Model-Driven Development and Assembly of Content Management Applications

Michael Richmond[†] Prasad Deshpande[†] Brendan McNichols[‡]

Savitha Srinivasan[†] Vladimir Zbarsky[†]

{mrichmon, prasadd, btmcnich, savita, vzbarsky}@us.ibm.com

ABSTRACT

The use of service-oriented development architectures can speed up business application development allowing market changes to quickly be reflected in enterprise software. We have developed a novel service-oriented development architecture and associated tooling to support on model-driven development of content management based applications.

This architecture supports the set of roles necessary for content management: Data Architect and Component Integrator. The data architect role is concerned with capturing domain knowledge using tooling that supports visual repository schema design and data modeling in a disconnected mode. The Component Integrator role assembles business and process logic 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 involved in the 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. Managers cannot afford to postpone business process changes until the IT department or outside contractors can reflect changes in their software systems.

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 tooling for Content Management (CM) which supports model-driven devevelopment (MDD) and generation of service-oriented components based on domain models. 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 services-oriented application development, highlighting two key development roles that are the focus of our tooling. Section 4 describes our tooling support for Model-Driven Design of an application data model. Section 5 describes our tool support for web interface design and assembling services-oriented applications. In Section 6 we outline earlier related work. Finally, in Section 7 we summarize 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 var-

[†]IBM Almaden Research Center, 650 Harry Road, San José, CA 95120 [‡]IBM Rational, 8383 158th Ave NE, Redmond, WA 98052.

ious 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 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. In addition, there is an increasing trend towards content management for collaborative applications, including streaming media, video conferencing meetings or presentations, Web collaboration sessions or threads, Webcast content, and instant messages. 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.

1.2 CM Application Architecture

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. In this paper, we address this requirement by describing a solution to make it more intuitive for the user to create a CM system by visually representing the system data model as a set of diagrams. The data model is then used to generate service-oriented components that are integrated to construct the application.

Current CM tools 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 component gener-

ation, documentation generation, or interface construction tools. Our approach separates the modeling process from the deployment process to facilitate distributed, easy-to-use development tools.

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. 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 structure much like an object in object-oriented languages.

An itemtype typically represents a single content document and has a set of user-defined index attributes. Each itemtype can contain one or more objects that correspond to actual document content, annotations or notes. 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 claims and each claim may contain multiple supporting documents. The CM repository stores an instance of defined itemtype for each stored content document. These instances are known as 'items'.

The flexibility of the CM meta-data model allows the creation of itemtypes that combine attributes from different business processes (perhaps health insurance and life insurance which are legacies of separate, acquired divisions) centralizing information from both as busi-

ness and regulatory needs dictate. Different views of this information can represent a combined electronic customer folder. An example might be an insurance company total customer view, or a health-only folder.

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. 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 Development

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. This server interfaces with the mid-tier business logic 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 perform 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.

Applications developed today to use Content Management services are typically hand crafted amalgamations of a range of technologies. This is particularly endemic since requirements for CM applications often require the application to be available in several competing and overlapping presentation technologies and integrate with other enterprise resources such as legacy applications and data stores.

The overriding difficulty is that each of the technologies being used operate 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 SERVICE-ORIENTED COMPONENT DEVELOPMENT

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. This provides managers and other process experts the ability to implement software changes quickly as they are less reliant on the skills of IT specialists.

When discussing services-oriented development it is useful to refer to the various roles involved in the development lifecycle. For example, one can consider the role of component developer as being separate from the role of component integrator whereas these roles are combined in traditional application development. It is this separation of component/service implementer from that of the application implementer which allows applications to be built at a high level of abstraction.

In our work, we are primarily focusing on two specific development roles:

- the Data Architect role, and
- the Component Integrator 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 domain data elements to be used in the application. These elements are then assembled to form the application data model. Since this model is the basis for all persistent data storage and access its design is a critical step in the application development process. 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 the 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 and bases much of the data model design on elements found in the domain being modelled. The concerns of the data architect role are focused on capturing an accurate model of the application domain. As such, it is useful to separate this role from those roles explicitly involved in application development. An application developer needs to be concerned with the business logic without having to worry about how to store or represent the domain data.

The application components supported by our development model can be characterized by their coupling with the underlying CM data model as:

1 data model dependent, or
2 data model independent.

The data-independent components may be developed once and made available to the compo-

nent integrator as a static library of components. Whereas the data model dependent components need to be developed after the data architect has defined the data model.

Rather than require re-development of components each time a data model is defined, or refined, our tooling automates the generation of the appropriate service-oriented components. These components are generated during component integration based on the data elements defined by the data architect.

