ABSTRACT. This paper presents TERRA and ACQUA3 two mechanisms that support adaptable definition and execution of workflows. Given the specification of a process as a workflow (activities, flow, agents and behaviour) TERRA generates a graph representing the workflow. Such a graph is used for verifying structure correctness and termination of a workflow specification. Once verified, TERRA generates and stores structures (objects, events, reactions) necessary for executing a workflow. ACQUA executes workflows according to a parametric behaviour model that represents workflow execution policies in terms of dimensions associated to values. ACQUA is supported by event based communication mechanisms for interacting with agents that participate in the execution of workflows activities. Finally, TERRA and ACQUA provide mechanisms for modifying the structure of a workflow and its associated behaviour.

KEYWORDS: Computer Science, e-commerce, workflow technology, flow diagram.

1. Introduction

Today most applications are cooperative. Workflow technology provides useful solutions for specifying and implementing applications that enable automatic execution of business processes such as virtual enterprises, medical, control and e-commerce applications. A business process consists of activities with a common objective. Activities that are executed by agents (human or software components) cooperate for achieving the global process. For example, figure 1 shows the process implementing an e-commerce application.

---

1 Laboratory Logiciels Systèmes Réseaux, LSR-IMAG, 681, rue de la Passerelle, BP. 72, 38402, Saint Martin d’Héres Cedex, France - www-lsr.imag.fr
2 Centro de Investigación en Tecnologías de Información y Automatización, UDLAP, Universidad de las Américas, Sta. Catarina Mártir, , Cholula, Puebla, 72820, México - www.udlap.mx/~centia
3 The results obtained in this work belong to the Database group NODS of Laboratory LSR-IMAG, France. The implementation experience is conducted under the direction of the NODS group and with partial resources of the Database Technology Group, CENTIA-FUDLAP, which only has use rights.
The process is not only described by its activities but also by their execution order (i.e., sequential, in parallel) and data types that they exchange. According to the example, to choose items, clients first access a catalogue (activity AccessCatalogue) that describes products (e.g., characteristics, properties, and price)\(^4\).

Then, an order (ProcessOrder) is specified. If chosen products are available, in parallel, the payment process is started (Authorization and Billing), and a package is prepared (PreparePackage) and the stock is updated (StockUpdate). Finally, the package is delivered to the client (Deliver). The account number, the catalogue, chosen products are data that flow among activities. A workflow is the computerized representation of a business process. It specifies the various activities of a business process that have to be executed in some order, the flow of data between activities and the collaborating agents that execute activities to fulfil a common objective. Figure 2 illustrates the workflow process corresponding to the electronic purchase application.

\(^4\) Activities are numbered to indicate the order of their executions.
or change input data of the activity ProcessOrder. These requirements underline the need of mechanisms that support adaptable and flexible definition and execution of workflows.

Adaptable workflow management

Workflow technology has evolved and provides models and systems [12, 4, 16, 11, 18, 7, 1] and techniques that enable workflow evolution and reuse [21, 17, 20, 19, 5]. From our point of view, the first step needed to introduce adaptability to workflows is the orthogonal specification of definition and execution aspects. Under this hypothesis it is possible to define:

- A workflow structure as the composition of three orthogonal elements: activities, order and synchronization and agents. In this way, the same group of activities can have different associated orders and synchronizations and different agents. This implies that the activities of a process are ordered and synchronized differently according to the policies of the enterprise that implements it. Similarly, such activities can be executed by different agents according to the organizational structure of the enterprise where it is executed.

- Policies adopted for executing a workflow (scheduling, worklist management, activities association to agents). Parameterizing policies, it is possible to assume that the same workflow can be executed according to different policies [2].

Besides this separation strategy, it is necessary to define mechanisms that enable the modification of a workflow structure and its associated execution policies. Such mechanisms cooperate for implementing adaptable workflow management and build an adaptable workflow management system (AWFMS).

Main contributions

This paper presents the implementation experience of TERRA and ACQUA, two mechanisms that respectively support adaptable definition and execution of workflows.

- TERRA an infrastructure for defining adaptable workflows. The system implements a knowledge model [2] that provides concepts for representing workflows (applications) as a set of ordered and synchronized activities that require and produce data and that can be executed by agents (existing applications). Furthermore, given a process defined as a workflow, an execution model, that specifies its execution behaviour, must be defined. Finally, TERRA provides verification mechanisms that test well formed workflows and termination.

