is authorized to access. Using these locators, the client can communicate directly with the resource manager using HTTP or FTP to retrieve the content documents.

This decoupling of meta-data management and access control from content management and delivery offers a number of important advantages; including performance, scalability, exploitation of database capabilities for meta-data management and use of high-speed file system access for content documents.

2.2 CM Data Model

Data modeling capabilities are essential to enterprise content management. We will not describe all the data model elements in this paper, but rather focus on some important ones. The primary building blocks of the DB2 Content Manager data model are itemtypes and attributes. In database terminology, an itemtype corresponds to a table or relation and an attribute corresponds to a table attribute. However, unlike a flat relational table, an itemtype can have a hierarchical tree like structure much like an object in object-oriented languages. Attributes for an itemtype can be structured with parent and child relationships that match the hierarchical structure found in real-world customer application environments. This feature is used to modeling repeating groups in which multiple instances or values of attributes may be present. For example, a customer insurance policy could have multiple operators and multiple vehicles to be insured. The root node is called the root component and the inner nodes are called child components. Itemtypes also capture information regarding versioning policy and retention period for the items.

Itemtypes could be of different types. For example, a document part itemtype represents a part of a document. It has attributes and an associated content that can be of different types. There are some system defined document parts that represent annotations, notelogs etc. A document itemtype represents an entire document. It also has some attributes and is associated with a set of document part itemtypes that represent parts of that document. In DB2 CM, attributes are first class objects. Thus, unlike in relational systems, attributes can exist by themselves and need not belong to any particular item type. Groups of attributes that are used together can be defined as attribute groups for convenience. CM data model design phase consists of defining the different attributes, attribute groups, itemtypes, document itemtypes, etc. that the application will use. Application data consists of instances of these itemtypes, called 'items'. The flexibility of the CM meta-data model allows the creation of complex itemtypes that represent real world entities.

DB2 Content Manager allows custom applications to build more complex inter-item peer-to-peer relationships using links, references and foreign keys. Links are many-to-many typically used to associate arbitrary external relationships between any two items. Links can be used to implement the concept of folders by linking the folder item to the items that it contains. References are a way to represent a pointer from any component in an item hierarchy to any item of any type in the system. Applications can also define attributes as foreign keys to external DB2 Universal Database tables that are not part of the DB2 Content Manager schema.

2.3 CM Application Architecture

A typical CM Application adopts the classic 3-tier architecture consisting of:

1. backend CM Server,
2. mid-tier application logic and presentation handling,
3. thin web-based clients.

The backend CM Server is generally made up of a cluster of CM library servers together with one or more CM resource managers as detailed in Section 2.1. The mid-tier interfaces with the backend server via proprietary Java APIs that export the necessary functionality to establish a connection with the server, perform CRUD (Create, Retrieve, Update, Delete) operations over the data, and execute server administration tasks.

The CM APIs are designed to expose all of the CM functionality to the developer. The drawback is that since CM servers are complex systems this complexity results in a complex API. The complexity is required if developers are to have full access to CM functionality, but it hinders development of the average application.

The mid-tier is primarily concerned with hosting and executing the raw business logic of the application. This logic is typically implemented using a collection of Java classes which are triggered by hand coded JSP and/or Servlet-based presentation layer.

With a thin client model, the mid-tier is also responsible for generating the instructions that are interpreted by web-based clients to provide the presentation layer. Since both the business logic and much of the presentation layer is concentrated at the mid-tier, current practice often results in the merging of these elements. The net result is a monolithic mid-tier application which is difficult to maintain, debug or extend to new presentation technologies.

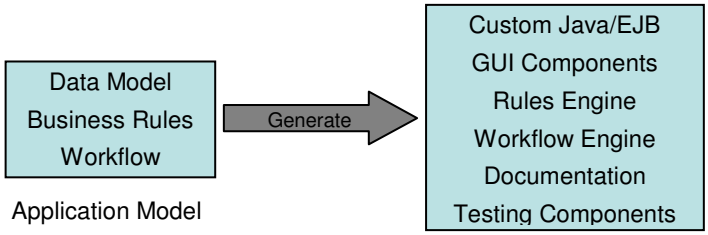

Figure 1. Model Driven Development

Also, the overriding difficulty is that each of the technologies being used operates at fairly low levels of abstraction. This requires the application developer to be an expert in the CM APIs, JavaBeans, along with any supported presentation technology such as JSP/Servlets, struts, portlets, etc.

