# Integrating Light-Weight Workflow Management Systems within Existing Business Environments

Peter Muth, Jeanine Weissenfels, Michael Gillmann, Gerhard Weikum
University of the Saarland
{muth,weissenfels,gillmann,weikum}@cs.uni-sb.de, http://www-dbs.cs.uni-sb.de

## Abstract

*Workflow management systems support the efficient, largely automated execution of business processes. However, using a workflow management system typically requires implementing the application's control flow exclusively by the workflow management system. This approach is powerful if the control flow is specified and implemented from scratch, but it has severe drawbacks if a workflow management system is to be integrated within environments with existing solutions for implementing control flow. Usually, the existing solutions are too complex to be substituted by the workflow management system at once. Hence, the workflow management system must support an incremental integration, i.e. the reuse of existing implementations of control flow as well as their incremental substitution.*

*Extending the workflow management system's functionality according to future application needs, e.g. by worklist and history management, must also be possible. In particular, at the beginning of an incremental integration process, only a limited amount of a workflow management system's functionality is actually exploited by the workflow application. Later on, as the integration proceeds, more advanced requirements arise and demand the customization of the workflow management system to the evolving application needs.*

*In this paper, we present the architecture and implementation of a light-weight workflow management system, coined Mentor-lite, which aims to overcome the above mentioned shortcomings of conventional workflow management systems. Mentor–lite supports an easy integration of workflow functionality into an existing environment, and can be tailored to specific workflow application needs.*

## 1 Introduction

When a workflow management system is used to implement a computer supported business process from scratch, all the functionality of the workflow management system can be fully exploited. In particular, the control flow will be completely specified in the workflow management system's specification language, and the invocation of workflow activities according to the specified control flow will be completely controlled by the workflow engine at execution time. In particular, external applications that are called by the activities of a workflow can be integrated by means of a simple invocation interface (e.g. using CORBA or DCOM). This is what virtually all currently available workflow management systems provide and the workflow management coalition (WfMC) proposes an architecture along these lines [39].

### 1.1 Problem Statement

However, only in rare cases there is an opportunity to computerize a business process from scratch. Typically, computer support for business processes has been implemented before workflow management systems are introduced. This is not only true for the application programs implementing the activities of a workflow, but also for the control flow. Examples are shell scripts for invoking application programs according to the outcome of other applications, timer-controlled invocations of applications, stored database procedures in combination with trigger definitions, or even programs in high-level programming languages that access databases and invoke other applications. In the following, we refer to such existing control flow implementations as *scripts*. Since existing business environments typically mix scripts and activities, they quickly become very complex to maintain and to adapt to new application needs. Workflow management systems provide a solution, but using them typically requires all control flow to be reimplemented. The costs of such reimplementation are often prohibitive. Even more important, initial problems with the new implementation are very likely to occur and may cause the entire application environment to be unavailable for a long time. Only if a smooth transition from the old "hand-made" workflow to a workflow management system supported implementation is possible, workflow management systems will gain more acceptance for usage in existing business environments. A smooth transition requires the incremental substitution of scripts by facilities of the workflow management system. These incremental steps can be performed whenever the corresponding part of the business process needs to be modified or extended anyway. From a technical point of view, this requires means to incorporate scripts into the specification language of the workflow management system, and it requires interfaces from the workflow management system to the scripts.

Even if the required features of the specification language and the required interfaces exist, the usage of workflow management systems is often prohibited by their huge system footprint. This is particularly important if only a small part of their functionality is (currently) exploited. Used in such a way, current workflow management systems are too expensive, create too much overhead or simply do not perform well enough.

To make things worse, even the most full-fledged workflow management systems suffer from limited extensibility. Although most systems were designed to support diverse kinds of workflow applications, optional administration facilities like worklist management and history management are often not powerful enough and can not be sufficiently tailored to application needs. The problem is that the requirements posed on these components by different workflow applications are as diverse as the workflow applications themselves. For example, the need for complex strategies for assigning work items to actors, such as load balancing between actors and the principle of dual control, can not be predicted in advance. In essence, administration facilities are small workflows themselves, and require support of almost all components of the workflow management system. This observation is not properly reflected in most architectures of workflow management systems today, as administration facilities are not considered as an integral part of the workflow management system.

### 1.2 Contribution of the Paper

Based on the above observations, the need for a review of current architectures of workflow management systems is obvious. A new architecture must support an easy, incremental integration of

This work was performed within the research project "Architecture, Configuration, and Administration of Large Workflow Management Systems" funded by the German Science Foundation (DFG).

workflow management systems into existing environments, must consider tailorable administrative facilities as an integral part, and must aim at minimizing the system footprint.

We advocate a light-weight system architecture that is based on a "slim" workflow management kernel, yet can be adapted to the full spectrum of applications through appropriate extensions. We argue that protocols for integrating scripts as well as optional workflow administration functionality should themselves be implemented as workflows. This novel technique distinguishes our approach from earlier proposals towards light-weight and adaptable architectures. However, there is always a tradeoff between a simple kernel architecture with limited functionality and the benefit of sharing functionality of a rich kernel. The challenge is to clearly identify the required kernel functionality and its interfaces.

### 1.3 Related Work

The problem of designing interfaces to existing applications has challenged system designer long before research on workflow management began. In the area of database integration, so called *mediators* [40] were developed. The aim of mediators is to make data stored in different data sources, which even use different data models, available in a homogeneous manner. These data sources need themselves be prepared for the integration by surrounding each of them with an appropriate wrapper. The approach involves basic interface design, means for implementing dataflow, translation mechanisms, and even data repositories for maintaining meta data describing the various data sources. The most notable projects in this area are the IBM Garlic project [30], the DISCO project [21,26], and the TSIMMIS project [16,22]. In contrast to the needs of workflow management systems, however, these projects focus on connecting data sources, not on connecting application programs in general.

