Cuarto Congreso Colombiano de Computación 4CCC
Sociedad Colombiana de Computación
UNAB - UIS
© 2009 Los Autores. Todos los Derechos Reservados.

A State Machine Based Coordination Model applied to Workflow Applications

Mario Sánchez*
Universidad de los Andes
Bogotá, Colombia
mar-san1@uniandes.edu.co

Jorge Villalobos
Universidad de los Andes
Bogotá, Colombia
jvillalo@uniandes.edu.co

Daniel Romero
University of Lille 1,
INRIA Lille, Nord Europe,
Laboratoire LIFL UMR CNRS 8022
daniel.romero@inria.fr

ABSTRACT
Traditionally, workflow applications use a single language to describe every relevant detail of a business process. Therefore, the complexity of the languages used and their implementations has increased, creating problems related to evolution and maintenance. A possible approach to lower this complexity is to separate the elements of a process according to dimensions or perspectives, similarly to what is done in Aspect-Oriented Programming. The problem is that most workflow tools do not support explicit dimensions, and previous implementations of these ideas had important limitations.

This paper presents Cumbia, a platform to build workflow applications supporting multiple dimensions. In Cumbia, an executable model is used for each dimension, and these executable models are expressed with a coordination model based on synchronized state machines. Among other advantages, this approach renders possible the usage of dimension-specific languages, thus easing maintenance and evolution of processes, engines and languages.

Keywords
Programming and Software Engineering, Model Driven Engineering, Business Process Modeling, Aspect Oriented Workflows.

1. INTRODUCTION
Nowadays, a growing number of contexts are taking advantage of workflow applications. In these, there is a central coordination element that leads the cooperative execution of several active entities, and provides a way to integrate them to achieve a common goal [16]. Perhaps the best known workflow language is currently BPEL [28], which is used for composing and coordinating web services. Moreover, there are nowadays hundreds of different languages and engines, which offer different features and are usually targeted towards particular application contexts. For instance, there are workflow languages specialized in scientific applications [25], in ubiquitous computing [11], or in human interaction [29].

The usage of workflow applications has surged because of the many advantages they offer. One of these advantages is the ability to separate the order of execution of tasks (the coordination) from the actual tasks performed (the computations), which eases the integration of heterogeneous components. Furthermore, this separation between coordination and computation favors modularity and reuse. This can be seen when new processes are created, using only existing, configurable elements. Another important characteristic of workflow applications is that they are used in contexts that tend to evolve frequently, because they provide the necessary flexibility to adapt to the changes. For instance, in a financial company it is common to have changing business rules, changing processes, and new systems and tools that have to be integrated to the existing application stack. Workflows are capable of handling this kind of changes without making huge investments in new systems.

The central element in a workflow application is the control dimension (or perspective): it comprises entities that describe the control-flow, that is, the tasks that have to be performed, and their ordering [26,6]. In most applications, control is complemented by entities from other dimensions, such as data, time and resources. Which dimensions appear in a specific application depends on the context where that application is used. For instance, in an application to support distributed software development processes it is not enough to describe the tasks to be performed. It is also necessary to describe the structure of the development team, the capabilities of its members, the timing restrictions that the process should abide, and management policies for the data produced by the process. Thus, the description of these processes requires more than control: it also requires elements from the dimensions of resources, time and data.

The problem that we address in this paper is that currently most workflow engines make no separation between dimensions. Thus, users have to use languages that mix elements from every dimension, and therefore, these dimensions become permanently entangled. From the users' point of view, this makes processes more difficult to maintain and to evolve. From the point of view of developers of workflow engines this also has important consequences. On the one hand, the evolution of languages tends to make them grow and become complex. Therefore, the engines to support their execution also tend to become more complex and

* Supported by the VLIR funded CARAMELOS project: http://ssel.vub.ac.be/caramelos/ and by Colciencias.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Conference'04, Month 1–2, 2004, City, State, Country.
Copyright 2004 ACM 1-58113-000-0/00/0004…$5.00.

difficult to maintain. On the other hand, the capacity of evolution of these applications is also limited because a high coupling between dimensions makes it difficult to adapt them independently to changing requirements. Furthermore, it becomes very difficult to add extra dimensions late in the development cycle.

With current mainstream techniques to build workflow engines, it is not easy to have multi-dimension support. In the first place, these approaches lack the flexibility to represent various dimensions and support all their variations. On the other hand, they do not offer the capabilities to manage the changing relationships between dimensions. As section 2 shows, there have been works to solve some of these problems, and some existing approaches offer partial solutions to them.

This paper presents Cumbia, a platform that can be used to build workflow engines with multi-dimensional support. This means that Cumbia makes it possible to execute processes described as a composition of dimensions, and use a different language to describe each dimension. This has the following advantages: first, control languages are simplified because they do not have to include elements unrelated to control; it is also possible to design very suitable[1] dimension specific languages, and thus processes easier to understand and to maintain; finally, evolution of the engine is simplified because each dimension can evolve independently, within certain restrictions [22].

The fundamental proposal of Cumbia is to use a state machine based coordination model to describe the dimensions. This is achieved by means of dimension specific metamodels described in terms of a coordination element that we have called *open objects* (see section 3). Thus, workflow processes are groups of models conformant to those metamodels, which are executed in a coordinated fashion. The underlying coordination mechanism offers characteristics that are important in workflow applications: it supports synchronous and asynchronous interaction, and it offers powerful extensions mechanisms (see section 4).

This paper introduces the Cumbia platform and presents its main elements. First, it presents some related works that produced interesting ideas that inspired Cumbia. Then, section 3 presents the basics of the coordination model and the open objects. Section 4 illustrates the usage of open objects in *Cumbia-XPM*, a metamodel for describing the control dimension of processes. Finally, section 5 presents the tools developed to support the usage of Cumbia, and some Cumbia-based applications used in different contexts.