- ACQUA is an infrastructure for executing adaptable workflows. The system implements a parametric behaviour model [2] that defines dimensions associated to values. Dimensions represent policies that can be adopted for executing workflows. For example, activities scheduling, association of activities to agents, worklist managements. Distributed workflow execution is done by coordinating autonomous agents under a message (events) based communication model. ACQUA provides interfaces for modifying behaviour models associated to workflows by changing predefined values associated to parameters. The system implements strategies for processing modifications. Such strategies define whether changes are persistent or not and modifications scope (i.e., current workflow executions, next executions).

Our work contributes with mechanisms that enable adaptable, flexible and distributed execution of business processes using workflows. Generally speaking, the approach adopted can be summarized as follows. Each process is specified as a workflow (activities, flow, agents and behaviour). TERRA generates a graph representation used for verifying its structure and termination. Once verified TERRA generates and stores structures required for executing the workflow (objects, events, reactions).
Given an explicit request, ACQUA retrieves the definition of a workflow and builds an execution plan taking into account its associated behaviour model. Assuming that agents are available, the systems execute the workflow according to adopted policies. Both infrastructures are implemented on a JAVA platform, particularly, using the Java Message Service (JMS) for the event based communication. The object oriented database management system FastObjects is used for storing workflows.

Paper organization

The remainder of the paper is organized as follows. Sections 2 and 3 respectively describe the architecture and main functions of TERRA and ACQUA and discuss implementation issues. Section 4 illustrates the use of both systems through a validation experience. Section 5 compares our work with other systems that provide adaptability to workflows. Finally, Section 6 concludes the paper discussing our main contributions, the originality of our work and research perspectives.

2. TERRA

Figure 3 presents an overview of the approach adopted by TERRA for defining workflows. In general, given a workflow specification, TERRA verifies the specified workflow and generates an executable workflow composed by a set of objects, event types (messages) and Event Condition Action rules.

![Workflow definition in TERRA](image)

Figure 3. Workflow definition in TERRA
General architecture

Figure 4 shows the general architecture of TERRA consisting of four main modules that implement workflow specification, verification and generation.

**Specification module** provides an interface that provides formularies for specifying workflow elements according to the knowledge model implemented by the system. Based on a specification, the module creates a graph that represents the specified workflow structure.

**Verification module** verifies workflow structure correctness and termination. If the workflow is not a well structured or it does not verify a termination property the problem is signalled; otherwise the workflow is sent to the generation module.

**Generation module** generates executable workflows where activities are objects that implement method calls corresponding to agent interfaces; order and synchronization are implemented by ECA rules and communication with agents is materialized by events. The module creates a workflow schema (in the database sense) and stores it in a database (FastObjects).

**Adaptability module** processes structural changes on pre-defined workflows. Changes can be done independently of the workflow execution state. This module interacts with the verification module for testing the correctness of the new workflow definition. The module implements strategies that determine whether changes are taken into account immediately (i.e., current workflow executions) or they are differed until the next execution of the workflow.

Main functions

Workflow definition in TERRA is done in four steps: specification, verification, generation and adaptability. The first three steps are sequential, while the last one is orthogonal and causes the execution of the verification and generation steps.

**Specification.** In this step a workflow is specified as a collection of ordered and synchronized activities. Workflow specification is based on knowledge model [2] that provides concepts for representing workflows. According to the model, an activity can represent a simple task such as the access to a data base;
or a composite task such as determining whether a credit can be authorized.

According to the model, the order associated to activities is expressed with ordering operators (e.g., seq, and(or)-join, and(or)-split); conditions under which an activity can be executed are expressed by pre and post conditions and invariants; activities synchronization are expressed as temporal constraints. The model provides also structures for representing data flow among activities. Given a specification, a Petri net that represents the workflow, is generated.

**Verification.** It is possible to introduce inconsistencies while specifying a workflow. Verification process aims at detecting such inconsistencies before putting the process into execution. In particular, we consider two types of inconsistencies, deadlock and multiple terminations.

The verification process adopted is inspired on [23] and it mainly consists in determining whether operators \( \text{and-split (or-split)} \) and \( \text{and-join (resp. or-join)} \), are balanced. Intuitively, structural verification of a workflow is done on a graph that represents it. A recursive analysis is applied on such a graph for verifying whether every \( \text{and-split (resp. or-split)} \) structure is followed by an \( \text{and-join (resp. or-join)} \). If this condition does not hold, an exception is raised. In this step, workflow structure and termination property are tested using an algorithm proposed by [8, 9]. Formally, we say that a workflow is well structured.

