ADAPTIVE BUSINESS OBJECTS
A New Component Model for Business Integration

Prabir Nandi, Santhosh Kumaran
IBM T J Watson Research Center, 1101 Kitchawan Road, Yorktwon Heights, NY, USA

Keywords: Business Process Integration, Business Performance Management, Model-driven Architecture, Model-driven Development, Composite Application, Services Oriented Architecture, Web Services

Abstract: We present a new component model for creating next generation e-Business applications. These applications have two overriding requirements: (1) Ability to change the application behaviour quickly and easily in line with the fast-changing business conditions and (2) Seamless integration of people, process, information, and systems. Our new component model is built around the concept of Adaptive Business Objects, and fulfils both the above requirements. This paper describes this component model and demonstrates its use in real business solutions.

1 INTRODUCTION

In today's global economy, enterprises are changing continually, entering into new markets, encountering new competitors, introducing new products and restructuring themselves through mergers, acquisitions, alliances and divestitures. In order to stay competitive in such environments, enterprises not only need to organize and operate their business processes in an efficient manner, but also require a solution that can easily adapt to the changes. This calls for new software platforms and technologies that support adaptive business solutions. In contrast, the emphasis of traditional business process management systems is on defining and managing the rigorous behaviour of highly repeatable business processes, overlooking the necessity for management of changes.

Another important trend in the enterprise IT solutions area is the growing emphasis on integration. The business value of IT grows substantially when the IT solutions seamlessly integrate people, processes, information, and applications. Information technology has made substantial progress on enterprise workflow systems, information integration, people collaboration, and application integration, but mostly in isolation. There is a need for new programming paradigms that bring people, processes, information, and applications together to create integrated solutions.

In this paper, we discuss a new component model and an associated programming model that begins to address these requirements. The component model is based on a concept called Adaptive Business Object (ABO). At its core, an ABO is a component with the behaviour defined using a Mealy Machine (Mealy, 1955) - a finite automata with output. In this paradigm, a computer program is a collection of communicating ABO instances.

The outline of the paper is as follows. In section 2, we present Adaptive Business Objects (ABOs) in detail. Section 3 discusses the programming model for creating programs using the ABO component model. Section 4 employs a simple customer scenario to illustrate the ABO programming model. In Section 5, we provide a clear description of the research contributions of this work and position it in the context of what exists today. Section 6 provides a brief overview of our experience in applying this technology to real life business problems. We provide a summary of related work in Section 7 and conclude by reiterating the value propositions of this technology in Section 8.

2 ADAPTIVE BUSINESS OBJECTS

An ABO is an abstraction of a business entity with explicitly managed state. The state information is manifested in a *currentState* attribute that provides a condensed view of the overall state of the entity at a specific point in its life cycle. However, the key value proposition of ABOs is the additional

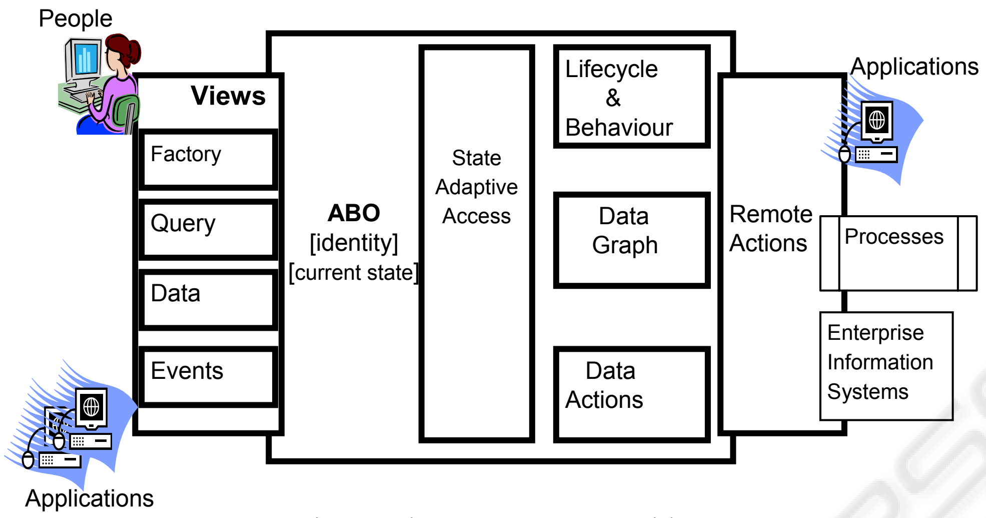

Figure 1: The ABO Component Model

functional contracts (depicted in Figure 1) that combine seamlessly with this basic characteristic (of explicitly managed state) to provide a holistic component abstraction that is semantically and syntactically complete in modelling all relevant aspects of the business entity. Below we discuss these functional contracts in detail.