2. RELATED WORK

This section presents other research groups' works that cab be related to Cumbia. We have classified these works in the following three broad topics: coordination models and Reo; aspect-oriented workflow languages; and YAWL.

[1] The term suitability refers to the match between the constructs available in the modeling language and the concepts in the application domain [12]. Suitability is the metric used in many works to explore the relationship between workflow languages and patterns: control-flow patterns [18], resource patterns [19], data patterns [20], etc.

As Papadopoulos and Arbab showed in their survey [16], there are many different coordination models, which can be classified as data-driven or control-driven. In the former group, they mainly focused on Linda and its descendants. However, Linda-like models have some deficiencies that make them inadequate to model workflows [17]. Nevertheless, some of their underlying ideas are useful in modern coordination models, such as the independence between the computation and coordination, or the usage of a shared space for data (as is required in one of the implementation strategies described in [21]). Among the control-driven models identified in [16], there is Manifold [17]. This model has several similarities to Cumbia-XPM and the open objects, but its design totally separates data from control. Whereas in Cumbia-XPM we can use the flow of data to control the state of a process, Manifold is completely event-driven. Also, there is no direct interaction between elements, other than the flow of data through streams.

A more recent and powerful coordination model is Reo [2]. It is an exogenous language based on channels that can be used to create very complex connectors. The usage of Reo as glue-code between heterogeneous components promotes loose coupling because it makes possible to separate them and externalize their interaction and the flow of data between them. Reo has been applied to the problem of composing web services, as is presented in [7] and [13]. In these works, several features of Reo lead to advantages over other solutions such as BPEL: using Reo it is possible to dynamically reconfigure the channels to restructure the processes; distribution and mobility is also supported by the coordination model; finally, the formal description of Reo makes it possible to apply model checking techniques to the web services compositions.

One of the fundamental points of our proposal is the separation of processes into different perspectives, which is a strategy that has been applied in a small number of other workflow related projects. AO4BPEL [5] is an aspect-oriented workflow language based on BPEL, which allows the definition of aspects using BPEL for the advices and XPATH as point cut language. AO4BPEL offers static and dynamic weaving, using a specialized BPEL engine. Padus [3] is also a workflow language that extends BPEL. It has a logic-based point cut language, and it also uses BPEL to describe the advices. It only supports static weaving, but since the outputs of its weaving process are valid BPEL processes, they can be run in unmodified BPEL engines. There are several differences between Cumbia and AO4BPEL or Padus. In the first place, they can only use BPEL as the advice language, and thus it is impossible to have dimension specific languages. Furthermore, join points are limited to syntactic elements present in the process definition, or to 'internal join points', which are fixed steps required for web services' consumption. Compared to the flexibility offered by Cumbia and the open objects, which can have state machines as complex as necessary, their approach limits the possible extensions. Finally, there is a difference in the way they modularize with concerns. Although in many cases they use aspects to reduce tangling, they also use aspects to express changes in a modular fashion. We do not share this usage because aspects should modularize crosscutting concerns and a concern is not crosscutting with itself. Although it is a useful decomposition, it should not be called aspect-based.

Another project that has explored the application of aspects to workflows is AMFIBIA [1]. AMFIBIA is a metamodel that formalizes the core elements of business process modeling. It is based on a core that groups common concepts, and on concerns (aspects in their terminology) that specialize these shared concepts. AMFIBIA has several similarities to Cumbia: it separates workflow dimensions; it is formalism-independent and it is possible to have dimension specific languages; the dimensions used are not fixed and new ones can be added. Nevertheless, there are important differences between AMFIBIA and Cumbia, particularly in the relation between the elements of each formalism and the common elements. In their approach, there is a mapping between the elements of each concern and the core concepts. In Cumbia, the elements of each concern are specializations of the open objects. In their case this mapping is the foundation of the synchronization mechanism, because the coordination can only happen if the core is included.

Finally, there have also been projects that have explored particular dimensions (or perspectives) in workflows. For instance, in [21], the main topic discussed is the data perspective, but several works related to other perspectives are also referenced (time, resources, transactions, and functionality). However, there is an essential difference between their strategy and ours in that they propose the need to have a clear separation of perspectives when modeling a process, even if its implementation and execution integrates the perspectives into a single solution. With our strategy, the perspectives are separated when modeling and stay separated, but coordinated, during execution.

As discussed before, most existing workflow languages and engines combine all the dimensions present in a process. YAWL is a very well known exception to this, since it is a workflow language that was specifically designed to support the control perspective and the original control-flow patterns [26]. From the Cumbia viewpoint, YAWL is also interesting because it clearly describes the workflow language and the underlying coordination model as two different things, albeit tightly related. YAWL's development started with an analysis of the Petri nets' suitability to accommodate the patterns and, after finding some limitations, they proposed a formal coordination model based on a Labeled Transition System (LTS). Similar to Cumbia-XPM, YAWL is an intermediate language to support the execution of high level languages. Since the coordination model was inspired by Petri Nets, it includes concepts such as places, conditions and tokens. It also uses state machines to describe the life-cycle of elements, but these nets are identical for every element and are used only for documentation purposes and not for coordination or composition. One advantage of the formal semantics of YAWL is the possibility to analyze and verify processes, which has been explored using a tool called Woflan[2]. Currently, neither Cumbia-XPM nor open objects have a formal definition and semantics. Finally, YAWL also offers extensibility capabilities using Proclets [27].