Many applications today use common web standards such as HTML to solve interoperability problems. This approach has also been adopted by workflow management systems. DartFlow [6] and Staffware Global [34] use web pages with embedded Java applets as its front end. The interface of Panta Rhei [10] to client applications and to remote systems is solely based on the exchange of HTML forms. However, these approaches address only the client issues of interoperability, not the connection to existing scripts and applications.

For the invocation of external applications, distributed programming environments like CORBA or DCOM [13] can be used. They make distribution transparent and even provide basic transaction services. For example, CORBA is used by the OrbWork [33] workflow enactment service as part of the Meteor$_2$ project. In Meteor$_2$, an alternative enactment service, WebWork [23] solely based on web technology has also been developed. In Lotus Notes [29], OLE2 capabilities are used to integrate application programs. However, in Lotus Notes many different interfaces and languages like agents, macros and Lotus-Script can also be used to expand interface functionality.

Work closely related to our approach is done in the MOBILE project. There, existing (legacy) application programs are integrated into the MOBILE workflow management system [19] by means of a specific wrapping mechanism for application programs [12,32]. The emphasis is on reliability and recovery aspects. Therefore, the authors distinguish between transactional and non-transactional application programs. A similar approach is also pursued in Meteor$_2$ by considering different task types.

None of the above work considers the integration of existing scripts, i.e., making them available as building blocks for workflow applications. Hence, to the best of our knowledge, our approach to incrementally integrate existing scripts both in terms of basic interfaces as well as in terms of the workflow specification language is a novel idea and has not been considered elsewhere.

In addition to the interface problem, tailoring of workflow management systems to specific workflow application needs has become an important issue. However, most workflow management systems, both products and research prototypes, are rather monolithic and aim at providing full fledged support for the widest possible application spectrum [1,7,9,11,20,37]. On the other hand, the idea of extensible and adaptable system architectures is not new. Extensible database systems have been investigated in research and development for a decade, with mixed success but wide agreement that this is the right architectural paradigm [5]. A similar paradigm is being pursued for constructing highly adaptable, distributed middleware [2, 3]. In the area of workflow management, however, the need for extensible and adaptable architectures has not received much attention so far, with the most notable exceptions being the MOBILE [19,20], Meteor$_2$ [23,33], INCA [4], and WASA [38] projects. Our specific approach towards an extensible, light-weight system, namely, viewing major extension components as workflows themselves, is, to the best of our knowledge, a novel idea.

### 1.4 Outline of the Paper

Based on an example scenario, we reason about the architectural decisions we made for our light-weight workflow management system, Mentor-lite. Discussions of important implementation aspects of Mentor-lite show the feasibility of our approach. The paper is organized as follows. In Section 2, we give a detailed discussion of the architectural requirements based on an example scenario. Section 3 presents the basic architecture of Mentor-lite. Section 4 discusses how Mentor-lite can be integrated into an existing environment. Section 5 concludes the paper.

## 2 Architectural Requirements

Two architectural requirements have mainly driven the design of our workflow management system Mentor-lite. First, it must be easy to integrate Mentor-lite into existing computerized business processes. Second, the development and extension of facilities like worklist management and history management must not be harder than designing a workflow itself.

In order to discuss these requirements, we present a simple workflow scenario in an electronic commerce environment. It considers a large mail order business selling expensive electronic devices such as high-end computers and telecommunication equipment. We will focus on the main business process, i.e. registering incoming orders and delivering the ordered items, for discussing our scenario.

Assume that our mail order business has already computerized parts of the business process. Incoming orders by mail or phone are registered in a database. As high-end electronic devices are expensive, the credit-worthiness of each customer is checked before the order is processed. This is done by automatically consulting a corresponding online service, e.g. by using encrypted e-mail. If the check is successful, the inventory database is checked to see if the ordered items are in stock. If this check is also positive, an invoice is printed and the ordered items are delivered to the customer together with the invoice. The control flow of the business process is implemented by simple scripts and application programs, e.g. using embedded SQL. Such an implementation can be found in many enterprises today where workflow management systems are not (yet) used.

### 2.1 Integrating Control Flow Scripts

Now assume that our mail order business wants to exploit the Internet to present its products. Customers should be able to place orders via the web. In this case, no invoice will be sent. Instead, payment will be done through credit cards. This extension of the main business process requires additional activities to be integrated into the business process, e.g. registering orders that come through the web, and the substitution of existing activities, e.g., checking a customers credit card instead of her credit-worthiness. In addition, modifications of the current control flow scripts are required to incorporate the new activities.

In order to implement the new business process, the mail order business might either choose to modify the existing scripts, or undertakes the adventure of reimplementing its business processes by using a workflow management system. In the first case, the mail order business will face the same problem again if future extensions are necessary. In the second case, all investments in the current implementation of control flow are lost. If dependencies with other business processes of the enterprise exist, all the business processes have to be reimplemented at once. This also includes existing wrappers to application programs that perform activities in the business process. In addition, initial problems with the new implementation are very likely to occur. There is a high risk of not being able to do business for days or even weeks until all transition problems are solved.

This consideration clearly shows that workflow management systems must support a smooth transition from "manually" implemented control flow to control flow handled by the workflow management system. To do so, it is important that a workflow management system is able to integrate the existing implementations for handling control flow. Later, these scripts can be incrementally substituted by the workflow management system.