**Generation.** The objective of this step is to generate and store structures needed for executing a workflow including JAVA objects that implement activity execution, ECA rules that implement ordering and synchronization and events that implement communication between a workflow execution engine and agents (see Figure 3).

**Structural adaptability.** This step is independent of the previous and its objective is to process structural changes of workflows. For example, add or delete an activity, modify the order and/or synchronization among activities, modify data flow and the association of agent types to activities. Given an adaptability operation, the systems triggers the verification and generation processes.

Finally, a modification results in the specification or the reconfiguration of a new workflow. In the last case, the programmer must specify whether changes should be considered immediately or until the next execution of the workflow. Immediate modification processing implies the interaction with the execution engine (i.e., ACQUA).

### 3. ACQUA

ACQUA is an adaptable workflow execution engine. The system implements a parametric behaviour model [3] that provides parameters associated to predefined values. Parameters represent policies used for executing a workflow. ACQUA also provides interfaces that can be used for modifying dynamically policies associated to workflow execution.

The execution process consists of three phases: execution plan generation, agent subscription and execution. Given a workflow definition (knowledge and behaviour model) retrieved from a database, the execution plan is generated. Such a plan integrates activities, order and execution policies. In parallel, agents dynamically subscribe themselves to the execution engine. Using the execution plan, the engine interacts with agents for affecting activities and for monitoring activities execution. A global execution state is constructed and maintained for deciding how to continue executing a workflow.
General architecture

Figure 5. General architecture of ACQUA

Figure 5 shows the general architecture of ACQUA consisting of four main modules described as follows. The system provides a user interface for retrieving workflows, storing modified workflows and observing their execution.

**Initialization module** generates an execution plan (Transformation sub module) based on a workflow specification retrieved by the Retrieval sub module. Execution plans are also stored in a database managed by the OODBMS (FastObjects).

**Subscription module** receives agent connection requests and contacts the (AgentManager) sub module for subscribing them. This sub module controls available agents and manages information such as agent types and location.

**Execution module** coordinates the execution plan, manages subscribed agents (i.e., subscribe, assign) and their associated worklists (worklist manager sub module).

**Adaptability module** processes modification requests on policies associated to workflow behaviour. Such requests can be done explicitly by a programmer or they can be triggered by messages (events) received by ACQUA. According to the execution state, the system decides whether to execute the modifications immediately or until the next execution of the workflow. Modifications scope can denote all current workflow executions or a particular one.

**Main functions**

**Execution plan construction.** First, the workflow specification is retrieved from FastObjects: objects implementing the workflow activities, ECA rules implementing the flow (order and synchronization operators) and policies associated to each activity and to the whole workflow. The execution plan is implemented by a graph whose root is the workflow name and the nodes can be activity nodes (terminal nodes)...
and operator nodes (non terminal nodes) that have also information on execution policies.

**Agent subscription.** Agent availability (human/application) is essential in the execution process. ACQUA assumes that agents are autonomous and that they can be connected or disconnected dynamically. For each subscribed agent, the system generates a Listener that acts as a proxy between an agent and the Agent manager of the execution module. Each agent and its associated Listener are subscribed respectively as event producers and consumers in the event service.

**Execution.** The execution starts in response of an explicit request from the user. First the workflow definition (execution plan) is retrieved from the specification database. Then the coordinator instantiates and configures the necessary operators. Figure 6 illustrates the configuration that has been built to handle the execution of the purchase application. Operators communicate among them using asynchronous events and with agents using proxies. They have execution parameters that are configured according to the behaviour model.

![Figure 6. Workflow execution](image)

The execution implemented by ACQUA starts in response of an explicit request of the user. First the workflow is instantiated by retrieving its specification from the database. Then, an execution plan is generated (execution graph) and execution parameters are configured.

For executing an activity, the coordinator, implemented by an operator node (non terminal node), contacts the Agent Manager that chooses a subscribed agent according to agent assignment policies associated to the activity. If an agent is available the Agent Manager assigns the activity and the Worklist Manager inserts it in the agent worklist. Otherwise it notifies the coordinator that decides what to do according to scheduling policies associated to the activity. An activity is executed according to its associated execution policies. A Listener notifies the execution state of an activity to the coordinator so that it can decide how to schedule other activities.