3. COORDINATION ELEMENTS: OPEN OBJECTS

This section presents the coordination model that is the core contribution of the paper. This coordination model is based on a basic element called 'open object'. By using several of these open objects it is possible to build extensible and executable models that have all their elements synchronized. In this section the elements that form an open object are first presented; then the mechanisms that allow the coordination of several open objects are discussed, as well as their extensibility capabilities; finally, the requirements to implement a system based on open objects are discussed.

3.1 Structure of an Open Object

One of the advantages of the object oriented paradigm is the capacity to materialize in a model the elements of a problem, their behavior, and their relations. Object-based models usually replicate the structures of the problem domain, using object attributes to recreate relations and method calls to model interaction. In order to take advantage of the capacity of building isomorphic structures to the problem domain, our composition elements are based on objects, but they have some additional features that expose their internal state. That is why we call them 'open objects'.

In the traditional object-oriented paradigm, a state of an object is a particular combination of values of its attributes. To know this state, it is necessary to call its methods. The number of states reachable by an object depends on the values that its attributes can have. However, most of the time, the elements that interact with an object are interested only in a subset of its reachable states and some of the possible states of the object can be grouped together to create a simpler abstraction of its life cycle. This does not mean that attribute-based states should be eliminated. Instead, these new, broader states can be materialized in an external state machine synchronized with the object. For instance, in the case of a counter from 0 to 100, one possible abstraction can identify only three different states: "Stopped," "Counting," and "Finished." This reduction in complexity simplifies monitoring and coordination since the state machine can easily publish notifications when a state change occurs and this can serve to coordinate other elements.

An open object is composed of an *entity*, a *state machine* associated to the entity, and a set of *actions*. An entity is just a traditional object with attributes and methods: it provides an attribute-based state to the open object, and its methods are a place where part of its behavior can be implemented. The state machine materializes an abstraction of the life-cycle of the entity, allowing other elements to know this state and react to its changes. This can be done using methods defined in the interface of the open object, which is based on the interface of the entity, and is enriched with the methods needed to access and navigate the state machine. Finally, the actions are pieces of behavior that are associated to transitions: when a transition is processed, its actions are executed in a serialized way.

3.2 Coordination of Open Objects

The execution of an open objects-based model depends on the execution and coordination of its elements. Because of this, open objects offers two different interaction mechanisms: one is asynchronous and based on events, and the other is synchronous and based on actions and method calls. These two alternatives complement themselves and can be used to describe very complex interaction patterns.

[2] Woflan, URL: http://is.tm.tue.nl/research/woflan/

Events are the most important coordination mechanism in Cumbia because they are used to maintain the synchronization not only between open objects but also between entities and their respective state machines. This section explains how events are processed and used to keep state machines updated, but first it is necessary to discuss what generates events. From the standpoint of an open object, events can be produced by its own entity, by its state machine, or by external elements, which can be other open objects or even elements external to the model. An entity usually generates an event when one of its methods is called and changes its internal state. Thus, the event is generated to inform the state machine about the change and try to maintain the consistency between the internal, attribute-based state of the entity, and the current state of the state machine. Events can also be generated when state machines change state: each time a state machine moves from its current state to the next, events are generated to show that (*i*) the original state is abandoned, (*ii*) the processing of a transition starts, (*iii*) the processing of a transition finishes, and (*iv*) a new state is reached (see Figure 1)[3]. Finally, events can be generated by other sources, such as external systems or because of user interaction. Cumbia does not differentiate those events, and they are treated exactly like internal events.

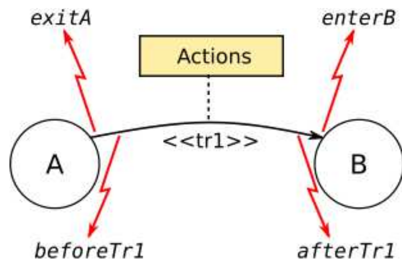

Figure 1: Events generated by state machines.

Every transition in a state machine has an associated expected event description. This means that whenever an event that matches the description is received, the transition has to be taken. Events are described with an expression of the form **[ELEMENT]eventType**, where **[ELEMENT]** describes who is expected to generate the event, and **eventType** specifies the particular type of event expected. Note that **ELEMENT** is not the identifier of a specific element, but a relative reference that can be used to locate it. For example, if an open object used in a hierarchical structure has an event described as generated by **[PARENT]**, then the generator of the event can be located ascending in the hierarchy. If an event is described as generated by **[ME]**, then the generator is the same open object that owns the state machine (see Figure 2).

Events are processed by open objects in the following way. First, events are received and stored in a queue that is local to the open object. Events are then processed one by one, until the queue is empty. Events' processing in multiple open objects happens in parallel within our current implementation of the Cumbia kernel (see section 5): a separate thread is used to process the events of each open object. However, a valid alternative implementation is to use a single thread to process every event, using some algorithm to select queues, or even using a shared queue for all open objects. The selection of one of these alternatives does not have an impact on the execution semantics; nevertheless, there may be an impact on non-functional requirements such as efficiency or scalability.

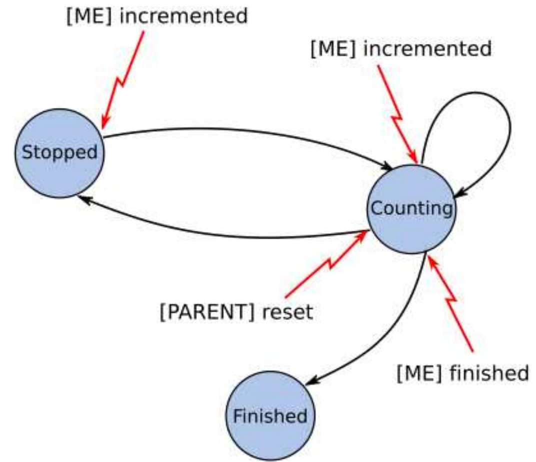

Figure 2: State machine with source events.