Most workflow management systems today only have an interface to start existing applications with a set of input parameters and to catch the completion notification plus the output generated by the application. Integrating scripts requires additional interfaces, e.g., for observing those parts of a business process which are controlled by these scripts. In addition, the workflow specification language must also be powerful enough to model the execution of scripts and to reflect their execution states. This observation leads to our first architectural requirement:

(1) *A stepwise integration of workflow management functionality into existing environments must be supported. This requires the integration and stepwise substitution of existing control flow implementations.*

In case of our mail order business scenario, this means that the workflow management system must coexist with the script which checks the credit-worthiness of customers, checks the inventory database and finally initiates the delivery of the ordered items to the customer. The workflow management system will be responsible for the extensions of the workflow, i.e. for registering orders made on the web and for handling the payment by credit card.

## 2.2 Seamless Integration of System Extensions

There are several components of workflow management systems with highly workflow application dependent requirements on their functionality. These include worklist management, workflow monitoring, history management, and even interfaces to other workflow management systems. Consider, for example, the worklist manager. The major task of the worklist manager is to assign work items to actors based on a specific assignment strategy, also known as "role resolution". Depending on the workflow application, assignment strategies can become very complex (e.g. principle of dual control, load balancing). Monitoring of workflows provides administrators with information about the current state of a workflow. The kind and the level of detail of this information strongly depends on the workflow application. The same holds for the history of workflows, which can be considered as an aggregation of monitoring information. If different workflow management systems are involved in the execution of a workflow, interfaces between these systems are required. If the involved systems do not support a standard interface, if a common interface does not provide enough functionality, or if applications using the interface do not perform well enough, application specific interfaces have to be used.

It is easy to see that general purpose components for the above facilities will be complex to design and will increase the footprint of workflow management systems considerably. This is a major drawback, as in most cases only a subset of their functionality is actually exploited. Hence, a workflow management system should provide means for implementing system extensions such that they can be seamlessly integrated into the system. This leads to our second architectural requirement:

(2) *A workflow management system must facilitate the implementation and seamless integration of system extensions. Applications must not suffer from runtime overhead or large system footprints caused by system extensions unless their functionality is actually exploited.*

In the next section, we show how these requirements together with other, well known conventional requirements for workflow management systems are reflected in the architecture of Mentor-lite.

# 3 Mentor-Lite Architecture

In this section we present our approach to a light-weight and tailorable workflow management system. It is based on the Mentor workflow management system [24,25,42,43], but aims at a simpler architecture. The main goal is to provide only kernel functionality inside the workflow engine, and consider system components like history management and worklist management as extensions on top of the kernel. The key point to retain the light-weight nature is that these extensions are implemented as workflows themselves. An invocation interface for application programs as well as an interface for integrating scripts is provided. We have coined our approach Mentor-lite. We first present the component architecture of Mentor-lite, followed by a brief introduction of the Mentor-lite workflow specification language, namely state and activity charts.

## 3.1 Component Architecture

As shown in Figure 4, the basic building block of Mentor-lite is an *interpreter* for workflow specifications based on state charts. Two additional components, the *communication manager* (*ComMgr*) and the *log manager* (*LogMgr*) are closely integrated with the workflow interpreter. All three components together form the *workflow engine*, which in our case is identical to the workflow kernel.

Like in Mentor, Mentor-lite supports a protocol for a distributed execution of workflows. The communication manager is responsible for sending and receiving synchronization messages between the engines. These messages contain information about locally raised events, updates of state chart variables and state information of the local engine [25]. When a synchronization message is received, the corresponding updates at the receiving site are performed. In order to guarantee a consistent global state even in presence of site or network faults, we currently use the TP-Monitor Tuxedo [28, 35] for delivering the synchronization messages within queued transactions. In the near future, we plan to substitute the Tuxedo facilities by CORBA and object transaction services [8].

The log manager provides logging facilities and recovery mechanisms. A separate *workflow log* is used at each site where a Mentor-lite workflow engine is running. Databases like the *workflow repository* (i.e. a repository of workflow specifications) or the *worklist database* can be shared by Mentor-lite workflow engines at different sites.

Application programs and scripts are connected to the workflow engine by specific *wrappers*. Details of this interface will be presented in Section 4.

Workflow specifications are given in terms of state and activity charts, the specification language already used in Mentor. The interpreter performs a stepwise execution of the workflow specification according to its formal semantics. For each step, the activities to be performed by the step are determined and started. In contrast to Mentor, which initially included a code-generation phase, specifications in Mentor-lite are interpreted at runtime and can therefore be dynamically modified.

Application dependent facilities like *worklist management, history management* and *monitoring* are implemented on top of the engine as state and activity charts. Hence, they are interpreted by the workflow interpreter just like any other workflow specification. A detailed example for implementing worklist management is given in [41], and for history management in [45].