3. MODEL DRIVEN DEVELOPMENT ARCHITECTURE

Development approaches for business applications is shifting away from the traditional reliance on custom code developed in-house. Instead, business process applications are being constructed at higher levels of abstraction with a heavy focus on assembling "service-oriented" components that provide individual units of application functionality. These components allow business and process logic to be assembled at a higher level than traditional application development.

Each service-oriented component provides a well-defined set of services to other components in the final application and may rely on services provided by other components. In this scenario, the task of building applications becomes that of collecting the components which provide the desired services and specifying the interaction between these components. Apart from the resulting benefits related to code-reuse, the higher level of abstraction afforded by services-oriented development reduces the need for specialized development skills for application development.

These components can be characterized by their coupling with the underlying application model as:

1. model dependent, or
2. model independent.

The model independent components may be developed once and made available to the component integrator as a static library of components. On the other hand, model dependent components need to be developed after the application model has been defined. In the model driven development (MDD) approach, the application model forms the basis for all other development. Rather than require re-development of components each time an application model is defined, or refined, the tooling environment can play a critical role by automating generation of some of these components. Figure 1 shows the model driven development process. The application model comprises of various aspects such as the data model, business rules specification and workflow specification. Based on the application model, various components relevant to different parts of the application can be generated automatically, such as:

1. Higher level EJB and Java components. Rather than working with the generic low level CM API, we can have customized java classes for each itemtype. For example, we can generate a class called InsuranceForm to represent an insurance form. These components provide a simpler implementation compared to the generic API, since there is no need for the application code to explicitly interpret the data model.
2. GUI components such as search panels and results display that depend on the definition of the itemtypes
3. Business rules engine. Based on the business rules specification we can generate the runtime rules engine components. Business rules are used for various reasons such as data validation, and runtime actions based on certain conditions.
4. Workflow engine. The workflow specification can be used to generate the required code for integrating the application with a workflow engine.
5. Documentation. System documentation is an important requirement for sharing information among people in different roles. A major part of documentation could be generated automatically from the data model specification.
6. Testing components. The data model definition can be used to generate test components for automatically generating test data and maybe some basic test cases.

We have built an integrated framework to support this process, with tooling support for the different phases. In terms of the application model, the focus of our initial work has been on application data modeling. Support for business rules and workflow specifications are part of our ongoing work. To illustrate the MDD concept, we will focus on the UI building aspect of the post modeling development process. Since different people might be involved in different aspects of the development process, it is useful to identify the various roles so as to provide the appropriate tooling support. For example, one can consider the role of the data architect as being separate from the role of component developer, UI developer, integrator or tester. In our current work, we have primarily focused on two specific development roles:

- the Data Architect role, and
- the UI Architect role.

3.1 Data Architect Role

Applications provide a way for users to work with data that is stored persistently either in a database or content management system. The structure of this data depends on the application being developed. For example, an insurance application will have data corresponding to insurance policies, claims and the people insured.

The data architect role is concerned with defining the different data elements to be used in the application. The collection of these elements referred to as the application data model. Data model design is a critical step in the application development process, since this model is the basis for all persistent data storage and access. The data architect takes into account the different objects, their attributes and relationships between objects. These are then mapped onto the data model provided by the backend store. It could be a relational model in the case of a database system or a richer hierarchical model in the case of a content management system. Thus, in the context of the DB2 Content Manager, this process consists of defining the different attributes, itemtypes, documents, etc. that will be used to represent application data. The data architect ideally has expertise on the capabilities and limitations of the data store so that they can determine an optimal way of representing the data on the data store.

The data architect role relies on skills which are generally held by a database administrator rather than those of an application developer. As such, it is useful to separate this role from the application developer, since an application developer needs to be concerned with the business logic without having to worry about how to efficiently represent the data.

3.2 UI Architect Role

The UI Architect role is concerned with integrating pre-existing graphical interface components and service-oriented mid-tier components to construct a functional CM application. This involves identifying the components which provide the functionality needed in the application and establishing links between these components to provide data passing and interaction between them.

The service-oriented components used to assemble an application may originate from one of three sources:

1. commercial component libraries,
2. in-house development of custom business components, or
3. integration tool generated components.

The last category refers to the components generated by the model driven development process. The development tooling such as the data modeling tool or application assembly tool can provide wizards and other user-interaction approaches to generate these components. For example, based on the user selecting a required itemtype from a CM server, the component integration tool can generate an appropriate service component to access the itemtype data in the application and the corresponding UI components for searching and displaying results of that itemtype.

Each of these categories of service-oriented components together with libraries of graphical interface components allows business and process logic to be assembled at a higher level, or meta-layer to form an application. A lot of time and effort can be saved by automating the generation of these components.