To process an event, it is necessary to analyze the transitions that start in the current state of the open objects' state machine. If the event matches the description of the expected event for any of those transitions, (*i*) that transition is taken, (*ii*) its associated actions are executed, and (*iii*) the state machine changes state. If the event does not match the expected event for any of the transitions that start from the current state, the event is discarded and it is never processed again[4]. If there are more than one transition with matching expected events, one of them is randomly selected[5].

The other coordination mechanism of open objects is based in actions and method calls. In general, actions are used to add semantics to state's changes and produce some kind of effect on the model. In particular, actions can be used to call entities' methods, to communicate with other applications, and to generate further events. By combining these two mechanisms of interaction it is possible to define the coordination of open objects with very fine granularity, in a synchronous or asynchronous fashion. Furthermore, these coordination rules are defined externally to the open objects' implementation, and they are thus easily modifiable, even at execution time.

[3] While the transition takes place, the state machine is considered to still be in the original state (A).

[4] This strategy was selected because it reduces the possibility of having disordered events. Although discarding events may appear as a source of non-determinism, it forces metamodels' developers to be more careful in their designs. Furthermore, it changes a very difficult problem (processing disordered events) for a simpler one (missing events that can be requested again).

[5] This situation should be considered an error made by the designer of the state machine. Although it is possible to introduce this kind of non-determinism, it should be avoided if possible.

3.3 Extension Mechanisms

Open objects offer three extension mechanisms which give flexibility to models and metamodels. These mechanisms have different capacity of expression, and they can be combined if necessary.

The first mechanism involves the modification of actions associated to a state machine. This method, called *simple extension*, can be applied to the definition of an element, and affect all its instances, or it can be applied at runtime to affect a single instance. This mechanism is the simplest to use, but it is not as powerful as the other two. The following snippet of code shows an example of the usage of this mechanism to modify an element of the metamodel.

```
<extended-type name="ExtendedPort" extends="Port">
    <extension transitionName="Receive">
       <action name="Store Data"
               class="cumbia.actions.StoreData"/>
    </extension>
</extended-type>
```

The code extends an existing element called `Port`, and creates an extension called `ExtendedPort`. The code specifies that the new, extended type is similar to the base type, but has an additional action called `Store Data` associated to the transition `Receive`. The new action is implemented in the class `cumbia.actions.StoreData`.

This mechanism can also be applied at runtime, to modify a running instance. However, instead of using an xml to describe the extension, the open objects' API is used to add the desired action.

The second mechanism involves modifications to the structure of the state machine that add or remove states and transitions. This extension mechanism is much more expressive than the previous, as it allows deeper changes to the behavior of the models. Using it, it is possible to alter the abstraction of the life-cycle of the entity, thus changing the way in which other open objects relate to it. The usage of this mechanism should take into account the risk of creating new elements that will not synchronize properly with the existing ones.

The third and last extension mechanism allows the creation of new open objects by specializing existing ones. This mechanism requires modifications to the implementation of the entities to change the part of behavior of open objects that is not expressed with a state machine.

The three mechanisms presented cover a wide range of the possible extension requirements that can surface in a workflow application: they can be used to accommodate small changes to the behavior of an element, but they can also be used to introduce totally new elements and adjust the others to it. Finally, these extension mechanisms also offer alternatives that vary in their expressiveness and their complexity: they were designed to address different requirements and have specific expression capacities that determine how much of the behavior of the elements can be modified. For instance, two of the mechanisms can be used to extend the open objects behavior, but do not require modifications to the implementation of the entities. Because of this, a developer that uses open objects has to take these two factors into account when selecting which mechanism to apply.

3.4 Usage and Implementation

In order to use the open objects, it is necessary to build metamodels based on them. As presented on the introduction, the idea is to have a metamodel for each dimension in a workflow application. Metamodels are built with the following steps: first, it is necessary to identify the elements that should be part of the metamodel, and establish their relationships, attributes and behavior. Then, for each element a state machine has to be designed. In this step, special attention should be put on the interactions between elements and on the required actions. Finally, the metamodels and their specialized open objects have to be implemented: the attributes and methods of the open objects are described in the entities' code; then, the state machines are described using an xml-based language; then, the actions used by the state machines are implemented; as a last step, a textual description of the metamodel is created, naming the open objects included in it, specifying their entities and state machines, and declaring their relationships. Most of these steps are supported by the Cumbia editor that is presented in section 5.

The most important tool that we have implemented for Cumbia is what we called the *Cumbia Kernel*. In the first place, the Kernel understands metamodel descriptions and is capable of managing them. The Kernel is also capable of understanding model descriptions written with an xml-based syntax. Using this information, the Kernel can create instances of the models; this requires the instantiation of open objects following the definitions included in the metamodel. Finally, the Kernel is also capable of supporting the execution of the open objects: it offers an interface to interact with any open object, and manages the reception and distribution of events.

Although the Kernel provides most of the common behavior shared by every metamodel, the Kernel is rarely used without any modification. Instead, an *engine* is usually developed for each metamodel. These engines are always based on the Kernel; thereby it is not necessary to re-implement any of the functionalities offered by it. What most engines provide is behavior specific to the metamodel, and interfaces to interact with the models. For instance, the BPEL [28] engine described in section 5, offers specific behavior to manipulate BPEL data, and an external API based on web services.