2.1 Lifecycle & Behaviour

The lifecycle of the business entity modeled by an ABO is defined using a finite state machine (FSM). The states of the FSM represent the set of possible states the entity may assume throughout its lifecycle. An ABO receives external events via its public interface (or Views) and reacts to it via state transitions. Potential side effects are modeled by associating Data and Remote actions with state transitions. Thus the behaviour of an ABO is not hard coded into the methods that implement the interface as in traditional business objects. Instead, it is externalized via a model with finite state machine as its underpinnings.

2.2 Data Graph

Unlike traditional business objects, data is not contained inside an ABO. Instead, ABO uses a data graph to dynamically aggregate information on demand from heterogeneous data sources. Aggregation and presentation of data in this manner is just another manifestation of the ABO behavior and thus influenced by the state of the ABO. The graph structure implicitly enforces the data relationships and their cardinalities. This abstraction provides the modeler with the ability to specify a data model irrespective of the physical store i.e. structured, semi-structured or unstructured data. In other words data and metadata are not distinguished as such.

2.3 State Adaptive Access

People and software applications may interact with the ABO at various points in its life cycle. As part of such interactions, parts of the data graph may be accessed or manipulated or business events may be sent to the ABO by invocations of ABO's public interface. However, the ability of the outside world to raise events or access the ABO data will need to adapt based on the current state of the ABO. For example, a Purchase Order ABO may allow the *Requistioner* the ability to manipulate the Order data while the PO is in the *Draft* state. But once the PO has been *Accepted* for processing, the *Requisitioner* may have read-only access to the Order data. The representation of the entity lifecycle by state machine provides the ability to specify state adaptive access control to business roles. The modeler can specify read, write, search access on data and authorize access for events for each business role in each state.

2.4 Data Actions

The Data Actions are used to model CRUD operations on parts of or the whole data graph. However, they can only be added to the ABO as a side effect on state transitions. This effectively enforces the constraints regarding data integrity, transaction scope, access control and business semantics.

2.5 Remote Actions

ABOs affect the outside world via the Remote Actions. The model uses the well-known Command design pattern (E. Gamma et.al, 1995) to define the Remote Actions. The model can generate the appropriate Web Services Definition Language (WSDL) (Christensen, E. et.al, 2001) definitions for the command interfaces and bound the receivers to any network accessible service during deployment. In the business process context, such services could include other ABOs, workflow processes and enterprise information systems.

2.6 Views

Views present the external interface or API of the ABO. There are 3 main components that constitute the View.

Query. This enables the user to search for ABO instances satisfying certain criterion. Searchable fields include the *currentState* and the entire data graph. For example, one can search for all Purchase Orders that are in the Approved state, with a processing priority of 'High' and purchase amount 'greater than $1000'. The query returns a list of ABO instances.

Data. This is the interface to obtain partial or the whole data graph of a particular ABO instance.

Events. These specify the events accepted by the ABO and the corresponding event parameters.

For human users, the screens to drive user navigation can be automatically generated based on the state adaptive access. Thus there will be a 'view' for each ABO state for each interacting business role.

3 ABO PROGRAMMING MODEL

The ABO programming model is based on Object Management Group's (OMG) Model Driven Architecture/Development (MDA/MDD) (Mukerji, J. et.al, 2001) approaches. MDA/MDD is emerging as an important methodology to create software applications. The basic premise is to express the key abstractions (the models) of the application domain in UML and then transform these models to create executable code for a specific platform (e.g. J2EE). We employ the MDA principles as the basis of the ABO programming model.

Figure 2 shows the ABO programming model in a nutshell. We have created a UML class diagram of the ABO component model. A detailed description of the UML model is out of scope of this paper but we will capture the key facets in the next section. Rational Rose XDE Modeler (Rational XDE, 2004) was used as the tool to create the ABO UML model. The Eclipse Modeling Framework (EMF, 2004), is a modeling framework and code generation facility for building tools and other applications leveraging Object Management Group's MOF (Meta-Object Facility) technologies.