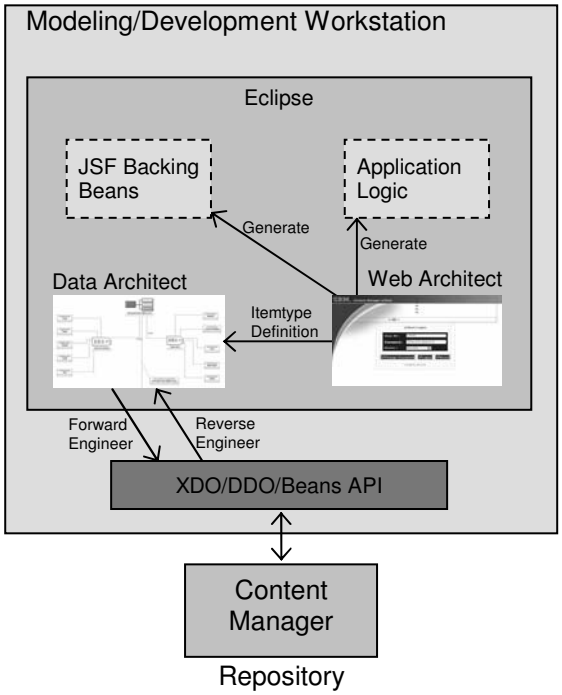

Figure 2 CM Architect Architecture

4. SYSTEM IMPLEMENTATION

We have developed a MDD framework for supporting the CM application building process. Since extensibility of the framework is a prime consideration, we based it on the open Eclipse platform. The tools are implemented as a feature (set of plugins) for Eclipse. These plugins extend the eclipse environment with CM specific views and functionality. We refer to this feature as the CM Architect Workbench. Figure 2 shows the architecture of the CM Architect Workbench including the major interfaces between individual tools and backend CM repository.

Currently, the CM Architect Workbench tooling is comprised of a data architect tool and a web architect tool. The data architect tool is designed to support the data architect role as described in Section 3.1. Whereas the web application architect tool supports the UI architect role described in section 3.2.

The plugin architecture of Eclipse is very flexible. The CM Architect can be easily extended by developing additional plugins. These plugins can depend on and leverage the functionality of existing plugins. Thus, in future we can add plugns to model business rules and workflow specifications or to implement additional MDD features such as EJB code generation, documentation generation, test case generation and so on.

5. CM DATA ARCHITECT

Current CM tools for schema definition are tightly coupled with the CM server and directly manipulate the data definitions on the server. This coupling limits the scope of "what-if" exploration, common in iterative development, that a developer can perform. In addition, this lack of a model-driven approach does not allow integration with other development tools such as service component generation, documentation generation, or interface construction tools. Our approach separates the modeling process from the deployment process to facilitate distributed, easy-to-use development tools. The CM Data Architect aims to make it more intuitive for the user to create a application data model by visually representing the system data model as a set of diagrams. The data model is then used to build service oriented-components that can be integrated to develop the application.

5.1 Eclipse Modeling Framework

The modeling tool is based on the Eclipse Modeling Framework (EMF). The EMF unifies Java, XML, and UML technologies so that they can be used together to build better integrated software tools. EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. Models can be specified using annotated Java, XML documents, or modeling tools like Rational Rose, then imported into EMF.

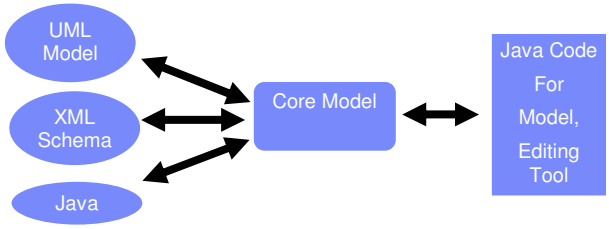

Figure 3. Eclipse Modeling Frameweork

As shown in Figure 3, the process begins by defining the EMF model in UML, XML or annotated java. The EMF code generation facility then generates the following:

1. **Model** - provides Java interfaces and implementation classes for all the classes in the model, plus a factory and package (meta data) implementation class.
2. **Adapters** - generates implementation classes (called ItemProviders) that adapt the model classes for editing and display.
3. **Editor** - produces a properly structured editor that conforms to the recommended style for Eclipse EMF model editors and serves as a starting point from which to start customizing.