3.2 Component Integrator Role

The component integrator 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 advent and on-going standardization of component frameworks together with the growing acceptance of services-oriented development is opening a market for commercially developed libraries of components. These libraries focus on the needs of a given industry and provide components which implement common, well defined processes in the industry.

For example, a vendor may offer a component library which meets the needs of the airline industry. Such a library would provide components which handle ticket issuing, baggage handling, and seat assignment. Ideally, these libraries would implement industry agreed upon standards such as the OMG Domain Specifications.

Where commercial libraries are not available or do not fully meet the needs of the business it may be necessary to develop custom service components in-house. These components are then used to build an internal library of services that are focused on processes used within the organization.

Finally, the development tooling such as the data modeling tool or application assembly tool can provide wizards and other user-interaction approaches to generate additional 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.

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. This allows the application assembler to focus on the overall business process being implemented and the interaction of services at a high-level rather than delving into the implementation details for each of these services. This focus is on "what needs to be done" by the application rather than the details of "how it is done" allows changes to be rapidly effected in business applications to reflect changes in business processes.

4 CM ARCHITECT WORKBENCH

We have developed a feature for WebSphere Studio Application Developer (WSAD) which extends the environment with CM specific functionality. This feature is known as the CM Architect Workbench. Figure 1 shows the architecture of the CM Architect Workbench

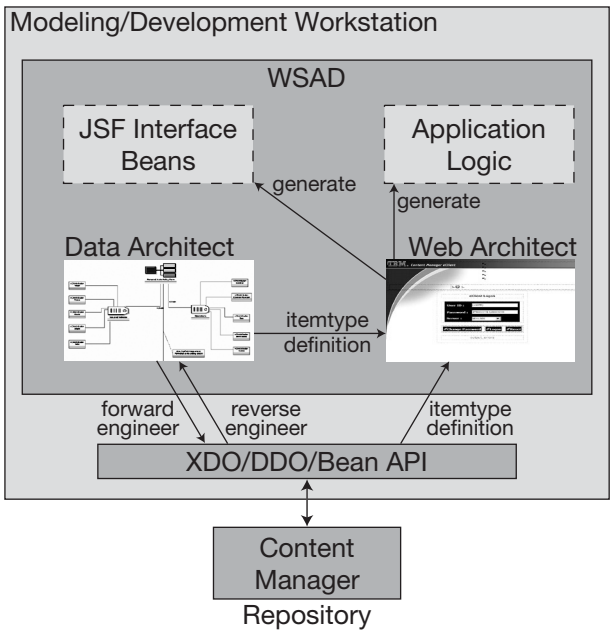

Figure 1 CM Architect architecture

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 component integration tool. The data architect tool is designed to support the capture of domain-specific data models by the data architect role as described in Section 3.1. Whereas the web application architect tool supports the assembly, customization and development of applications based on these domain models by the component integrator role described in Section 3.2.

The CM Architect Workbench encodes the modeling and logic semantics that are specific to content manager and uses these semantics to prevent the construction of domain models which violate the semantics of the CM data store.

4.1 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. It is a framework and code generation facility that lets you define a model in any of these forms, from which you can generate the others and also the corresponding implementation classes. We defined a UML meta-model for content management and generated the corresponding Java classes using the EMF code generation framework. 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 Attributes. Similarly there are classes corresponding to other CM building blocks. Our plugin allows users to build CM data models based on this EMF meta-model. Thus the user can define different attributes and itemtypes that will be used by the application. The tool also provides a mechanism to visually represent the model as a set of diagrams thus making it more intuitive for the developer as well as for others with whom the model is shared. This is detailed below in Section 5.

4.2 Decoupled Model Development

The EMF framework allows the model 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 CM data models, the architect workbench is not tightly coupled