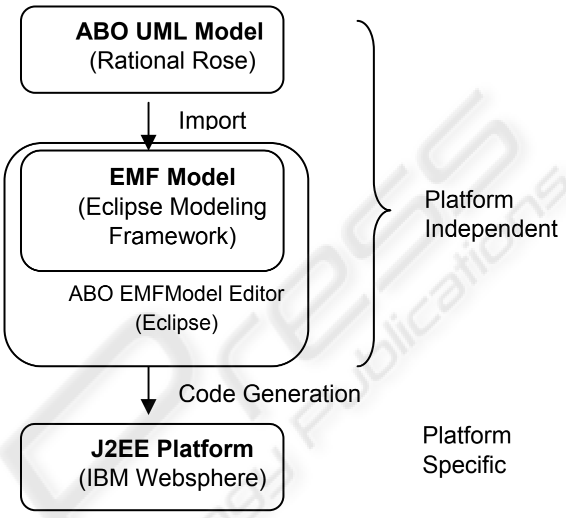

Figure 2: The ABO Programming Model

From a model specification described in XMI (XML Metadata Interchange), 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. The EMF plug-in for Eclipse was used to import the ABO UML model, generate the corresponding Java classes and also create the basic editor. The basic editor was then augmented to make it more user-friendly. This tool was then used to create the application specific ABOs. The EMF tool saves the ABO model in XMI format. We created a collection of code generation utilities that takes the ABO XMI files and generates appropriate platform (J2EE) specific artifacts deployable and executable on IBM WebSphere platform. We will cover the details of the J2EE code generation and mapping in our next section.

4 ABOS IN ACTION

In this section we illustrate the use of ABOs in creating a business integration solution by applying the ABO programming model to solve a real customer problem.

4.1 Scenario

The business scenario involves a large Retail store fulfilling job openings in one of its stores. The process starts with the Store Manager selecting a job opening to be filled. Subsequently an advertisement is released, applications are gathered, applicants are short-listed, interviewed and screened. Finally one or more applicants are selected for the job.

The process requires information to be integrated from heterogeneous sources. For example, applicant resumes are submitted online, faxed, or mailed. Faxed or mailed resumes are converted to an electronic format. The resumes are then kept with the Retail store's content/document management system. The background screening required for selected applicants is done via a Web Service made available by a federal agency. The rest of the information is available in structured formats and stored in relational databases.

4.2 ABO Design

Traditional analysis and design of business processes focus exclusively on activities performed during the business process execution, the ordering of these activities, and the data that flows through these activities. We postulate that the artifacts are just as important as the activities. The artifacts correspond to entities that are created, modified, or destroyed as a result of the business process execution. This artifact view of the business process is different from the data flow model just as the activity flow model is different from the basic action semantics of a computer program. Both specify