**Behavioral adaptability.** This phase is independent of the previous ones and its objective is to enable changes on execution policies associated to workflows behaviour. A modification request can be done by TERRA or by a programmer. Given an adaptability request related to a workflow, ACQUA analyzes its
“current” execution state to determine whether such a change can be immediate or deferred. According to the modified policy, immediate changes are done on the execution plan (policies associated with activities and the whole workflow) or on policies implemented by the Agent Manager.

4. Experimentation

We are currently conducting an experimentation with TERRA and ACQUA for defining e-commerce processes. In the following we present examples related with the application shown in figure 2.

Specification

Each activity in a workflow is defined by its name, precondition, post condition and invariant, and by its associated data. The following XML document shows the specification to the activity AccessCatalogue:

```xml
<activity>
    <activityName>AccessCatalogue</activityName>
    <data>
        <attributeName>UserName</attributeName>
        <attributeType>String</attributeType>
        <attributeName>ID</attributeName>
        <attributeType>String</attributeType>
        <attributeName>Password</attributeName>
        <attributeType>String</attributeType>
    </data>
    <precondition>''SELECT UserName FROM USERS WHERE USERS.NAME = %''</precondition>
    <postcondition>'''</postcondition>
    <invariant>''connected(UserName)''</invariant>
</activity>
```

In fact, TERRA provides an interface for specifying activities and their associated behaviour by choosing predefined values for a given set of parameters. As shown in figure 7, the activity AccessCatalogue is triggered explicitly, its precondition must be evaluated before its activation, if the evaluation fails the workflow execution is aborted and exception is raised (stop + exception). The activity postcondition must be evaluated immediately after its execution is completed, if it fails, an exception is raised.

![Figure 7. Specification of an activity](image-url)
Order and synchronization modes among activities are expressed with order (sequence, or/and-join, or/and-split) and temporal restrictions (meet, overlap). In our example, activities Authorization and Payment are triggered after ProcessOrder is completed (and-split) and keep a temporal constraint meet. That is to say, they must be executed just after ProcessOrder is completed (see figure 8).

<flow> <triggeringActivity>
    <activityName>ProcessOrder</activityName>
    </triggeringActivity>
    <orderingOp>and-split</orderingOp>
    <triggeredActivity>
        <synchronizationMode>meet</synchronizationMode>
        <activityName>Authorization</activityName>
    </triggeredActivity>
    <triggeredActivity>
        <synchronizationMode>meet</synchronizationMode> <activityName>Billing</activityName>
    </triggeredActivity> </flow>

Note that from the synchronization point of view, an activity A is represented by two intervals _elay[td1,td2] and duration[tD1,tD2] where td2 = tD1. Delay represents the interval during which an activity is ready to be executed but its execution has not started yet. Duration represents the interval during which an activity is being executed. Thus, three instants are relevant to an activity life cycle: (i) trigger(A) denotes the instant at which activity A is ready to be executed but the agent has not already started its execution; (ii) begin(A), denotes the instant at which the execution of activity A starts; (iii) end(A), denotes the instant at which the execution of A is accomplished.5

Figure 8. Order and synchronization specification

Agents organization associated to a workflow is specified by defining, for each agent, its name, position in an organization and role. For example, we can define an agent named 148.149.5.9 with position Bank Server and with roles Authorization and Billing.

5 Recall that the semantics of A1 ... An-1 and-join An is that the execution of A1, …, An-1 trigger An.
Verification

The generation of the workflow ElectronicPurchase results in a graph representing its structure (WorkflowGraph). Such a graph is used for verifying the workflow structure using the algorithm proposed in [8, 9]. Figure 9 illustrates the graph generated using our purchase application. It is an oriented graph whose nodes are either sub-processes or ordering operators, or activities. The root node represents the workflow process.

![Workflow Graph](image)

**Figure 9.** Purchase workflow graph

Without giving details about the verification, observe that there is one beginning activity AccessCatalogue and one final activity Deliver. Furthermore, there is only one And-Split operator that corresponds to an And-Join operator. Therefore the purchase workflow is deadlock free and it terminates once.

Note that if exceptions are raised during the execution of an activity (e.g. pre/post-condition or invariant are not verified) the decision whether to abort or continue the workflow execution is expressed in its behaviour. This hypothesis implies verification.