4. OPEN OBJECTS IN THE WORKFLOW CONTEXT

4.1 Cumbia-XPM

Cumbia-XPM is a metamodel constructed with open objects, designed to describe the control dimension of workflow applications. This means that each element in the metamodel (Figure 3) is a specialized open object, with a specific entity and state machine. Because of the centrality of the control dimension in workflow applications, Cumbia-XPM models are usually called the *process description*.

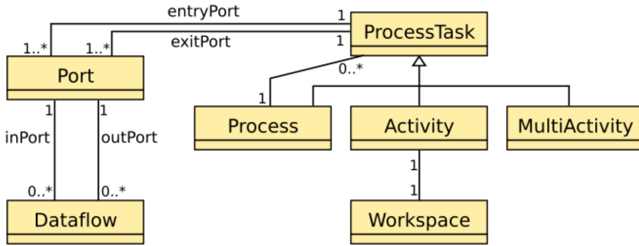

Figure 3: Cumbia-XPM Metamodel.

A Cumbia-XPM model has a hierarchical structure that has a *Process* at its root. To be executed, a process needs some entry products that are consumed when tasks are executed. These products are called initial data and are received through one of the entry ports of the process. When that process finishes its execution, it generates a set of data that can be recovered through one of its exit ports. To model this, we have defined an element called *Port* that can play the roles of entry or exit port. In both cases a port is given a set of data that has to be delivered to someone who needs it: in the case of an exit port of the process, the data is picked up by someone or somebody external to the process; in the case of an entry port, the data is picked up by one of the elements that is part of the internal structure of the process.

Executing a process means performing, in order, a set of either simple or complex (sub-processes) tasks. Each task of a process requires also initial data and produces results, thus tasks also need entry and exit ports. The internal structure of a process defines the execution order of the tasks and depends on *Dataflows*, which connect ports. A dataflow connected to an exit port of a task receives the results that it produces and makes them flow to an entry port of the next task, carrying along both data and control. Figure 4 depicts a sample process with a structure of ports and dataflows that clearly defines the control-flow for the process. In this image it is possible to see how splits, joins and loops can be achieved. For example, after `Receive Request` the activities `Consult Credit Rating`, `Evaluate Request` and `Study Credit History` are executed in parallel. `Make Decision` can be executed after the completion of `Consult Credit Rating`, `Evaluate Request` and `Study Credit History`.

The atomic tasks executed within a process are modeled in Cumbia-XPM with *Activities* and *Workspaces*. An activity controls the execution of a task and is responsible for managing the data it requires and the data it produces. Workspaces are encapsulated inside activities and are responsible for executing specific tasks. A workspace interacts only with the enclosing activity, which provides the data needed for its execution: when the data reaches one of the entry ports of the activity, it is given to the workspace to start its execution; when the workspace finishes its work, it generates an event, which the activity uses to pick up the data, reset the workspace (so that it can be executed again), and finally put the data in one of the exit ports. When one of those exit ports becomes full, the flow of control and data continues through the dataflows.

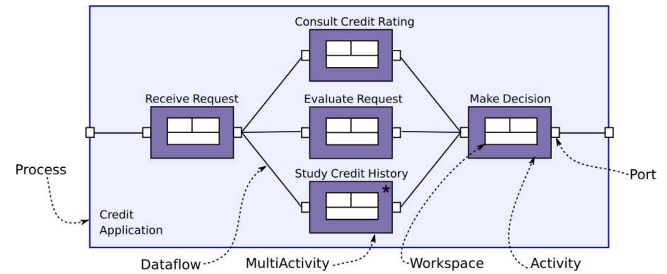

Figure 4: Elements of Cumbia-XPM in a sample process.

In the sample process there are five activities: `Receive Request`, `Evaluate Request` and `Make Decision` are activities expected to be made by a user; `Consult Credit Rating` is expected to be executed automatically by a workspace specialized in consuming web services. There are cases where it is necessary to execute in parallel several instances of the same activity. Cumbia-XPM has an element called *MultiActivity* to achieve that. Every time a normal Activity gets data from its entry ports, the corresponding workspace is executed once. In a MultiActivity, several instances of the workspace are created and executed in parallel when the initial data required is received. The number of workspaces executed can be defined during the design of the process, or during runtime, and new workspaces of a MultiActivity can be created in a dynamic way. In the sample process, several people should execute the activity `Study Credit History` at the same time, but the exact number is only known during the execution of the process.

Since Cumbia-XPM is based on the open objects' model, its extension mechanisms (simple extension, adaptation, and specialization) can be applied to it. For instance, to build a system that orchestrates applications, workspaces can be specialized to give them the ability to invoke web services. Thus, extended versions of Cumbia-XPM can be easily created.

4.2 Open Objects' Interaction in Cumbia-XPM

To clarify how the interaction between open objects and between elements of Cumbia-XPM proceeds, we now present a simplified scenario that includes only three elements: an activity that is going to be executed; an entry-port of that activity, which will receive the data that is needed to start the execution (entry-data); and a workspace that is inside the activity and will be responsible for executing a specific task using the entry-data. The action in this scenario begins when the entry-port receives the data it was expecting; the activity then picks the entry-data, feeds and activates the workspace, and waits until it finishes its execution. An activity can have several entry-ports which can receive data concurrently; however, when one of those ports gets full, the activity waits until the workspace finishes its execution before verifying if there is any other full entry-port.

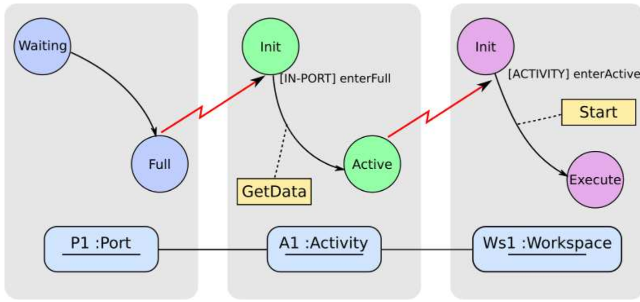

Figure 5: Partial interaction between a port, an activity and its workspace.