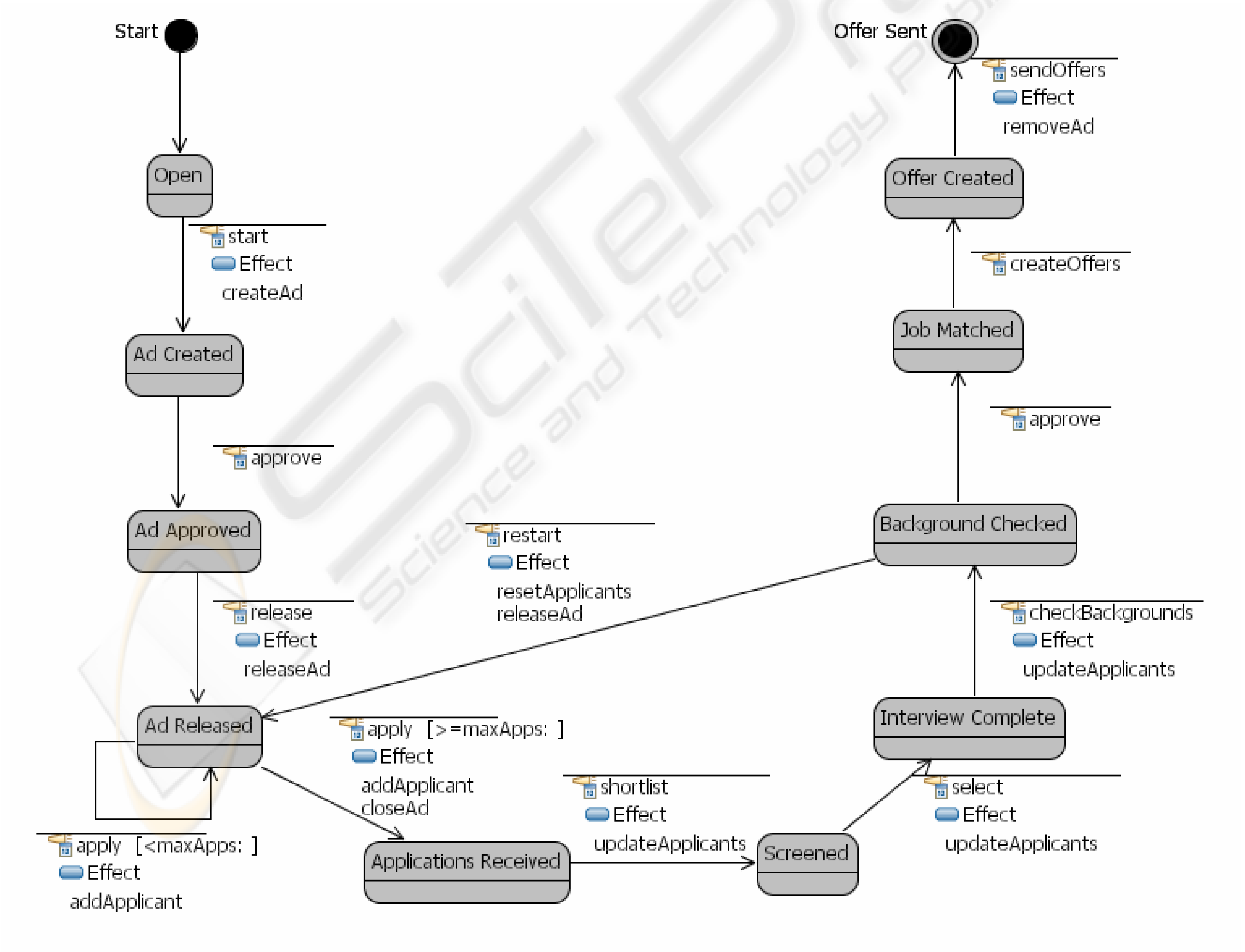

Figure 3: The Job Opening ABO State machine

meaningful constraints on a generic model such that they can be applied to solve problems in a specific application domain.

The artifact-centric analysis of the business process leads to a view of the business process as a sequence of activities used to create, manipulate, and close a Job Opening. In other words, this business process is really about managing the lifecycle of a Job Opening. Thus we use an Adaptive Business Object to model the Job Opening. This ABO brings people, information, and applications in the context of this business process. The business roles involved in this process are the Store Manager, the HR Representative and the Applicant.

4.2.1 Lifecycle and Behaviour

The Job Opening entity lifecycle is shown in Figure 3. The UML state chart is used as the design tool. State transitions are labeled as <Event>[Condition]/[Action]. For example, the transition from the *AdReleased* state to the *ApplicationsReceived* state is labeled as *Apply[>=maxApps]/addApplicant,closeAd* where *Apply* is the event, *>=maxApps* is the condition and *addApplicant, closeAd* are the actions. The ABO behavior demonstrated by this state transition definition is as follows. When an application arrives, the ABO state will be changed from *AdReleased* state to *ApplicationsReceived* state and the applicant will be added to the list of applicants if the number of applications received so far (including the current one) equals the maximum number specified. The evaluation of *>=maxApps* condition, may be delegated to a rules engine at runtime. The rules engine may use the current business policies in place to calculate the value of *maxApps*, the maximum number of applications that will be received for the job. The second action, *closeAd,* will initiate the appropriate set of steps to withdraw the advertisement, as the maximum number of applications for the job has been reached.

4.2.2 Data and Remote Actions

The Data actions and Remote actions are indicated in the state machine diagram as transition actions e.g. *Create Ad, Add Applicant* etc. Data Actions will be specified with the portion of the data graph that it manipulates.

4.2.3 Data Graph

The information brokered by the Job Opening ABO is federated among data repositories distributed across the network. The Job Opening data graph, shown in Figure 4, models the topology of this information. The Job Opening ABO uses this information to aggregate the views of the ABO on demand. Specifically, the data graph shows that the information needed to manage the Job Opening process is provided by five relational stores (Job Opening, Advertisement, Applicant, Contact, Interview), one document store (Resume), and one Web service (Background).

The nodes in the data graph are annotated with the type of data store. The data graph does not contain actual data, instead describes how data can be aggregated from enterprise information systems, services, and applications. For example, the applicant resumes are stored in a document store and the Resume node of the data graph merely holds the relevant metadata with a field (resume doc) holding the URI to the actual document. Thus the Job

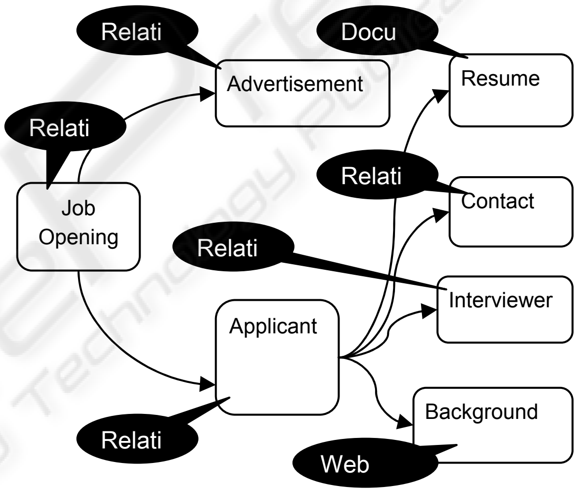

Figure 4: Job Opening ABO Data graph

Opening ABO data graph captures in a single logical model the information that requires to be integrated in the context of the process.