We used UML to define a model for content management. A subset of the model is shown in Figure 4. This EMF model captures the semantics of DB2 Content Manager. For example, itemtypes are represented by the CMItemType element. CMItemType comprises of an aggregation of CMAttributeMappings, CMAttributeGroupMappings and CMChildComponents. CMChildComponents themselves contain CMAttributeMappings and other CMChildComponents. This captures the hierarchical structure of root and child components. CMItemType can be specialized into CMDocumentItemType, CMDocumentPartItemType or CMResourceItemType. The EMF code generation framework is used to generate the corresponding

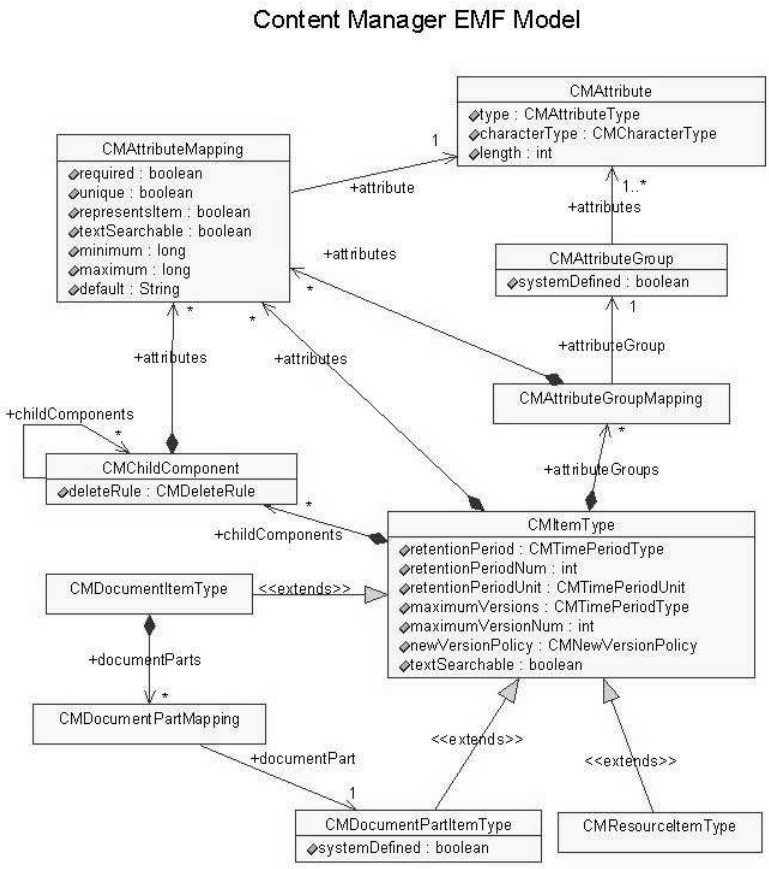

Figure 4

Java classes and a default editor from this model. These classes provide the meta-model for building Content Management data models. For example, the meta-model contains a class to represent a CM itemtype and class to represent a CM Attribute. Similarly there are classes corresponding to other CM building blocks. The generated editor provides the functionality for editing instances of this model, i.e. for editing application data models. Thus the user can define different attributes and itemtypes that will be used by the application. We customized the default editor generated to give a user friendly look and feel.

5.2 Visual Modeling

It is more intuitive to work with data models in a visual fashion by drawing diagrams. The diagrams depict the different model elements and the relationships between. Diagrams make it easy to share knowledge with other developer and also serve as documentation for the design. The data architect tool provides support for creating UML diagrams for the data model. The model visualizer is based on the Graphical Editing Framework (GEF) for Eclipse. The GEF allows developers to create a rich graphical editor from an existing application model. GEF uses the SWT-based drawing plugin Draw2d to create a graphical environment within Eclipse. Tool developers can then take advantage of the many common operations provided in GEF and/or extend them for the specific domain. Figure 5 shows an example of a diagram for an insurance data model. The model has a document item type called 'Policy' with two child components 'Operators' and 'Insured_Cars'. It has one document part item type called 'Policy_base'.

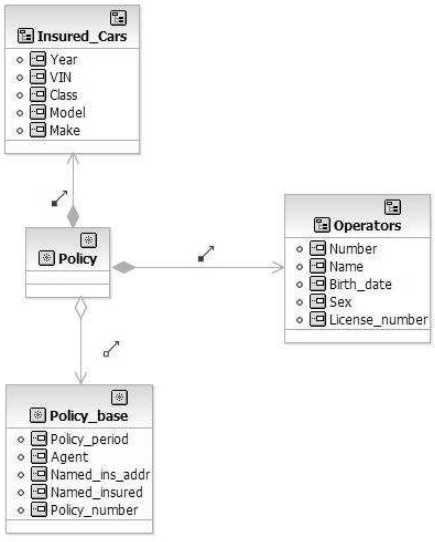

Figure 5. Insurance Data Model

5.3 Decoupled Model Development

The EMF framework provides a persistence mechanism that allows the model instance to be stored as a XML file. This allows the user to work with models without it being actually deployed on any CM system. Thus, unlike the current tools for defining data models, the architect workbench is not tightly coupled with any CM server and allows the user to work in an offline mode. Making model definition independent of the backend system has the following advantages:

1. The user can work on a data model iteratively without affecting the backend system.
2. It allows for easier integration with other tools such as the web application architect. Those tools can read the data model from the modeling tool rather than having to connect to the backend CM system.
3. It facilitates MDD features such as automatic generation of data-model dependent service components
4. It allows domain specific reference models to distributed easily with the product as xml files
5. It allows easier migration of data models from one server to another

The activity diagram shown in Figure 6 summarizes the sequence of actions supported by the CM Data Modeling tool for model driven application development.

5.4 Forward/Reverse Engineering

where the user loads the existing model elements from the CM server into a visual model. The forward and reverse engineering feature is implemented as part of the CM plugin The forward and reverse engineering code interfaces with the CM system using the Java client API to CM (called XDO/DDO). The forward engineering traverses the CM data model and for each CM element it takes an appropriate action using the XDO/DDO interface. For example, if a CMAttribute "Address" is defined in the model, it will create a corresponding Attribute definition in the CM system. Reverse engineering is a similar process where we read the CM model elements from the backend system using the XDO/DDO API and populate the EMF based model with the corresponding elements. Forward engineering can cause conflicts since the CM system might already have some model elements defined earlier. The tool has a conflict resolution mechanism where it compares the user defined model with the model already on the server and displays conflicts to the user. The user can then decide whether to go ahead with the change or to roll it back. This is conceptually similar to a merge operation.

6. WEB APPLICATION ARCHITECT

To illustrate the usefulness of MDD, we have developed a GUI-focused component integration tool to support the UI Architect role described in section 4. This tool is focused on the needs of component integrator role and allows the user to assemble applications using the drag and drop metaphor.

Our Web Application Architect tool is also implemented as a

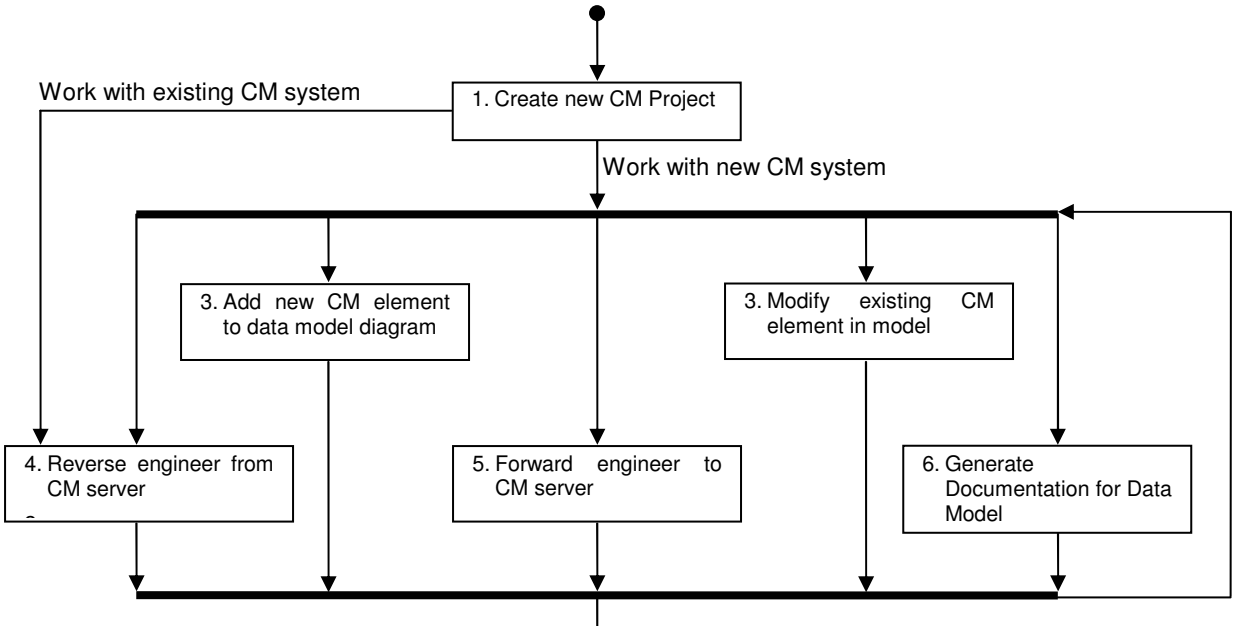

Figure 6. Activity Diagram detailing CM Data Modeling Functionality

Even though the bulk of model development is in a decoupled mode, we need to provide a way for users to actually deploy their model onto a CM system and to work with model elements already existing on a CM system. Our tool provides this functionality by giving the user the choice to synchronize with the CM system at any point through forward and reverse engineering. Forward engineering persists the model defined by the user onto a CM server. Reverse engineering is the complementary action