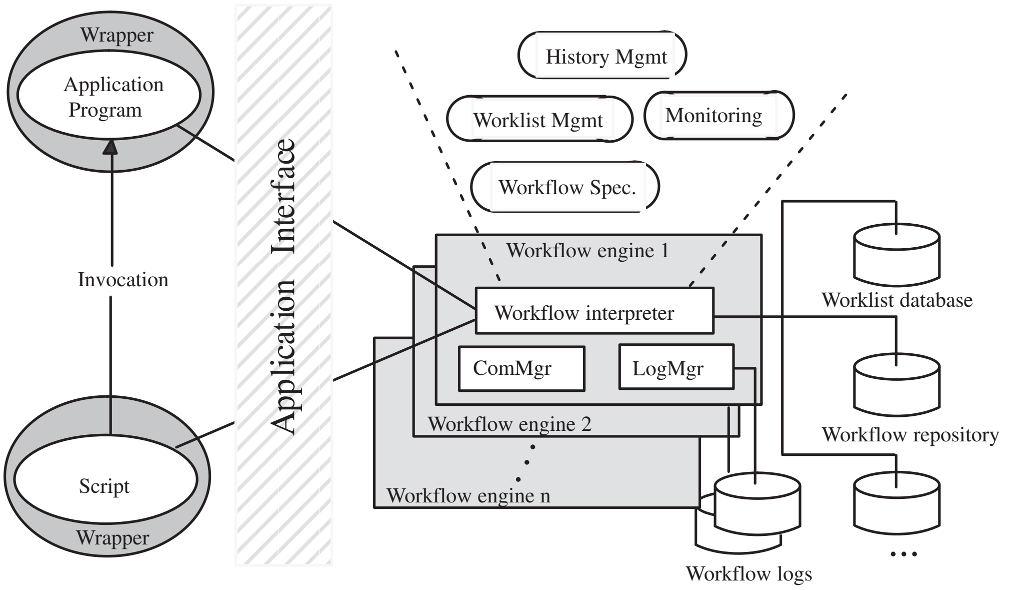


***Figure 4: The Mentor-lite architecture***

## 3.2 State and Activity Charts

In this section we briefly describe the formalism of state and activity charts [14, 15, 17] as used in Mentor–lite. This specification formalism has been adopted for the behavioral dimension of the UML industry standard [36]. State and activity charts comprise two dual views of a specification.

Activities reflect the functional decomposition of a system and denote the "active" components of a specification; they correspond directly to the activities of a workflow. An activity chart specifies the data flow between activities, in the form of a directed graph with data items as arc annotations.

State charts capture the behavior of a system by specifying the control flow between activities. A state chart is essentially a finite state machine with a distinguished initial state and transitions driven by Event-Condition-Action rules (ECA rules). A transition from state *X* to state *Y*, annotated with an ECA rule of the form *E[C]/A*, fires if event *E* occurs and condition *C* holds. The effect is that state *X* is left, state *Y* is entered, and action *A* is executed. Conditions and actions are expressed in terms of variables, for example, those that are specified for the data flow in the corresponding activity chart. In addition, an action *A* can explicitly start an activity, expressed by *st!(activity)*, and can generate an event *E* or set a condition *C* (e.g. *fs!(C)* sets the condition *C* to false). Each of the three components of an *E[C]/A* triple may be empty. Every state change in a state chart execution is viewed as a single step; thus, state changes induce a discrete time dimension.

Important additional features of state charts are nested states and orthogonal components. Nesting of states means that a state can itself contain an entire state chart. The semantics is that upon entering the higher-level state, the initial state of the embedded lower-level state chart is automatically entered, and upon leaving the higher-level state all embedded lower-level states are left. The capability for nesting states is especially useful for the refinement of specifications during the design process and for inserting predefined macros. Orthogonal components denote the parallel execution of two state charts that are embedded in the same higher-level state (where the entire state chart can be viewed as a single top-level state). Both components enter their initial states simultaneously, and the transitions in the two components proceed in parallel, subject to the preconditions for a transition to fire.

# 4 Integrating Applications with Mentor-lite

In this section we discuss our approach on how to integrate application programs within Mentor-lite. We distinguish between application programs which do not invoke or coordinate any other applications and scripts which implement existing control flow and call application programs themselves.

## 4.1 Interfaces to Application Programs

In most other workflow management systems, interfaces to application programs have to be completely implemented inside of application wrappers. In Mentor-lite, only basic communication interfaces are implemented by wrappers, e.g. using the distributed computing environment CORBA [27]. On top of these interfaces, protocols for complex interactions with application programs are specified in terms of state and activity charts. Being designed as a specification language for reactive systems, state and activity charts are particularly suited to serve this purpose. As a further advantage, we use the same language for specifying workflows and interface protocols. This way, interface protocols can be specified by workflow designers without the need to use other programming environments, e.g. for implementing interface protocols in C.

In Mentor-lite, each activity defined in an activity chart is tagged with its invocation method, e.g. remote procedure call (RPC), operating system call, or CORBA call. When interpreting the workflow specification, for each activity to be executed, a function implementing the invocation of the activity is called. As an example, we show the Mentor-lite CORBA interface for controlling application programs. It is realized using Orbix [18], a CORBA implementation. For each application program, a proxy object identifying its wrapper has to be created. The corresponding function looks like this:

*createProxy(const char* AppName)*
*{ ... p = _bind(:WrapperAppName) ; ... }*

The *_bind()* statement creates the proxy object for the wrapper of the application *AppName* in the workflow engine's address space. Further, it returns a handle (*p*) to this object.

The application wrapper must enable the workflow engine to start and stop the application program and to exchange information with the application. For example, the application programs should be able to raise events and set state chart variables and, in turn, the

```
interface Wrapper {
      void start();
      void stop();
      short get(in string data, out string results);
      short put(in string data);
}
```

***Figure 5a:*** **Generic interface definition for application wrappers in IDL**

```
class WrapperApp: public WrapperBOAImpl {
      void start(CORBA::Environment &IT_env) { ... system(App); ...};
      void stop(CORBA::Environment &IT_env) { ... };
      CORBA::Short get(const char* data, char *& results, CORBA::Environment &IT_env) { ... };
      CORBA::Short put(const char* data, CORBA::Environment &IT_env) { ... };
}
```

***Figure 5b:*** **Implementation of an application specific wrapper**

application programs should be able to receive updates of state chart variables.