4.2.4 State Adaptive Access

Access for manipulating the Job Opening data graph and/or sending events to it need to be restricted between the three business roles at different states of the Job Opening ABO. For example, Store Manager and HR Rep are authorized to view the Job Opening ABO while it is in *AdCreated* state. But only the HR Rep can send the *ApproveAd* event.

4.3 Platform Specific Mapping

In this section we discuss the platform-specific mapping of the ABO model to the J2EE platform.

This is accomplished by generating J2EE specific code from the ABO XMI file.

Figure 5 shows the Job Opening solution running on a J2EE platform e.g. IBM's WebSphere Application Server (IBM Websphere, 2004).

Each node (data group) of the Job Opening ABO data graph maps to an Entity EJB. Appropriate container managed relationships are generated to indicate the relationships and their cardinalities. The fields in the data graph become the entity EJB attributes with some marked as key fields. The *sourceType* field of each data group is mapped to the different JCA resource adapters through which connection and access of the heterogeneous data stores are managed.

Client access to the "data graph" is via the Job Opening Data Session EJB. We use an implementation of the Service Data Object (SDO) specifications (Beatty, J. et.al, 2003) for this purpose. The ABO data graph is mapped to a SDO disconnected data graph structure and read, filled, and queried via the EJB Data mediator configured with the Job Opening Data Session EJB. Although the SDO architecture supports heterogeneous data sources directly, we have preferred to implement data graph nodes as entity EJBs backed by JCA resource adapters (RA) primarily to take advantage of the caching and transaction support provided by the J2EE container. It also discards the necessity of using diverse data mediators.

The Job Opening State Machine EJB is a specialized implementation for state machine support using container managed entity EJBs. The operations/events from the ABO definition is mapped as remote methods whereas data and remote actions are mapped to private methods of the entity EJB. A specialized state machine controller maps remote method invocations to private methods based on the current state of the EJB and the state machine definition. Guards are also mapped to private methods.

Lastly, the state adaptive access specifications are used to create the portlets that bring together the three view components – query, data and events specific to the interacting business role player.

5 ANALYSIS

In this section, we discuss how the ABO component model supports the requirements of adaptivity and integration we laid out in the beginning of the paper for the next generation enterprise solutions.

ABO model provides adaptivity at three levels:

1. There are two design points that contribute to the adaptive behavior of an ABO at the lowest level.

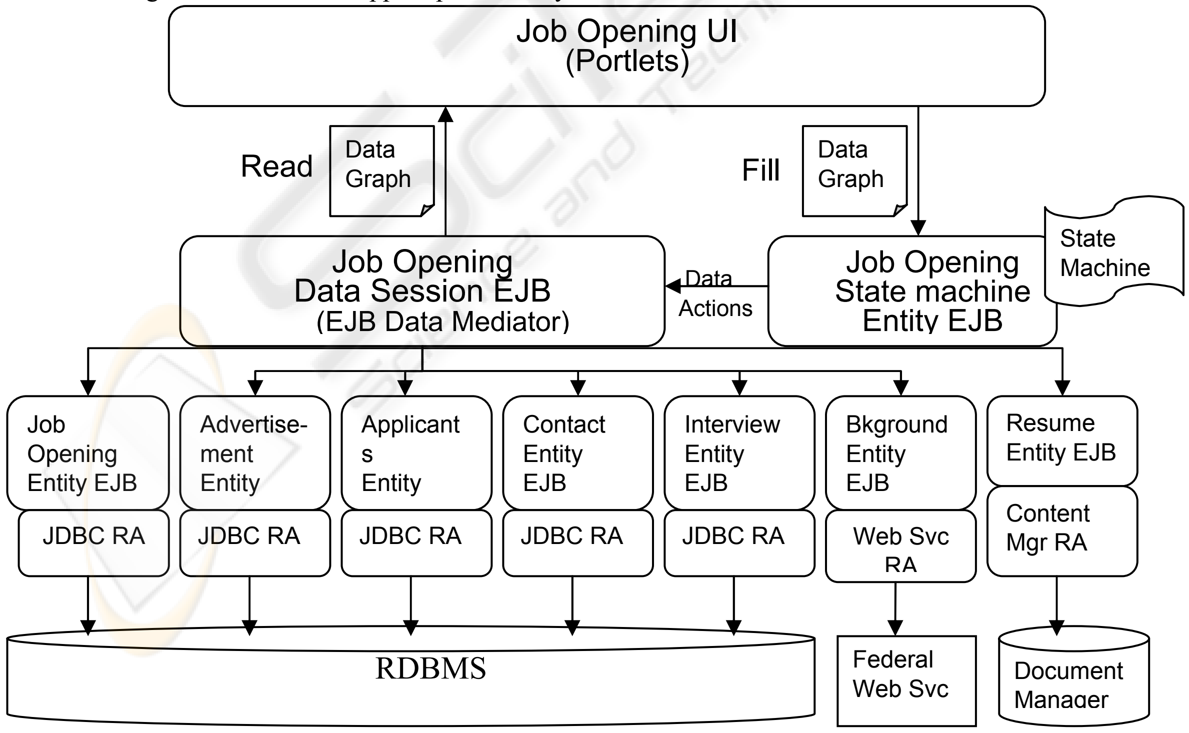

Figure 5: Job Opening Solution on the J2EE Platform

First, the use of command design pattern for remote and data action invocations as part of state transitions leads to dynamic service binding and provides flexibility in the way an ABO effects the environment in response to the processing of an event. An example of this is the “Release Ad” action of the Job Opening ABO. The actual releasing of an advertisement could change based on the business policies and business environment. The command design pattern enables the dynamic binding of an appropriate release mechanism with the *Release Ad* action at runtime. Second design point is the use of a rule engine for condition evaluation. The result from the condition evaluation is used to determine which transition to take when an event is received. By externalizing this to a rule engine, it becomes possible to make this decision based on business logic brought in play dynamically at runtime.

2. The definition of the ABO behavior using a formal computational model coupled with our use of Model Driven Architecture as the basis of the ABO programming model facilitate adaptivity at the second level. The ability to externalize the behavior via a formal model and the ability to create runtime artifacts that implement the behavior via automated code generation lead to adaptive components. The externalized models may be manipulated programmatically or by human involvement. For example, the Job Opening process may be streamlined by triggering the screening process as soon as an application is received. This change can be easily implemented by changing the state machine definition of the Job Opening ABO using an n appropriate tool and regenerating the runtime artifacts.
3. We envision the next generation enterprise integration solutions to be made up of collections of **communicating ABO instances**. The emergent behavior of such a system is determined by the behaviors of the participating ABOs. System behavior may be changed by adding a new ABO to the system and modifying the behaviors of the existing ABOs. For example, the Job Opening system with just one type of ABO could be easily augmented to include "offshoring" by creating an “Offshoring Opportunity” ABO. The net result will be to check for opportunities to offshore whenever a job opening is created.

ABO model achieves seamless integration of people, processes, information, and applications by having a simple computational model that subsumes all these facets of a business integration solution. Below we describe how these facets are effortlessly captured in the ABO computational model.

1. The views are the mechanisms by which people are integrated with the system. The views provide context-sensitive, role-based access to the information brokered by the ABO. Additionally, views also provide a mechanism to send business events to the ABO.
2. The key to business process integration is the explicit definition and management of the state of the ABO. Essentially, a business process results from the tracing of the states of the ABO as it processes business events. People are easily integrated with the business processes since the views serve as the conduit of the business events originating from people.
3. The data actions fired as part of state transitions and the data graph these actions use to dynamically aggregate information from federated data repositories are the facets of ABO that facilitate information integration. People, processes, and information meld together as a business event originates due to user interaction with an ABO View, this event triggers a state transition, this transition invokes a data action with an appropriate data graph as an argument, and his action returns with aggregated data that is delivered via the modifications to the View. ABO provides a simple component model such that instances of ABO exhibits this integrated behaviour at runtime. Application integration with processes, people, and information is achieved by invoking Remote actions as part of state transitions.

In traditional workflow systems, data integrity poses a big challenge as the complexity of the workflow starts to grow. The ABO approach could augment this process and provide encapsulated artifacts to manage and mitigate this risk. Encapsulated, state adaptive artifacts, like ‘an order’, ‘a customer service request’, ‘the job opening’ etc., ensure data integrity by allowing access to the contained data if and only if the current state allows it.

Designing business process applications around a set of key business artifacts provides a nice factoring of concerns and enables a way to ‘break’ and broker workflows. The ABO encapsulates what needs to be done (via the events it can accept in that state), while the workflows specify how to achieve it (launched via transition actions). A process thus designed can scale to an arbitrary limit in terms of its complexity. Analogies can be drawn between object orientated programming and procedural programming. Workflow based languages (like BPEL4WS (Andrews, T. et.al, 2003)) provide a “procedural” semantics to business process modeling, whereas the artifact-centric modeling in ABO's augments 'procedural' descriptions with the power of object-oriented paradigms. Additionally, time tested object-oriented design patterns can more naturally be applied to design complex, adaptive business processes and systems.

6 REAL LIFE VALIDATION

In Section 4, we described a relatively simple business application to illustrate the ABO concepts and programming model. However, we have successfully applied the ABO design principles, tools, patterns and techniques to real world customer problems. Below is a partial list with brief descriptions about the business problem and the ABO design choices.

The first customer provides IT outsourcing services to Small and Medium Businesses. The end to end business processes cover SOW, Quote, Order, Installation, Invoicing, Parts management and Supplier management. The process was designed with 8 communicating ABOs,

Customer, is used to manage customer related data and their business status – live, inactive etc.