plug-in to Eclipse which can be used in parallel to our Data Architect tool described earlier. The supporting business component libraries are based on the JavaServer Faces (JSF) component framework. All business components generated by our tool during application assembly also conform to the JSF specification [10]. The JSF framework builds on Java Server Pages (JSP) technology and allows components to access the common data environment provided by the JSP architecture. This

environment is unique for each client and provides several separate name spaces which are scoped according to the lifetime of the namespace.

A typical CM application is based around a series of screens each of which contains a number of panels which provide discrete units of generic application functionality. For example, a logon panel is a unit which is comprised of text fields to collect the user name, password and server name together with the appropriate labels and control buttons such as 'Logon' and 'Reset'.

These functional units, commonly referred to as business units or business components, are comprised of a GUI element and some form of backing logic to provide the actions provided by the business component. In the above example of a logon panel, the backing logic would provide the methods executed when either of the buttons is clicked. For the logon button, the action would authenticate the details provided by the user and establish an authenticated server connection which can be passed to other business components.

Our efforts to support component based development for CM have lead to the development of a library of business components which can be integrated to form a CM web-based application. As described in Section 3, we have categorized this library into model-independent generic components and model-dependent specialized components.

Generic components mirror the traditional code library in that they are written and compiled ahead of time and then used as black box components during component integration. This use-case generally favors components which are data model agnostic and are capable of adapting their behavior at runtime to match the underlying data model.

In contrast, specialized components are tightly bound to a specified data model or model element. These include java classes and UI components specialized to an itemtype. Rather than having to implement them for each new data model, they are generated during component integration in our MDD framework. The generated code can be further specialized by the user if necessary.

6.1 Application Construction

To construct a CM application, the component integrator uses the web application architect to perform the following series of steps:

1. Create a new CM Application project,
2. Add page to project,
3. populate page with required business components, (repeat steps 2 & 3 as required)
4. Specify page flow using action triggered navigation rules,
5. Deploy application to test application server.

The focus of our tooling support is around steps 3 and 4 above, and shown in Figure 7.

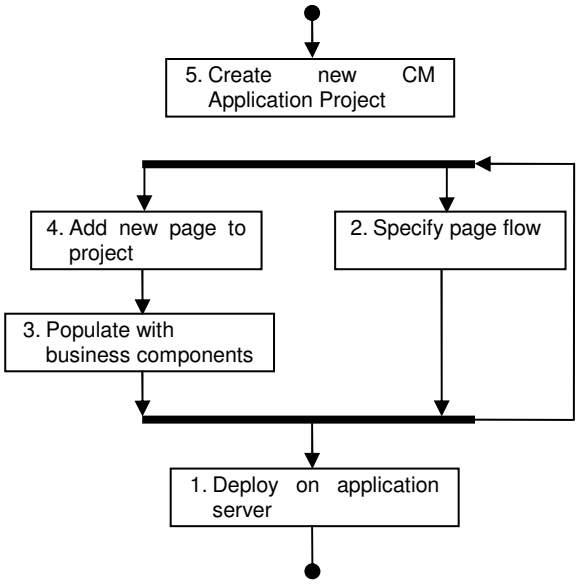

Figure 7 Web Application Architect activity diagram

Our GUI Construction tool introduces a palette of CM specific business components. This palette contains all of the pre-built components in our generic component library together with a button for each type of specialized component we support. This palette is shown in Figure 8.

After adding a new page to a project, the user drags the desired business components from the "CM JSF Components" palette onto the application page. In the case of generic components such as a 'Logon' or 'Logoff Button' component this action inserts the appropriate JSF tags into the page and generates managed beans to provide the necessary action logic.

If the component dragged onto the page is a specialized component the user is presented with a list of appropriate data elements from the Data Architect tool. For example, when the user drags a 'Search Panel' onto the page a dialog listing the itemtypes available in the CM Data Architect tool (or a CM server) is shown. After an itemtype has been selected the Web Application Architect tool interacts with the Data Architect to retrieve the definition for the selected itemtype. This definition is used to populate the UI search panel with the names of searchable attributes, data format hints and a text box to input a constraint over each searchable attribute. Additionally, dropping the panel onto the page triggers the code generation for the backing JavaBeans which implement the behavior of the search panel and adds them to the project.