The generic interface of application wrappers is shown in Figure 5a. Four functions constitute the interface:

- The *start()* function invokes the application. In the example implementation in Figure 5b the application "App" is started by issuing an operating system call.
- The *stop()* function for terminating an application.
- The *put()* function to transfer information from the WF engine to the application.
- The *get()* function to transfer information from the application to the WF engine.

The WF engine can access all functions of the wrapper implementation by using the handle returned by the *_bind()* call. For example, the WF engine can ask the wrapper to start the application *App* by calling *p->start()*. While the interface definition is generic, its implementation is different for each application, i.e. there is a different implementation class of the wrapper interface for different applications. This of course requires the application program to offer the desired functionality in its application programming interface (API).

As mentioned above, this set of functions is used as a building block for implementing complex invocation and interaction protocols between the Mentor-lite kernel and application programs in terms of state and activity charts. Generic state charts, e.g. for enabling an actor to start, stop, suspend and resume an activity, form a library of invocation protocols which can be used in each workflow specification.

## 4.2 Interfaces to Scripts

The approach of providing only a simple wrapper and specifying complex interface protocols in terms of state and activity charts is also used for integrating existing scripts. A Mentor-lite library contains state and activity chart templates which model generic steps for integrating existing scripts, e.g. templates for modelling the interception of application calls issued by scripts. This transfers the responsibility for invoking applications to Mentor-lite and enables the extension of existing control flow without reimplementing everything from scratch. Our mail order business scenario will serve as an example to illustrate different steps of the integration.

In the first step, we define a state and activity chart for each script. The activity chart consists of a single activity representing the script, the state chart consists of a corresponding state and an initial transition which starts the activity. In order to link the activity to the script, a simple wrapper providing an invocation interface is required. Together, the state and activity chart specify a workflow consisting only of the existing script. Given this simple form of integration, we have three options to continue:

(1) The workflow specification is extended by inserting new activities and the corresponding control flow. In this case, the script is called as part of an extended workflow in the same way as one atomic activity and remains unchanged.
(2) Calls to application programs issued by the script are rerouted to the workflow management system kernel by intercepting the application invocation. This way, the script is decoupled from the activities it invokes. The activities can be substituted by other ones, even by complete subworkflows. The new activities and subworkflows are invoked in place of the original activities. Using dummy activities that implement a null action makes parts of the script ineffective. This option provides for an incremental substitution of scripts by state and activity charts.
(3) Scripts are substituted by state and activity charts, i.e., completely rewritten from scratch. This is the most comprehensive, but also the most expensive approach.

In the following, we discuss how options (1) and (2) are used to extend an existing control flow implementation in our mail order business scenario. Using option (3) usually causes too much overhead and is disregarded in this paper.

As described in Section 2.1, our mail order business wants to exploit the Internet. Customers should be able to place orders via the internet, payment will be done through credit cards. This extension requires additional activities to be invoked, and existing activities to be substituted by new ones.

The new workflow is specified in terms of state and activity charts, still making use of the existing script which registers incoming orders in a database, checks the credit-worthiness of the customer, checks an inventory database to see if the ordered items are in stock, and prints an invoice. Figure 6 shows the activity chart, with the script represented by activity *Script.* There are four additional activities, namely the *WebOrder* activity, the *ElectronicBilling* activity, the *SendResults* activity, and the *CheckCreditCard* activity. *WebOrder* and *ElectronicBilling* extend the original workflow according to option (1). *WebOrder* inserts orders placed by using a web-interface into the database, such that they are available to *Script. ElectronicBilling* performs the payment by credit card. Activity *SendResults* will be explained later. The forth activity, *CheckCreditCard*, substitutes the original check of the customer's credit-worthiness. Its integration requires that the original call to the credit-worthiness check by the script is intercepted and redirected towards the credit card check. The handling of the credit card check should be controlled by Mentor-lite. This extends the workflow according to option (2).

In Figure 7, the corresponding state chart is shown. For better readability, we use the suffix "*_S*" for all state identifiers. Whenever a web order arrives the order is placed in the order-database by activity *WebOrder*, which is started in the initial transition to state *WebOrder_S*. The existing script, i.e., activity *Script* is started after the activity *WebOrder* is finished, i.e., when the condition *WO_OK* evaluates to true and the nested state *Script_S* is entered. The nesting of this state is necessary to conceptually split the

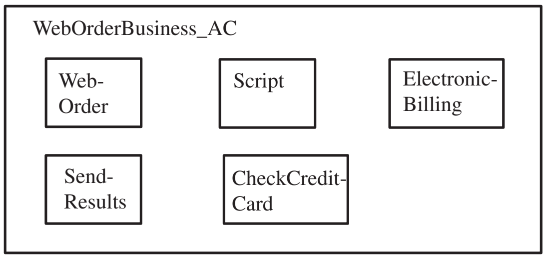


***Figure 6:*** **The activity chart for the web order business extension**

script into two parts. The first part, represented by state *ScriptA*_S, corresponds to the existing control flow before issuing the check for credit-worthiness, while the state *ScriptB_S* is entered after the check is done. Instead of calling the original check for credit-worthiness, the activity *CheckCreditCard* is invoked when the transition to state *CheckCreditCard_S* fires. After the script is finished, indicated by condition *Script_DONE*, activity *ElectronicBilling* is started.