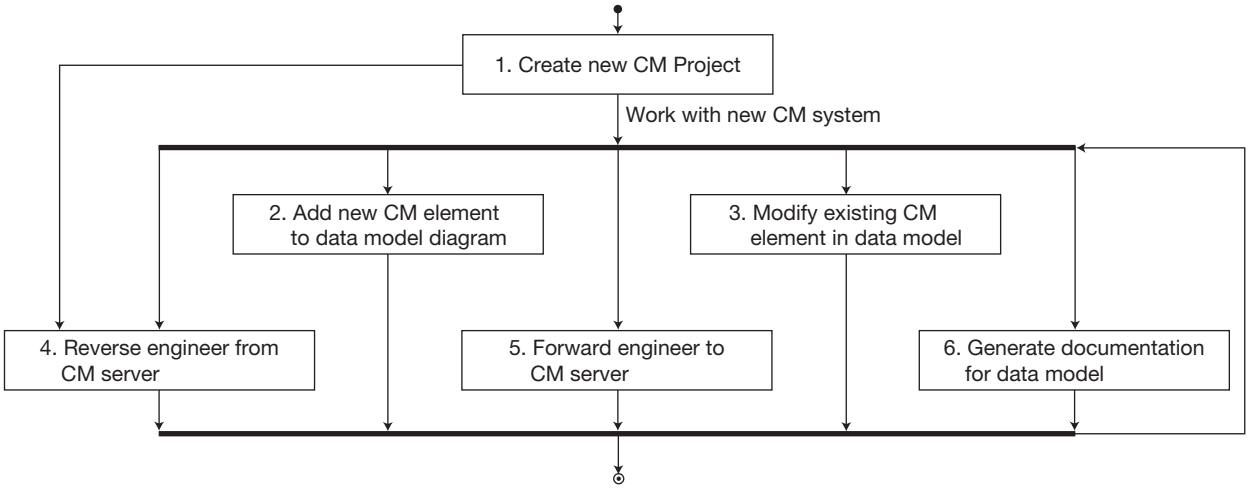

Figure 2 Activity diagram detailing CM Data Modeling process.

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 advanced development 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, and
5 It allows easier migration of data models from one server to another.

The activity diagram shown in Figure 2 summarizes the sequence of actions supported by the CM Data Modeling tool for model driven application development.

4.3 Forward/Reverse Engineering

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 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. However, the CM system might already have some model elements defined earlier. The tool compares the user defined model with the model already on the server to detect conflicts. Conflicts that cannot be resolved automatically are handled by taking user input into account. This is conceptually similar to a merge operation. 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.

5 WEB APPLICATION ARCHITECT

We have developed a GUI-focused component integration tool to support the component integrator role described in Section 3.2. This tool is focused on the needs of component integrator role and allows the user to assemble applications using the familiar drag and drop metaphor.

Our Web Application Architect tool is based on the WSAD development environment. We have implemented the tool as a plug-in to WSAD which can be used in parallel or independently of our Data Architect tool described earlier. The supporting business component libraries are based on the JavaServer Faces (JSF) component framework and all business components generated by our tool during application assembly also conform to the JSF specification [15].

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 that 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 passed to other business components.

Each business component relies on some range of input values from other components, performs some processing and provides some range of output values to other components. The component's processing may be triggered on a page load, or by explicit user input. Additionally, component processing may effect state changes in the application which have the side-effect of providing on-screen output to the user.

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.

Each of our business components conform to a well defined public contract [4] that is specific to an individual component or category of components. This contract defines the input name-value pairs expected to be available in the common data environment, the processing conditions and outcomes, and the output name-value pairs published into the common data environment by the component.

For example, the contract for a logon component specifies no input requirements, processing for a `logon` action in which the provided user details are authenticated. On successful authentication the component publishes a server connection under a specific name in the common data environment and generates a `logonSuccess` action. If the authentication fails the component generates a `logonFailure` action and stops processing. For a `reset` action, the component processing clears all user input elements and resets any component state established during a logon attempt. The output of the component consists of either a server connection and a `logonSuccess` action, or a `logonFailure` action.

Our efforts to support component-based development for CM have lead to the development of a library of business components and associated contracts which can be integrated to form

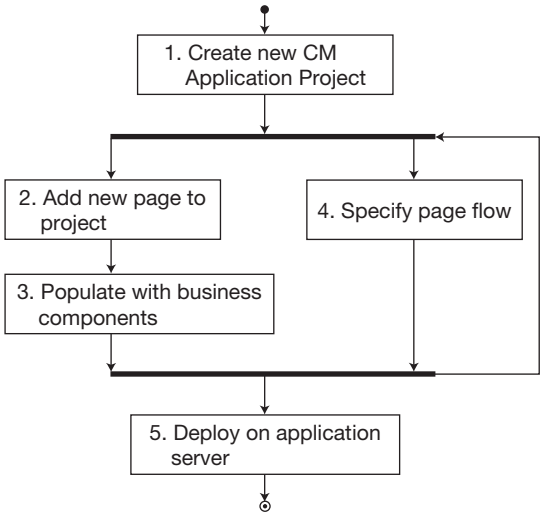

Figure 3 Web Application Architect activity diagram.