The three state machines depicted in Figure 5 correspond to fragments of the state machines of a Port, an Activity and its Workspace. These three state machines were designed to be composed: some of the events that one of them generates are expected by the others to continue their execution. We now describe the interaction between those state machines that starts when the port receives data.

1. The port P1 receives entry-data and takes the transition from state Waiting to state Full.

2. The state machine of activity A1 receives the event *enterFull* and takes the transition to state Active, executing the **getData** action, which retrieves the data from the port and gives it to the Workspace.

3. The state machine of workspace Ws1 receives the event *enterActive* generated by the Activity's state machine. The transition to state Execute is taken, executing the action **start**, which initiates the execution of the workspace.

This interaction sequence is repeated every time an activity instance starts its execution.

5. CURRENT IMPLEMENTATION

Besides the *Cumbia Kernel*, we have also implemented some support applications. The most important ones are an editor for metamodels and a testing platform. As shown in Figure 6, the editor provides support for the definition of open objects' specializations, and also for the definition of relationships between open objects. After metamodels are fully defined, the editor generates metamodel descriptions that are understandable for the kernel, and also generates the code templates that developers need to fill in order to specify all the metamodel-specific behavior.

The Cumbia's testing platform is what we use to perform automatic testing of every Cumbia-based engine. This platform proposes a very powerful structure to model test cases, and it can be used to create specialized test frameworks for each engine. Since Cumbia applications are highly concurrent, a lot of synchronization issues appear when things are verified during execution; furthermore, since elements in the Cumbia-XPM metamodel are strongly interrelated, they are not susceptible to be tested independently. To solve these problems we developed a testing framework that allows the definition, execution and verification of testing scenarios. Scenarios are described by a static structure (a Cumbia-XPM process), instructions to control its execution and some assertions that have to be verified after its execution. The information to validate the assertions is gathered from execution traces in a fashion similar to what is done in [8,9].

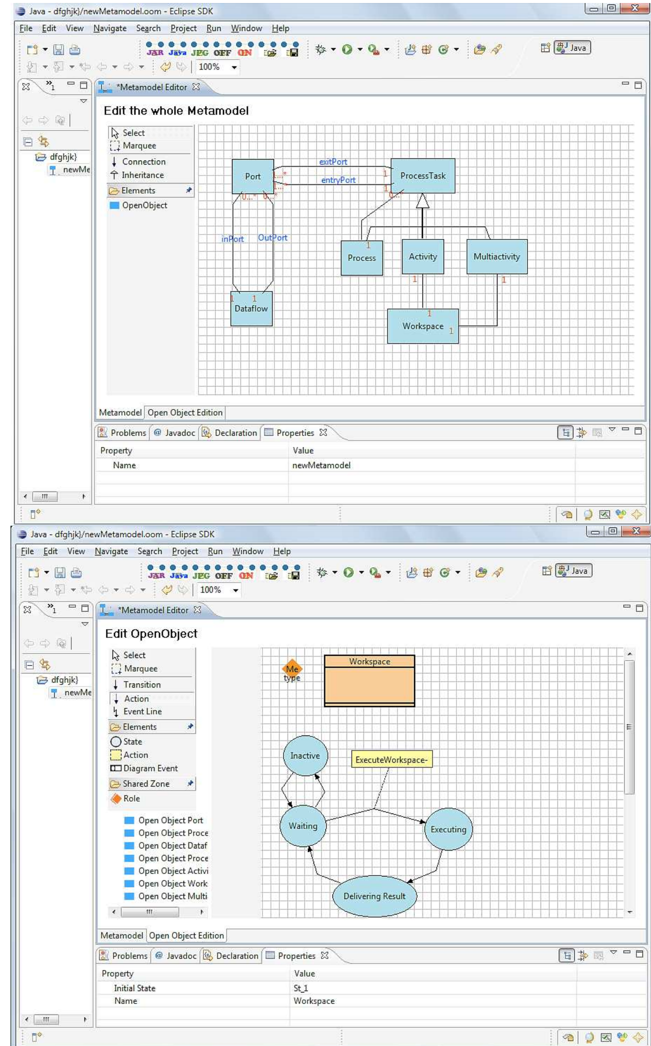

Figure 6: Metamodels' Editor while editing a) a metamodel and b) an open object.

Using the Cumbia platform, we have developed several metamodels with their respective engines. We have centered most of our efforts in Cumbia-XPM because of the centrality of the control dimension in workflow applications. Because of this, we have also developed an editor to graphically design Cumbia-XPM processes (see Figure 7). We have also defined other metamodels to handle dimensions of workflows that are tightly related to control. *Cumbia-XRM* (eXtensible Resources Model) was designed to model the resources used in the execution of a process (working personnel or machines, for example) and is also capable of modeling complex rules to assign these resources to tasks. *Cumbia-XTM* (eXtensible Time Model) can be used to model time

restrictions over the execution of a process. For instance, one restriction can specify that if a set of activities lasts more than three days, then an email has to be sent to the supervisor. Each of these metamodels has its own engine, and XTM and XRM models are composed to Cumbia-XPM processes using synchronized state machines [23]. Thus, the coordination methods used between open objects inside XPM are also used to synchronize elements from several metamodels. The relationships between elements of different models are specified externally and an application that knows all the engines is responsible for creating the necessary linkages. This external definition also makes it is easy to later add new models or modify their relationships.