Note that the script, represented by activity *Script*, is not modified at all, i.e. the code of the script remains unchanged. But one of its calls to application programs, the credit-worthiness check, is intercepted and redirected to the workflow engine. To show how this is implemented, we have to go into more detail about invocations of wrappers and their communication with the workflow engine when executing the part of the state chart which corresponds to the nested state *Script_S*. Figure 8 shows actions performed by the workflow engine, the script and its wrapper, a substitution program (stub) for the credit-worthiness check, and the new activity *CheckCreditCard.* On the left hand side, the substates of state *Script_S* and the corresponding transitions are shown. On the right hand side, invocations of wrappers and the stub as well as the communication of these wrappers and the stub with the workflow engine is shown. When entering the state *Script_S,* the initial transition to the first substate *ScriptA_S* fires, i.e. the activity *Script* is started. Since all activities that represent application programs are invoked via their wrappers, the start of activity *Script* results in the execution of the wrapper of the script. The wrapper will start the script which proceeds to the call of the credit-worthiness check application. The original behavior was to issue this call and to wait for its completion. Now the call is intercepted by the stub for the credit-worthiness check. It sets the condition *START_CheckCreditCard* to true and passes the calling information for the credit-worthiness check to the workflow engine. Because we want to substitute the credit-worthiness check by a credit card check, the workflow engine now invokes the wrapper for the application program that checks credit cards by starting activity *CheckCreditCard*. After the corresponding application program has successfully finished, its wrapper will set the condition *CheckCreditCard_DONE* to true and pass the condition variable and the return parameters of the credit card check to the workflow engine. When the workflow engine receives the condition CheckCredit*Card_DONE*, the transition from the state *CheckCreditCard* to state *ScriptB_S* fires, and activity *SendResults* is started. This activity is necessary to relay the return values through the workflow engine to the stub, which in turn relays them to the script. Hence, the script does not notice a difference between the execution of the check for credit-worthiness and the check of the credit card. It proceeds with the rest of its code. Finally, the script will finish and its wrapper will set the condition *Script_DONE* to true.

The example has shown how the integration of scripts with workflows specified in Mentor-lite is performed. Note that the interfaces between the workflow engine and wrappers for application programs as defined in Section 4.1 are sufficient to implement the integration. Only a simple stub for implementing the redirection of a call to an existing application program is additionally required. This stub is generic and can be used for redirecting other calls as well. The protocol for using the stub was specified in terms of state and activity charts, showing the feasibility of using state and activity charts to integrate existing control flow implementation.

# 5 Conclusions

In this paper, we have investigated two major obstacles for integrating current workflow management systems into environments with already existing computerized business processes:

- Current workflow management systems are best suited for implementing all control flow from scratch. They provide only limited means for interacting with existing computerized business processes.
- Components like worklist management and history management are usually tailored for specific application scenarios. For other scenarios, they either cause too much overhead or lack required functionality.

In principle, these two problem areas are orthogonal issues. However, we believe that an extensible light-weight workflow management system is a crucial architectural asset for both directions. We argue that the system kernel should only provide basic execution