Engagement, is a primary artifact that provides a container for the master contract – the components required for each site, the tasks and work breakdown structures assigned to the individual service providers and parts needed for each task. This ABO also interfaces with a content management system that stores the electronic copy of the master contract.

Parts Catalog, manages the parts data.

Schedule, is the other primary artifact that handles the execution of the order from its inception till installation is completed at every site. Its lifecycle models problem resolution, coordinating amongst the different tasks, supporting customer and vendor interactions and ensuring the timeliness of the order execution. Some of the lifecycle states are, *Pending, Plan, InInstallStart, Live, Exception, InResolution, CustomerAccepted, Complete* etc.

Services Catalog, manages the service provider (vendor) data and the services they provide.

Site Profile manages information about the installation sites.

Statement of Work (SOW), the live document transacted between with the customer to negotiation and settle on the cost, schedules and milestones.

Task, a primary artifact that manages the execution of a single task. It communicates with the Schedule ABO at different points in its lifecycle to resolve problems and at the end signaling its completion. Some of the lifecycle states are *Live, Rejected, Accepted, Reschedule, Exception, Completed* etc.

The second customer is a premier Auto Services & Retail shop. The business process involves managing their Service Work Order process from appointment handling and scheduling, handling car drop-offs, pricing, technician assignment and real time line item level execution visibility. There system is implemented around two main ABOs,

Service Transaction, manages the overall service work order lifecycle and coordinates with individual line item execution. Some of the lifecycle states are, *AppointmentScheduled, AppointmentConfirmed, DroppedOff, Estimated, Purged, Hold*, *ReadyForWork, Working, Complete, ReadyForPickup, Delivered* etc.

Line Item, manages the execution of a line item from technician assignments, parts procurement, customer authorization and relays status information back to the Service Transaction as appropriate. Some of the lifecycle states are, *HoldForAuhthorization, Unassigned, HoldForParts, Assigned, Void, Working, Completed* etc.

The third customer is one of the top Telecom services providers. The business process provides end to end Order Provisioning from Sales Support, Ordering, Field Ops and Post Install. The Ordering process has been implemented so far and consists of 4 ABOs,

Order, is the primary artifact and coordinates amongst a number of workflows that do approvals, site surveys and reviews. It moves into a Provisioning state by creating and launching the children ABOs and coordinates the concurrent processing of individual parts of the Order. The Order moves out of the Provisioning state when the last of the order items have been provisioned. An interesting use case is how the Order ABO reacts appropriately to handle cancellation requests at various stages of the provisioning process.

DSL, provisions the DSL service.

Long Distance, provisions the Long Distance part of the provisioning. It has a dependency on Local being provisioned first.

Local, provisions the Local service

7 RELATED WORK

This work builds on our earlier work on Adaptive Documents (ADocs) (Nandi, P. et.al 2003). In ADocs we proposed a programming model for "business artifacts" that display state-dependent behavior and its usefulness in implementing complex business processes. The primary focus of that work was to show the collaborative aspect of the

ADoc in its ability to facilitate collaboration amongst a set of agents (human or software) to execute activities as units of collaboration. We demonstrated this with a real world business example where activities were generated by workflows. In ABOs we propose a formal component model that maintains the collaborative and state-dependent behavioral aspects of ADocs, but extends it for information integration. We also assert that ABOs provide a semantically complete Model-View-Controller (MVC) programming model in a single holistic component model.

There is significant amount of work in using state charts as the basis of modeling the accessibility, lifecycle and behavior of objects including the well-known State pattern (E. Gamma et.al, 1995). Some notable examples are its use for interface sequencing (Whaley, J. et.al, 2002), service composition (Jeng, J. et.al., 2000), communication protocols (Hanson, J. et.al, 2002), as controllers in MVC implementations, reactive and real-time systems (Harel, D et.al, 1998). The ABO component model draws from and builds on all such prior work. Interface sequencing is the ability of an ABO component to enforce constraints on the sequence in which it accept events via its application interfaces. Service composition is the ability of an ABO component to choreograph various backend services via actions on state transitions. ABOs may participate in dynamic, stateful, multi-party conversations by using remote actions for sending messages, views for receiving messages, with the messaging functionality delegated to the action implementers. ABO composition supports protocol nesting.

(Mukherjee, J. et.al, 2001) used state charts to model video data. The video objects in the video stream were partitioned into meaningful segments by state charts according to the object properties and their transitions. The database indexes were based on the states assigned to a group of objects rather than the objects themselves. If we use the ABO model to solve this problem, the heterogeneous content is aggregated based on relevant metadata and retrieval and access is dynamically assigned based on the state of ABO and the role of the requester.