Using Cumbia, we have also implemented two different versions of a BPEL engine: the first one used Cumbia-XPM as an intermediate language, whereas for the second one we modeled the BPEL metamodel and we built a specific engine for it. The first version of the engine was based on a transformation approach: first, we extended Cumbia-XPM with specialized workspaces to consume web services and handle xml data, and then we translated BPEL process definitions into semantically equivalent Cumbia-XPM definitions. Furthermore, we had to create a wrapper for the Cumbia-XPM engine in order to make it offer a BPEL compatible interface. Although this implementation was usable, it presented a problem that can always appear when translations are made, that is the problem of reversing the translation. This reverse translation is necessary, for instance, if someone is going to query the process status and is interested in receiving it in BPEL terms and not in Cumbia-XPM terms. In order to solve this problem in this implementation, we used a technique based on traceability information stored when the translation was made.

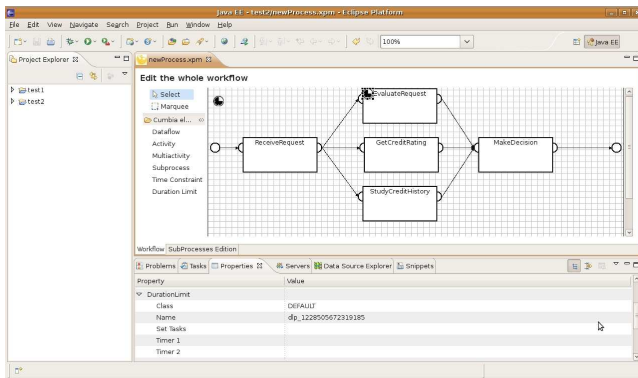

Figure 7: Cumbia-XPM's Editor.

Using the Cumbia platform and Cumbia-XPM, we also developed an application called *PaperXpress*. PaperXpress is a tool to support the collaborative writing of research papers. It allows the definition of ad-hoc processes, and it offers the support needed to coordinate tasks and to handle the results of the writing process. This application also applies the ideas of separation of dimensions; therefore it includes dimensions such as control, data and resources.

Another application that we have developed is an engine for IMS-LD [10]. Although technically this specification is used to define *learnflows* instead of *workflows* [14], the Cumbia platform was very suitable to build this application. Finally, we are currently developing a container for SCA [24] assemblies. The goal of this container is to provide a testing and analysis platform for SCA based solutions. Thus, through the usage of open objects the container exposes a lot of useful information about the internal state and behavior of the components.

6. CONCLUSIONS

This paper presented a coordination model based on synchronized state machines, and it showed the advantages of applying it to workflow applications. The paper also showed that the main problem that hinders maintenance and evolution in current workflow engines is the lack of capacity to separate dimensions. The direct consequence of this is that workflow languages tend to become big and complex. Consequently, the engines that run these languages, and the processes described with them, also become inflexible and difficult to maintain and evolve. These problems can all be solved if the Cumbia platform is used to build the workflow engines.

The Cumbia platform has the following characteristics. In the first place, it separates the dimensions involved in a workflow and uses *dimension specific languages* to describe them. This is possible, because each dimension is described with a *metamodel* that is described in terms of a coordination model based on synchronized state machines. The coordination elements of this model are *open objects*, and they have powerful coordination capabilities. These open objects also offer extension mechanisms that contribute to the extensibility of processes and languages.

Another characteristic of the approach is its uniformity, which contributes to its ease of use, and also facilitates reuse. Uniformity in Cumbia can be seen in the following points:

1. Every dimension expresses its behavior using the same coordination model. Thus, each dimension can be accessed and manipulated using the same mechanisms. This also makes it possible to have a common kernel that offers the most important functionalities; thus, engines for each dimension can be easily developed as extensions to the kernel.
2. The same methods of the coordination model that are used inside a model are also used for the composition between dimensions. Thus, open objects do not need to offer two different systems of coordination and their complexity is reduced.
3. The result of composing dimensions keeps the same properties of the individual dimensions: it is still an executable model and it is possible to coordinate other models with it.

The characteristics of Cumbia have direct consequences on the languages. Since they are used to describe only single dimensions, they can be simpler and more specific. As a result, they are easier to use, maintain and evolve. Furthermore, dimensions can evolve independently, without a significant impact on the other dimensions. Finally, the composition and coordination mechanisms make it possible to add or modify dimensions even at run time.

Another advantage of the approach is the potential of lowering development time: basic dimensions such as control or time can be reused, adapted and composed with a fine granularity and without limiting their applicability to other applications. However, new dimensions, or extensions to existing dimensions, can be developed easily using the editors that we have developed.

Finally, the Cumbia approach can also be used in other types of applications. This paper focused only on workflow applications, but the advantages offered by Cumbia can also be useful in more general contexts.

7. ACKNOWLEDGMENTS

We would like to thank the rest of the Cumbia group for their hard work and their contributions to this research. In particular, we would like to thank Camilo Jiménez, Fabio Quimbay and Diana Puentes for their efforts in the development of the editors.

8. REFERENCES

[1] Axenath, B., Kindler, E., Rubin, V. 2007. AMFIBIA: a meta-model for integrating business process modelling aspects. In *International Journal of Business Process Integration and Management* 2, 2 (2007), 120–131.

[2] Arbab, F., Mavaddat, F. 2002. Coordination through channel composition. In *COORDINATION '02: Proceedings of the 5th International Conference on Coordination Models and Languages* (London, UK, 2002). Springer-Verlag, 22–39.

[3] Braem, M. et al. 2006. Isolating process-level concerns using Padus. In *BPM 2006*. LNCS, 4102, Springer-Verlag, 113–128.