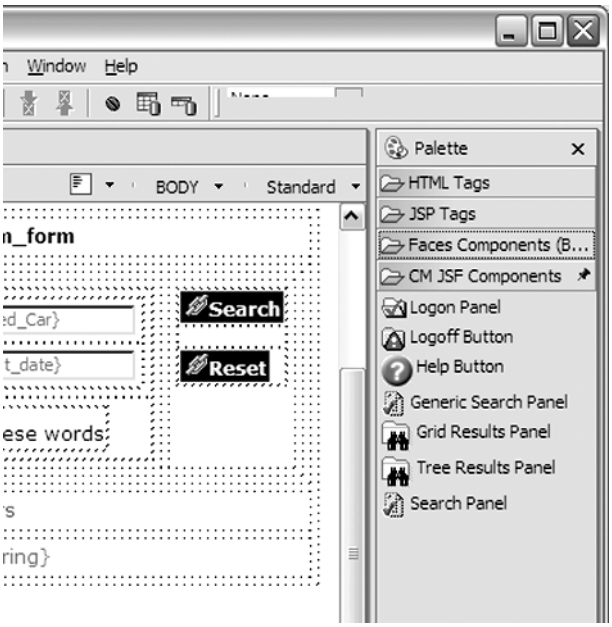

Figure 4 CM Web Application Architect tool palette.

a CM web-based application. We have categorized this library into two distinct groups:

- generic components, that are either unconcerned with the underlying data model or interpret the model at runtime, and
- specialized components, that are bound to a specific data model element (e.g. an item-type).

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 generated during component integration and are tightly bound to a specified data model or model element. These components provide a comparatively simpler implementation as against the generic components since there is no need for the component to interpret the data model at runtime. The implementation of these components is open to the user of our tool and is implemented in terms of domain model elements as defined in the data architect tool. This is achieved by generating data accessor classes based on the domain knowledge captured in the data model. The net result is that the implementation of our generated components tend to be cleaner and easier to further specialize than typical generic components.

5.1 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 specifying relevant data elements as required, (repeat steps 2 and 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 3.

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 4. The component palette with the data-independent and data-dependent components that it contains constitutes a library of domain-specific application components. The data-independent components may be written once and reused, whereas the data-dependent components are generated to operate on specific data during application development.

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 or a live CM server. 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.

5.2 Page Navigation

Page flow in the JSF model is controlled by a collection of navigation rules. Each rule defines either the destination of a page transition or an action method to invoke. An action method returns either null or another action. For rules which define action methods the action are handled by invoking the specified method and placing the resulting action back into the action event queue. Rules which define page transitions are handled by the JSF framework by performing a transition to the specified page.

For example, the `Logon` panel defines the four actions shown in Table 1. When the *Logon* button in the panel is clicked, the action `logon` is fired triggering the invocation `do logonCredentials.doLogon()`. This method authenticates the user, establishes a connection to the CM server and returns either a `logonSuccess` or `logonFailure` depending on the status of this processing. If an error occurs, an error message is put in the message queue and `logonFailure` is returned to trigger a transition back to the `Logon.jsp` page. If the `doLogon()` processing does not encounter errors, then a `logonSuccess` action is returned to trigger a page transition to the `Search.jsp` application page.

6 RELATED WORK

The Entity-Relationship (ER) model, originally proposed by Peter Chen in 1976 [7], is a

Table 1 Logon navigation rules.

Alias	*Action Reference*	*Destination*
Logon	`logonCredentials.doLogon()`	*
logonSuccess	*	`/Search.jsp`
logonFailure	*	`/Logon.jsp`
reset	*	`/Logon.jsp`

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.

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.

7 SUMMARY

We have presented a pair of tools designed to support model-driven development for Content Management. These tools supporting several roles in the application development lifecycle. Our goal has been to explore the effects of capturing a domain model in the application

developer tooling space. We are now investigating where relevant intelligence can be developed around the model tooling to support a broad range of development 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 and Component Integrator 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.
- JSF based business supports integration of CM data sources with other business resources such as foreign data repositories (e.g. relational databases) and existing applications (e.g. legacy code, web methods).

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. By simplifying application development and reducing the reliance on specialized development skills our tooling can reduce the development cycle for CM applications.

Our data modeling and component integration tools enable rapid application development cycles by simplifying the tasks performed by two major roles during CM application development. Initial feedback from CM customers and consultants on the value of this model-driven environment for CM applications has been extremely positive.

8 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., "A Hybrid Query Language for the Extended Entity Relationship Model", In *Journal of Visual Languages and Computing, Special Issue on Visual Query Systems*, 8(1), 1997.

[3] Angelaccio, M., Catarci, T., and Santucci, G., "QBD*: A fully visual query system", In *Journal on Visual Languages and Computing*, 1(2), 255–273, 1990.

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

[5] Bourdreau, 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 Spaienza", 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 *User Centered System Design: New perspectives in human-computer interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[14]Lyman, P., 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, CA, February, 2004.

[16]Olston, C., Wooddruff, 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`