State machines are in common use to specify the "system contracts" in most component models viz. COM, CORBA, EJB etc. (Smith, R., 2001). These system contracts describe persistence and transactions policies required of the component containers. The ABO component model can be mapped to an implementation framework and we have demonstrated one such mapping to J2EE. One way to think about the use of state machines in the ABO model is to prescribe the "application contracts". Note that when we map the ABO model to the J2EE model, these application contracts are implemented on the system contracts of the EJB container.

8 CONCLUSION

We argue that new software technologies and methodologies are needed to meet the demands of the next generation enterprise solutions. The primary characteristics of these solutions are adaptability and seamless integration of people, processes, information, and applications. In this paper, we presented a new component model called Adaptive Business Object (ABO) and a programming model for ABOs based on Model Driven Architecture to create adaptive business solutions that integrate people, processes, information, and applications.

ACKNOWLEDGEMENT

The authors wish to acknowledge the constructive feedback and critique provided by Nathan Caswell, Terry Heath, Robert Guttman, Fred Wu, Kumar Bhaskaran, Kamal Bhattacharya, Shubir Kapoor and David Cohn at IBM Research. They would also like to thank Mike Conner, Manish Modh and Eoin Lane of IBM's Software Group, for their enterprising efforts in applying the ABO model successfully to real life customer problems. Special thanks go to Chang Shu, who designed and developed the code generation utilities to map ABO models to IBM's Websphere platform and made this effort a reality.

REFERENCES

Mealy, 1955, George H. Mealy, *A method for synthesizing sequential circuits*, Bell System Technical Journal, 34(5):1045-1079, 1955.

E. Gamma et.al, 1995, "*Design Patterns – Elements of Reusable Object Oriented Software,*" Addison-Wesley Publishing Company, NY, 1995.

Mukerji, J. et.al, 2001, *Model Driven Architecture*, http://www.omg.org/cgi-bin/doc?omg/03-06-01

Rational XDE, 2004, *Rational Rose XDE Modeler*, http://www-306.ibm.com/software/awdtools/developer/modeler/

EMF, 2004, *Eclipse Modeling Framework* , http://www.eclipse.org/emf/

IBM WebSphere, 2004, *Websphere Application Server*, http://www-306.ibm.com/software/webservers/appserv/was/

Beatty, J. et.al, 2003, *Service Data Objects*, ftp://www6.software.ibm.com/software/developer/library/j-commonj-sdowmt/Commonj-SDO-Specification-v1.0.pdf

Nandi, P. et.al 2003, *ADoc-Oriented Programming*, The 2003 International Symposium on Applications and the Internet (SAINT'2003, Jan 27-31, 2003, Orlando, Florida, USA)

Smith, R., 2001, *The Screw Analogy.(Industry Trend Or Event)*, Smith, Roger - Software Development, v9 n3 March, 2001 / p7

Whaley, J. et.al, 2002, *Automatic extraction of object-oriented component interfaces*, International Symposium on Software Testing and Analysis. Proceedings of the ACM SIGSOFT 2002 International Symposium on Software Testing and Analysis 2002 p 221-231

Jeng, J. et.al., 2000, *Designing an FSM architectural framework for service-based applications*, IEEE Computer Society's International Computer Software and Applications Conference 2000. IEEE, Los Alamitos, CA, USA,00CB37156. p 234-239

Park, J. et.al. 1997, *Compositional approach for designing multifunction time-dependent protocols*, Proceedings of the 1997 International Conference on Network Protocols International Conference on Network Protocols 1997. IEEE Comp Soc, Los Alamitos, CA, USA,97TB100174. p 105-112

Hanson, J. et.al, 2002, *Conversation Support for Business Process Integration*, The 6th International Enterprise Distributed Object Computing (EDOC'02 - Sep 17-20, 2002, Ecole Polytechnic, Switzerland)

Hanson, J. et.al, 2002, *Conversation-enabled Web Services for Agents and e-Business*, The 3rd International Conference on Internet Computing (IC'02 - June 24-27, 2002, Las Vegas, Nevada, USA)

Mukherjee, J. et.al, 2001, *State chart based approach for modeling video of dynamic objects*, Proceedings of SPIE - The International Society for Optical Engineering v.4520 2001 p.74-83

Sanden, Bo I. 2000, *Implementation of state machines with tasks and protected objects*, Ada User Journal v 20 n 4 2000. p 273-288

Harel, D et.al, 1998, *The Statemate Approach: Modeling Reactive Systems with Statecharts*. Harel, D., Politi, M., McGraw-Hill, 1998.

Christensen, E. et.al, 2001, *Web Services Definition Language (WSDL),* http://www.w3.org/TR/wsdl

Andrews, T. et.al, 2003, *Business Process Execution Language for Web Services (BPEL4WS)*, http://www-106.ibm.com/developerworks/library/ws-bpel/