[4] Brogi, A., Canal, C., Pimentel, E. 2007. Behavioural types for service integration: Achievements and challenges. In *Electronic Notes in Theoretical Computer Science*, 180, 2, Elsevier, 41–54.

[5] Charfi, A., Mezini, M. 2006. Aspect-oriented workflow languages. In *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*. LNCS, 4275, Springer Berlin / Heidelberg, 183–200.

[6] Craven, N. and Mahling, D. 1995. Goals and processes: a task basis for projects and workflows. In *Proceedings of Conference on Organizational Computing Systems* (Milpitas, California, United States, August 13 - 16, 1995). N. Comstock and C. Ellis, Eds. COCS '95. ACM, New York, NY, 237-248.

[7] Diakov, N.K., Arbab, F. 2004. Compositional construction of web services using reo. In *Web Services: Modeling, Architecture and Infrastructure - Proceedings of the 2nd International Workshop on Web Services: Modeling, Architecture and Infrastructure*, WSMAI 2004, INSTICC Press, 49–58.

[8] Dwyer, M.B., Avrunin, G.S., Corbett, J.C. 1999. Patterns in property specifications for finite-state verification. In *ICSE '99: Proceedings of the 21st International Conference on Software Engineering*, Los Alamitos, CA, USA, IEEE Computer Society Press, 411–420.

[9] Hu, Z., Shatz, S.M. 2006. Explicit modeling of semantics associated with composite states in uml statecharts. In *Automated Software Engineering*, 13, 4, 423–467.

[10] IMS Learning Desing, version 1, February, 2003. http://www.imsglobal.org/learningdesign/

[11] Joohyun Han, Yongyun Cho, Jaeyoung Choi 2005. Context-Aware Workflow Language Based on Web Services for Ubiquitous Computing. In *ICCSA 2005*. LNCS, 3481, Springer 1008–1017.

[12] Kiepuszewski, B. 2003. Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows. PhD thesis, Queensland University of Technology (Brisbane, Australia).

[13] Limniotes, T. A., Papadopoulos, G. A., and Arbab, F. 2004. Web Services: separation of concerns: computation coordination communication. In P*roceedings of the 2004 ACM Symposium on Applied Computing* (Nicosia, Cyprus, March 14 - 17, 2004). SAC '04. ACM, New York, NY, 492-497.

[14] O. Mariño, R. Casallas, J. Villalobos, D. Correal, J. Contamines 2007. Bridging the Gap between E-learning Modeling and Delivery through the Transformation of Learnflows into Workflows. In *E-Learning Networked Environments and Architectures*, Springer.

[15] Object Management Group: Software Process Engineering Metamodel (SPEM), Version 1.1 (January 2005)

[16] Papadopoulos, G. A. and Arbab, F. 1998 Coordination Models and Languages. Technical Report. UMI Order Number: SEN-R9834., CWI (Centre for Mathematics and Computer Science).

[17] Papadopoulos, G. A. and Arbab, F. 1998. Modelling activities in information systems using the coordination language MANIFOLD. In *Proceedings of the 1998 ACM Symposium on Applied Computing* (Atlanta, Georgia, United States, February 27 - March 01, 1998). SAC '98. ACM, New York, NY, 185-193.

[18] Russell, N., ter Hofstede, A.H.M., van der Aalst, W.M.P. and Mulyar, N. 2006. Workflow Control-Flow Patterns: A Revised View. In *BPM Center Report,* BPM-06-22, BPMcenter.org.

[19] Russell, N., ter Hofstede, A.H.M., Edmond, D. and van der Aalst, W.M.P. 2004. Workflow Resource Patterns. In *BETA Working Paper Series*, WP 127, Eindhoven University of Technology, (Eindhoven, The Netherlands).

[20] Russell, N., ter Hofstede, A.H.M. , Edmond, D. and van der Aalst, W.M.P. 2004. Workflow Data Patterns. In *QUT Technical report*, FIT-TR-2004-01, Queensland University of Technology (Brisbane, Australia).

[21] Sadiq, S., Orlowska, M., Sadiq, W., Foulger, C. 2004. Data flow and validation in workflow modelling. In *ADC '04: Proceedings of the 15th Australasian database conference* (Darlinghurst, Australia). Australian Computer Society Inc. 207–214

[22] Sánchez, M., Villalobos, J., Deridder, D. 2008. Co-Evolution and Consistency in Workflow-based Applications. In *1st*

International Workshop on Model Co-Evolution and Consistency Management (Toulouse, France).

[23] Sánchez, M. and Villalobos, J. 2008. A flexible architecture to build workflows using aspect-oriented concepts. In *Proceedings of the 2008 AOSD Workshop on Aspect-Oriented Modeling* (Brussels, Belgium).

[24] Service Component Architecture - Assembly Model Specification, version 1.0, March, 2007. http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications

[25] Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M. (Eds.). Workflows for e-Science: Scientific Workflows for Grids. Springer, 2007.

[26] van der Aalst, W., ter Hofstede, A. 2003. Yawl: Yet another workflow language (revised version). In *QUT Technical report*, FIT-TR-2003-04, Queensland University of Technology (Brisbane, Australia).

[27] van der Aalst, W. M., Barthelmess, P., Ellis, C. A., and Wainer, J. 2000. Workflow Modeling Using Proclets. In *Proceedings of the 7th international Conference on Cooperative information Systems* (September 06 - 08, 2000). LNCS, 1901, Springer-Verlag, London, 198-209.

[28] Web Services Business ProcessExecution Language, Version 2.0, April 2007, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[29] WS-BPEL Extension for People (BPEL4People), Version 1.0, June 2007.