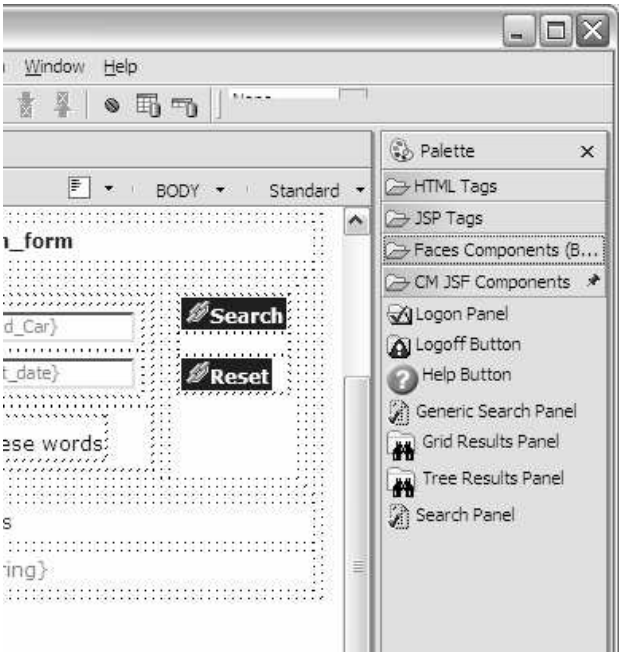

Figure 8 CM Web Application Architect tool palette

7. RELATED WORK

The Entity-Relationship (ER) model, originally proposed by Peter Chen in 1976 [7], is a conceptual data model that views the real world as entities and relationships. A basic component of the model is the Entity-Relationship diagram which is used to visually represent data objects. The ER model maps well to a relational model and is widely used for database design.

The need for easier user interaction with databases has been an active research concern since graphical user interfaces (GUI) first became widely available in the mid 1980s. Modern databases provide both command-line tools and GUI-based tools to manipulate schemas perform system administration tasks, express queries and display query results. The GUI-based tools typically rely on tree and table structures to provide the metaphors presented in the user interface. The limitation of these representations is that the user is still required to perform a mental mapping between the tree/table view of the database structure and the abstract conceptual model of the domain represented by the database. Also, most of these tools don't support the MDD feature of using the schema definition to generate additional components that can be used in application building.

Much of the research related to easing user-database interactions is focused on runtime aspects such as query expression [2][3], query result display [16] and navigation through the stored data. Collectively these tasks are referred to as Visual Query Systems (VQS) [6]. In comparison, relatively little focus has been placed on the interface provided by the tools used to define and manipulate data models and database schemas. Commercial database modeling products such as Rational XDE provide visual data modeling profiles which integrate into the broader software development cycle [8],[9]. These profiles are generally geared to UML modeling of relational databases. The OPOSSUM system, developed at the University of Wisconsin, Madison, allows a database schema to be edited through manipulation of the schemas visualization [10]. Haber et al. report that "schema visualization is the key issue in any attempt to improve schema management" with diagrammatic presentations being generally easier to understand for both beginning and advanced users [11].

In our data modeling tool, the user directly manipulates the data model elements to model the required domain. We claim that this approach to data modeling capitalizes on the benefits of general direct manipulation interfaces – specifically, ease of use and learning together with a reduction in the required mental mapping between the on screen representation of the data model and the abstract conceptual model of the domain [13].

Existing work related to user interface construction focuses on providing drag-and-drop tooling to layout individual interface widgets. Once the interface design has been completed with these tools, code which will produce the designed layout is generated. At this stage, with current tools, the developer then establishes hooks between the generated interface code and the corresponding business logic code. This model of interface construction is exhibited by XForms [17], NetBeans Form Editor [5], and Microsoft Visual Basic [12].

In our web application construction tool, the user manipulates user interface components that are comprised of both the graphical interface elements and the corresponding business logic for the component. These interface components are assembled with mid-tier service components by the user to assemble a complete application without the need to directly alter program code. As such, our approach is more closely related to web page construction tools such as Adobe GoLive and Microsoft FrontPage which allow the user to drag-and-drop page elements such as scrolling marques which are backed by business logic written in JavaScript.

8. SUMMARY

We have presented a Model Driven Development Architecture for Content Management supporting several roles in the application development lifecycle. This architecture supports roles such as the Data Architect and UI Architect. Our challenge has been to develop the relevant intelligence at the boundaries of these roles allowing for a smooth transition from one role into another within the same development environment. In this context - we have described our tooling technologies developed to support the Data Architect & UI Architect roles.