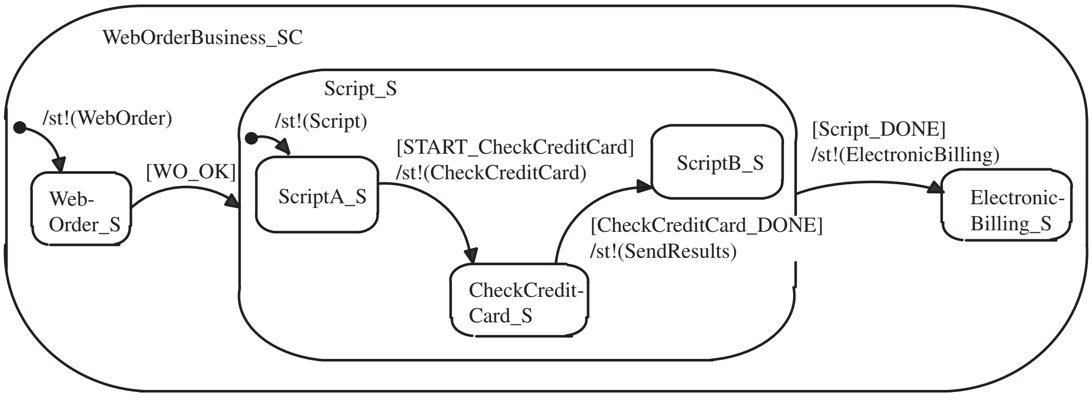


***Figure 7:*** **The state chart for the web order business extension**

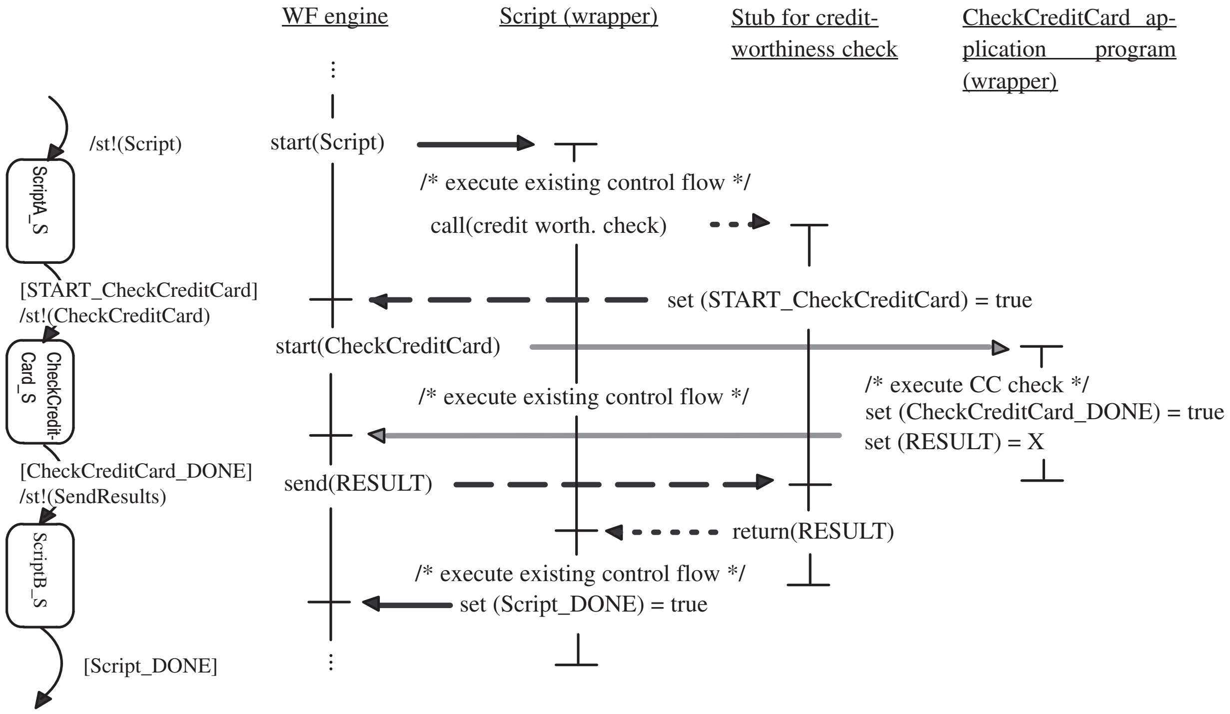


***Figure 8:*** **Protocol between workflow engine and the script**

and communication facilities, whereas solutions to the above problems are implemented in the workflow specification language, i.e., as workflows themselves. We have shown that state and activity charts, our method of choice for specifying workflows, are well suited for implementing specific protocols to integrate existing implementations of control flow in an incremental manner. Optional administration facilities can be specified on top of the kernel as subworkflows. Using this approach, functionality implemented in the kernel, e.g., distributed execution, reliable and fault tolerant execution, encryption etc., is automatically available to all extensions. This allows even a decentralized execution of role resolution strategies for work items or a decentralized history management without additional implementation effort.

In the future, we plan to extend our prototype implementation to get further proofs of concept. We will investigate three different areas. Application programs with interfaces to distributed computing environments like CORBA and DCOM enable workflows to use specific application facilities, far beyond a simple invocation. Using these facilities requires extended wrapping functionality on top of the distributed computing environment. But instead of implementing such extended wrappers in a programming language like C, we will use state and activity charts. Only the basic Mentor-lite kernel interface to the distributed programming environment will be implemented in C. In the second area, we will implement libraries of optional administration facilities such as worklist and history management. We plan to investigate the feasibility of our approach for even more demanding system components such as alerting mechanisms based on time constraints and interfaces to other workflow management systems. Finally, we will investigate the impact of our architectural decisions on the system performance. Our aim is to define simple but realistic benchmarks and to apply them for comparing Mentor-lite with other workflow management systems. A first proposal is presented in [44].

## 6 References


**[1]** G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, Functionality and Limitations of Current Workflow Management Systems, IEEE Expert, Vol. 12, No. 5, 1997

**[2]** G. Alonso, C. Hagen, Geo-Opera: Workflow Concepts for Spatial Processes, in: M. Scholl, A. Voisard (ed.): Advances in Spatial Databases, Lecture Notes in Computer Science, Vol. 1262, Springer, 1997

**[3]** G. Alonso, C. Hagen, H. J. Schek, M. Tresch, Towards a Platform for Distributed Application Development in [9]

**[4]** D. Barbara, S. Mehrotra, M. Rusinkiewicz, INCAs: Managing Dynamic Workflows in Distributed Environments, Journal of Database Management, Special Issue on Multidatabases, Vol. 7, No. 1, 1996

**[5]** M. J. Carey, D. J. DeWitt, Of Objects and Databases: A Decade of Turmoil, Proc. of Int'l Conf. on Very Large Data Bases (VLDB), Bombay, India, 1996

**[6]** T. Cai, P. A. Gloor, S. Nog, DartFlow: A Workflow Management System on the Web using Transportable Agents, Technical Report PCS-TR96-283, Dartmouth Collage, 1996

**[7]** A. Cichocki, A. Helal, M. Rusinkiewicz, D. Woelk, Workflow and Process Automation - Concepts and Technology, Kluwer Academic Publishers, 1998

**[8]** CORBA Services Available Electronically, http://www.omg.org/corba/csindx.htm

**[9]** A. Dogac, L. Kalinichenko, M. Tamer Ozsu, A. Sheth (Eds.), Workflow Management Systems and Interoperability, NATO Advanced Study Institute, Springer, 1998

**[10]** J. Eder, H. Groiss, W. Liebhart, The Workflow Management System Panta Rhei, in [9]

**[11]** D. Georgakopoulos, M. Hornick, A. Sheth, An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure, Distributed and Parallel Databases, Vol. 3, No. 2 , 1995

**[12]** R. Günthör, S. Jablonski, Transaction-Based Application Integration in Workflow Management Systems, Proc. of the Int'l Conf. on Database and Expert Systems Applications (DEXA), Zurich, Switzerland, 1996

**[13]** R. Grimes, Professional DCOM Programming, WROX Press, Birmingham, 1997

**[14]** D. Harel, State Charts: A Visual Formalism for Complex Systems, Science of Computer Programming, Vol. 8, 1987, pp. 231-274