Generation

For each non terminal node (operator node) in the WorkflowGraph TERRA generates event types, ECA rules and JAVA objects that implement terminal nodes (activity nodes). Using ordering and synchronization information, TERRA generates events of the form: \( _1 = \langle \text{type name: string}, \{< \text{variable name: (real, integer, string)}>\} \rangle \)

For example, the event \( E_1 \) is produced at the end of the execution of the activity ProcessOrder, and has as a context the object representing the activity ProcessOrder and the instant of its termination.

\[ E_1 = \text{EndActivity, \{[activity1: Activity, instant: real]\}} \text{ where activity.name = 'ProcessOrder'} \]

TERRA generates also composite events for handling Join situations. For example, the event \( E_4 \) is produced at the end of the activities Billing and StockUpdate.
Non terminal nodes in the WorkflowGraph (operator node) that implement ordering operators have associated
ECA rules that describe in which conditions the end of the execution of the left hand activity
(TriggeringActivity) will trigger right side (activities) (TriggeredActivity). The type of rule that can be generated
depends on the ordering operator semantics. In our example, ECA rules associated to the operator and-join are:


R.and-split-trigger specifies the fact that at the end of the activities Billing and StockUpdate, represented by
E4.Activity2 and E4.Activity3, trigger the execution of the triggered activity of the and-join operator, i.e. Deliver
provided that the billing has been executed successfully (BillingStatus = true). The WorkflowGraph and generated
structures are stored in FastObjects. The objective is to enable querying existing graphs.

Initialization

As we said before, initialization is the phase during which the execution plan is created. Such a plan is a
graph where the operator node has associated TriggeringActivity and TriggeredActivity. An operator node acts as
coordinator that knows (i) policies associated to the global execution of the workflow and to each activity;
(ii) events and rules associated to each activity. In our example, we have a coordinator node and-split where
policies associated to ProcessOrder, Authorization, Billing correspond to those specified in figure 7 and events
and rules are illustrated in the previous Section.

Once the execution plan has been built ACQUA executes the workflow interacting with agent and worklist
managers using Java Event Service as described in Section 1. ACQUA provides an interface where users
can monitor the execution of a workflow (see figure 10).
5. Related works

In this section we discuss characteristics of approaches and systems similar to the one we propose. WIDE [20] proposes three models to characterize workflows organization, information and process aspects. Similar to us, they aim at modelling workflow aspects in an orthogonal way. However, we remark that the process model merges some knowledge features with behavioural ones. For example, transactional aspects are specified within activities structure.

The REWORK [21] meta-model comprises a set of conceptual tools for the specification of workflow system architecture. Actors composing the system are represented by aggregate components called EOB. Response to evolving environmental requirements is carried out via the modification of corresponding system architecture.

[17] proposes a workflow meta-model for supporting dynamic workflows evolution and reuse. A taxonomy of operations that can be applied for modifying workflows is specified. Particular operations are defined for creating new workflows from existing ones and for migrating to certain workflow version (older or newer). Trigsflow[13,15,16,14] is based on a reactive approach where activities are assigned to agents with respect to their role and/or load. Activities can be delegated to other agents. The workflow execution engine in Trigsflow is embedded within an active DBMS.

In Brokers/Services [22] agents are brokers that execute services. Agents are specified and implemented as reactive components based on the event engine EVE [11]. Agents interact asynchronously (using events) to ensure workflow execution.

CoopWARE [18, 10] focuses on interaction aspects between workflow servers and agents. Different to Brokers/Services they adopt a centralized architecture where the WFMS coordinates agents. Our work is similar to the Brokers/Service in that it uses separate active services for executing workflows. Different to our system, behaviour aspects are specified in a fixed model based on a rule execution model.

6. Conclusions

This paper presented an approach for defining and executing workflow in an adaptable way. In order to provide adaptability workflow definition and execution, similarly to DBMS of the third generation, we adopt a separation principle of the definition and execution aspects. In this way, a workflow can have different agent models and can be executed with different execution policies. In order to validate our hypothesis we specified and implemented TERRA and ACQUA two independent infrastructures that cooperate for managing workflows. The distinctive characteristic of both systems is that they support dynamic structural and behavioural adaptability. Research perspectives concerning TERRA and ACQUA are related to services integration. We are currently conducting an experience for integrating e-commerce and data base services as part of the NODS [6] project.

References

11. A. Geppert, M. Kradolfer, and D. Tombros. EVE, Event Engine for Workflow Enactment in Distributed Environments. Technical Report 96.05, University of Zurich -Switzerland, Department of Computer Science, 1996.