Key advantages of this framework are:

- data model tooling decouples the design phase from the deployed system. This enables the model to evolve iteratively before committing to the deployed system. This is a critical requirement as today's process typically involves paper-based design of data models without the ability to use tools to collaboratively evolve the design.
- data modeling tool gives the ability to deliver out of the box domain specific models that encapsulate best practices in the specific domain - this becomes an important leverage for application developers in terms of reducing startup time.
- data modeling tool creates the path for effectively documenting the design and maintaining it as the design evolves by coupling the data modeling environment with web publishing tools.

- component integration tool gives the ability to construct an application using libraries of generic and generated itemtype specific components to quickly produce a running application.
- drag and drop component integration allows domain experts to be involved in implementing CM application implementation without the need for programming experience.

A visual development environment, rather than the tree-based text view and APIs provided by current tools, makes development of applications much more intuitive and easy-to-use. Automatic generation of higher level components simplifies application development and reduces the development cycle for CM applications. As the tooling develops with support for business rules and workflows, it will be closer to the goal of rapid application development. Initial feedback from CM customers and consultants on the value of this development environment for CM applications has been extremely positive.

9. REFERENCES

[1] Agrawal, R., Gehant, N. H. and Srinivasan, J., OdeView: The Graphical Interface to Ode. In *Proceedings of the ACM SIGMOD '90,* 34-43, Atlantic City, 1990.

[2] Andries M. and Engels, G., A hybrid query language for the extended entity relationship model. In *Journal of Visual Languages and Computing*, 8(1), 1997, Special Issue on Visual Query Systems.

[3] Angelaccio, M., Catarci, T. & Santucci, G., QBD*: A Fully Visual Query System. *Journal on Visual Languages and Computing*, 1(2), 255-273, 1990.

[4] Binder, R., *Testing Object-Oriented Systems*, Addison-Wesley:Reading, MA, 1999.

[5] Boudreau, T., Glick, J., Greene, S., Woehr, J., and Spurlin, V., *NetBeans: The Definitive Guide*, O'Reilly and Associates, Sebastopol, CA, 2002.

[6] Catarci, T., Costabile, M.F., Levialdi, S. and Batini, C., *Visual Query Systems for Databases: A Survey*. Technical Report SI/RR-95/17, Dipartimento di Scienze dell'Informazione, Universita' di Roma "La Sapienza", 1995.

[7] Chen, P. P., Entity-Relationship Model: Towards a Unified View of Data, *ACM Transactions on Database Systems*, 36(9), 1976.

[8] Gornik, D., *UML Data Modeling Profile*. IBM Rational Software Whitepaper TP 162 05/02, 2003.

[9] Gornik, D., *Data Modeling for Data Warehouses*. IBM Rational Software Whitepaper TP 161 05/02, 2002

[10] Haber, E. M., Ioannidis, Y. E. and Livny, M., OPOSSUM: A Flexible Schema Visualization and Editing Tool. In *Proceedings of the 1994 ACM CHI Conference*, Boston, MA, April 1994.

[11] Haber, E. M., Ioannidis, Y. E. and Livny, M., Opossum: Desk-Top Schema Management through Customizable Visualization. In *Proceedings of the 21st International VLDB Conference*, pages 527--538, Zurich, Switzerland, September 1995.

[12] Holzner, S., *Advanced Visual Basic 4.0 Programming*, M & T Books, 1996.

[13] Hutchins, E. L., Hollan, J. D. and Norman, D. A., Direct Manipulation Interfaces. In Norman D. A. & Draper S. W. (Eds.) *User Centered System Design: New Perspectives in Human-Computer Interaction*. Lawrence Erlbaum Associates: Hillsdale, NJ, 1986.

[14] Lyman, Peter and Varian, H. R., *How Much Information*, 2000. Retrieved from http://www.sims.berkeley.edu/how-much-info

[15] McClanahan, C. and Burns, E. (Eds), *JavaServer Faces Specification: Version 1.0*, Sun Microsystems, Santa Clara, California, February 2004.

[16] Olston, C., Woodruff, A., Aiken, A., Chu, M., Ercegovac, V., Lin, M., Spalding, M. and Stonebraker, M., DataSplash. In *Proceedings of the ACM SIGMOD* '98, Seattle, WA, June 1998.

[17] Zhao, T. C. and Overmars, M., *Forms Library: A graphical user interface toolkit for X*, April 1996. Retrieved from http://www.york.ac.uk/services/cserv/sw/graphics/xforms/forms.index.html