**[15]** D. Harel et. al, Statemate: A Working Environment for the Development of Complex Reactive Systems, IEEE Transactions on Software Engineering, Vol. 16, No. 4, 1990

**[16]** J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. Breunig, V. Vassalos, Template-Based Wrappers in the TSIMMIS System, Proc. of ACM SIGMOD Conf. (Demonstration), Tucson, Arizona, 1997

**[17]** D. Harel, A. Naamad. The STATEMATE Semantics of State Charts, Technical Report, i-Logix Inc., October 1995

**[18]** Manuals of Orbix 2.2, IONA Technologies PLC, 1997

**[19]** S. Jablonski, MOBILE: A Modular Workflow Model and Architecture, Proc. of Int'l Working Conf. on Dynamic Modelling and Information Systems, Nordwijkerhout, Netherlands, 1994

**[20]** S. Jablonski, C. Bussler, Workflow Management, Modeling Concepts, Architecture, and Implementation, International Thomson Computer Press, 1996

**[21]** O. Kapitskaia, A. Tomasic, P. Valduriez, Dealing with Discrepancies in Wrapper Functionality, 13éme Journées de Bases de Données Avancées, Grenoble, France, 1997

**[22]** C. Li, R. Yerneni, V. Vassalos, H. Garcia-Molina, Y, Papakonstantinou, J. Ullmann, M. Valiveti, Capability Based Mediation in TSIMMIS, Proc. of ACM SIGMOD Conf. (Demonstration), Seattle, Washington, 1998

**[23]** J. A. Miller, D. Palaniswami, A. P. Sheth, K. Kochut, H. Singh: WebWork: $METEOR_2$'s Web-Based Workflow Management System, Journal of Intelligent Information Systems, Vol. 10, No. 2, 1998

**[24]** P. Muth, D. Wodtke, J. Weissenfels, G. Weikum, A. Kotz Dittrich, Enterprise-wide Workflow Management based on State and Activity Charts, in [9]

**[25]** P. Muth, D. Wodtke, J. Weissenfels, A. Kotz Dittrich, G. Weikum, From Centralized Workflow Specification to Distributed Workflow Execution, Journal of Intelligent Information Systems, Special Issue on Workflow Management, Vol. 10, No. 2, 1998

**[26]** H. Naacke, G. Gardarin, A. Tomasic, Leveraging Mediator Cost Models with Heterogeneous Data Sources, Proc. of Int'l Conf. on Data Engineering (ICDE), Orlando, Florida, 1998

**[27]** OMG, The Common Object Request Broker: Architecture and Specification, Rev. 2.0, Technical Report, Object Management Group, 1995

**[28]** F. Primatesta, TUXEDO, An Open Approach to OLTP. Prentice Hall, 1994

**[29]** B. Reinwald, C. Mohan, Structured Workflow Management with Lotus Release 4, Proc. of IEEE Computer Society Int'l Conf., Santa Clara, California, 1996

**[30]** M. T. Roth, P. Schwarz, Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources, Proc. of Int'l Conf. on Very Large Data Bases (VLDB), Athens, Greece, 1997

**[31]** A. Sheth (ed.), Proceedings of NSF Workshop on Workflow and Process Automation in Information Systems, Athens, Georgia, 1996,
http://lsdis.cs.uga.edu/activities/NSF-workflow/proc_cover.html

**[32]** H. Schuster, S. Jablonski, P. Heinl, C. Bussler, A General Framework for the Execution of Heterogeneous Programs in Workflow Management Systems, Proc. of Int'l Conf. on Cooperative Information Systems (CoopIS), Brussels, Belgium, 1996

**[33]** A. Sheth, K. J. Kochut, Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems, in [9]

**[34]** Staffware Global White Paper, Staffware PLC, 1997, http://www.staffware.com/home/whitepapers/data/globalwp.htm

**[35]** Tuxedo System 5, System Documentation, Novell, 1994

**[36]** Unified Modeling Language (UML) Version 1.1, http://www.rational.com/uml/documentation.html

**[37]** G. Vossen, J. Becker (Eds.), Business Process Modelling and Workflow Management, Models, Methods, and Tools, International Thomson Publishing (in German), 1996

**[38]** G. Vossen, M. Weske, The WASA Approach to Workflow Management for Scientific Applications, in [9]

**[39]** Workflow Management Coalition, http://www.wfmc.org

**[40]** G. Wiederhold, Mediators in the Architecture of Future Information Systems, IEEE Computer Magazine, March 1992

**[41]** J. Weissenfels, P. Muth, G. Weikum, Flexible Worklist Management in a Light-Weight Workflow Management System, Proc. of EDBT Workshop on Workflow Management Systems, Valencia, Spain, 1998

**[42]** J. Weissenfels, D. Wodtke, G. Weikum, A. Kotz Dittrich, The MENTOR Architecture for Enterprise-wide Workflow Management. In: [31]

**[43]** D. Wodtke, J. Weissenfels, G. Weikum, A. Kotz Dittrich, The MENTOR Project: Steps Towards Enterprise-Wide Workflow Management, Proc. of Int'l Conf. on Data Engineering (ICDE), New Orleans, Louisiana, 1996

**[44]** M. Gillmann, P. Muth, G. Weikum, J. Weissenfels, Benchmarking of Workflow Management Systems (in German), Proc. of Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), Freiburg, Germany, 1999

**[45]** P. Muth, J. Weissenfels, M. Gillmann, G. Weikum, Workflow History Management in Virtual Enterprises using a Light–Weight Workflow Management System, Proc. of Int'l Workshop on Research Issues in Data Engineering (RIDE), Sydney, Australia, 1999