University of Klagenfurt
Department of Informatics
Austria

Technical Report

Walter Liebhart, 1995
email: walter@ifi.uni-klu.ac.at

The Workflow Activity Description Language WADL

Workflow management involves everything from modeling processes up to synchronizing the activities and tasks which constitute a workflow and which are performed automatically by software systems or partially automatic by humans. In particular, management of a workflow includes [GHS95]:

- *process modeling and workflow specification*
- *process reengineering*
- *workflow implementation and automation*

Performing workflow specification requires a *workflow model.* A workflow model typically includes a set of concepts that are usefull to describe processes, their tasks, the dependencies among tasks, and the required roles that can perform the tasks. During workflow specification the workflow designer is faced with the identification and modeling of functional, behavioral, organizational and informational aspects of a complex business process [Jab94].

The aim of this technical report is to focus on the *dynamic (behavioral)* modeling of a workflow process. By dynamic aspects the interdependencies or execution order of activities and tasks in a workflow are investigated. Since *automatic failure handling* (recovery and rollback) is supported in WAMO it is not only important to describe "regular" execution orders but also the behavior of a process in certain failure situations. If an essential activity or tasks fails then a corresponding compensation mechanism is activtated.

In WAMO [EL95], processes are structured hierarchically and the execution order of activities and tasks in the hierarchy are based on explicit control flow specifications. The execution of a hierarchical structured workflow starts at the top-level activity which leads to the execution of the first underlying layer of subactivities. Because of the hierarchic structure the control flow is always related to an activity and its directly subrelated subactivities. Additionally, the relation between a parent and a child activity may be specified as vital (default) or non vital.

To describe the semantics of the different control structures in a hierarchy the following aspects have to be clarified:

- What happens when an activity is started?
  - Which subactivities have to be started and in which order?
- When does a parent activity (control structure) terminate successfully / unsuccessfully?
  - What happens if a vital / non vital (sub)activity fails / succeeds?
    - Concerning the parent activity
    - Concerning other sibling activities
- What happens when an activity is compensated?
  - Which subactivities have to be compensated and in which order?
- What happens when an activity is aborted?
  - Which subactivities have to be aborted and/or compensated and in which order?

For the specification of the dynamic part of a workflow we have developed the simple to use and high-level *Workflow Activity Description Language WADL* which is based on the Workflow Activity Model WAMO (refer to [EL95] [EL94]). The definition and extension of WADL is still in work.

The basic elements of the model are activities, tasks, control structures and transaction specific parameters. The transaction specific parameters support selective use of transactional properties (e.g., atomicity, consistency, isolation and/or durability) for individual tasks, activities or entire workflows.

1 Definition of WADL in Backus-Naur form

```
<WF>          ::=   "definition" <Activity> "end"
<Activity>    ::=   <AID> <A>
<AID>         ::=   letter {letter | digit}
<A>           ::=   <SA>  | <RCA> | <FCA> | <PA> | <Task>
<SA>          ::=   "sequence" ["non vital"]  <Activity>  {["non vital"] <Activity>} "end"
<RCA>         ::=   "ranked-choice" <Activity> {<Activity>} "end"
<FCA>         ::=   "free-choice" <Activity> {<Activity>} "end"
<andPA>       ::=   "and-pa" ["non vital"]  <Activity>  {["non vital"] <Activity>} "end"
<orPA>        ::=   "or-pa"  <Activity>  {<Activity>} "end"
<IA>          ::=   "iteration" "non vital" <Activity>  "end"
<Task>        ::=   <T1> | <T2>
<T1>          ::=   "task" <TID> "(" ["force"] <StornoT1> ")"
                    "task" <TID> "(" "force" ")"
<T2>          ::=   "task" <TID> "(" ["force"] <StornoT2> ")"
<StornoT1>    ::=   "undoable" | "compensatable"
<StornoT2>    ::=   "none" | "critical"
<TID>         ::=   letter {letter | digit}
```

Example:

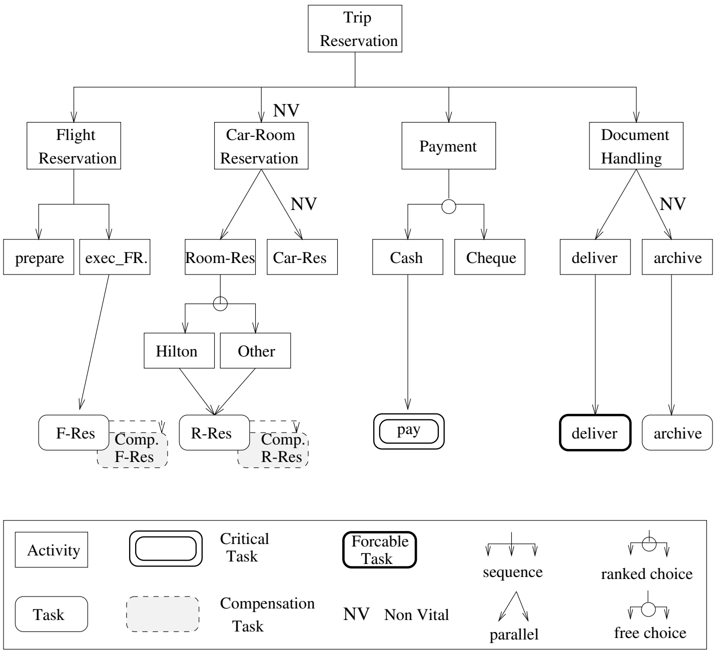

Figure: Trip Reservation

```
definition   Trip_Reservation
    sequence
        Flight_Reservation
            parallel
                prepare     task prepare (compensatable)       task inv_prepare (force)
                exec_FR     task exec_FR (compensatable)       task inv_exec_FR (force)
            end
        non vital Car_Room_Reservation
            paralle
                Room_Res
                    ranked-choice
                        Hilton   task R_Res (compensatable)    task inv_R_Res (force)
                        Other    task R_Res (compensatable)    task inv_R_Res (force)
                    end
                non vital Car-Res ...
            end
        Payment
            free-choice
                Cash    task pay (critical)
                Cheque ...
            end
        Document_Handling
            parallel
                deliver              task deliver (compensatable force) task inv_...
                non vital archive    task archive ...
            end
    end
end
```

2 A Short Overview of ACTA

ACTA [CR92] is a transaction framework which facilitates the formal description of properties of extended transaction models. It allows the specification of transaction types, whereby a transaction type intentionally describes a set of transaction instances that share *structure* and *behavior*. With ACTA, the effects of transactions on other transactions (intertransaction dependencies) and also their effects on objects (visibility of and conflicts between operations on objects) through constraints on histories can be formally specified. Transaction instances issue events, mainly *transaction events* (i.e., begin, commit) and *object events* (i.e., data manipulation events). An event causes a unit of work to switch to a particular state. The following concepts are important:

- A *history H* of the concurrent execution of a set of transactions T contains all the events associated with the transactions in T and indicates the (partial) order in which these events occur.
- The predicate $e \rightarrow e'$ is true if event e precedes event e' in history H. It is false, otherwise.
- $(e \in H) \Rightarrow Condition_H$, where $\Rightarrow$ denotes implication, specifies that the event e can belong to history H only if Condition_H is satisfied. In other words, Condition_H is necessary for e to be in H. Condition_H is a predicate involving the events in H.
- $Condition_H \Rightarrow (e \in H)$ specifies that if $Condition_H$ holds, e should be in the history H. In other words, $Condition_H$ is sufficient for e to be in H. Typically, (parts of) the semantics of one transaction type depend on its relationships to other types which may be expressed by *dependencies*. A dependency is an implication that constrains the occurrence or order of events of two transactions.

Dependency	Definition	Meaning
Abort Dep. t_j AD t_i	$(\text{Abort}_{ti} \in H) \Rightarrow (\text{Abort}_{tj} \in H)$	if t_i aborts then t_j aborts
Commit -on-Termination Dep. t_j CTD t_i	$(e \in H) \Rightarrow (e \rightarrow \text{Commit}_{tj})$ where $e \in \{\text{Commit}_{ti}, \text{Abort}_{ti}\}$	if t_i terminates then t_j commits
Serial Dep. t_j SD t_i	$(\text{Begin}_{tj} \in H) \Rightarrow (e \rightarrow \text{Begin}_{tj})$ where $e \in \{\text{Commit}_{ti}, \text{Abort}_{ti}\}$	t_j cannot begin execution until t_i either commits or aborts
Begin Dep. t_j BD t_i	$(\text{Begin}_{tj} \in H) \Rightarrow (\text{Begin}_{ti} \rightarrow \text{Begin}_{tj})$	t_j cannot begin execution until t_i has begun
Begin-on-Commit Dep. t_j BCD t_i	$(\text{Begin}_{tj} \in H) \Rightarrow (\text{Commit}_{ti} \rightarrow \text{Begin}_{tj})$	t_j cannot begin execution until t_i commits
Begin-on-Abort Dep. t_j BAD t_i	$(\text{Begin}_{tj} \in H) \Rightarrow (\text{Abort}_{ti} \rightarrow \text{Begin}_{tj})$	t_j cannot begin execution until t_i aborts
Force-Commit-on-Abort Dep. t_j CMD t_i	$(\text{Abort}_{ti} \in H) \Rightarrow (\text{Commit}_{tj} \in H)$	if t_i aborts, t_j commits

Table1: Examples for ACTA-Dependencies

Dependencies can arise due to *structure* (e.g., dependencies between a parent and child transactions) or due to *behavior* (operations on objects). Some examples of structural dependencies which are used in WADL are summarized in table 1.

3 Semantics of WADL

In this subsection we describe parts of the semantics of WADL with the help of structural ACTA dependencies. First we identify the essential *events* and *states* of our model, presented in Figure 2.

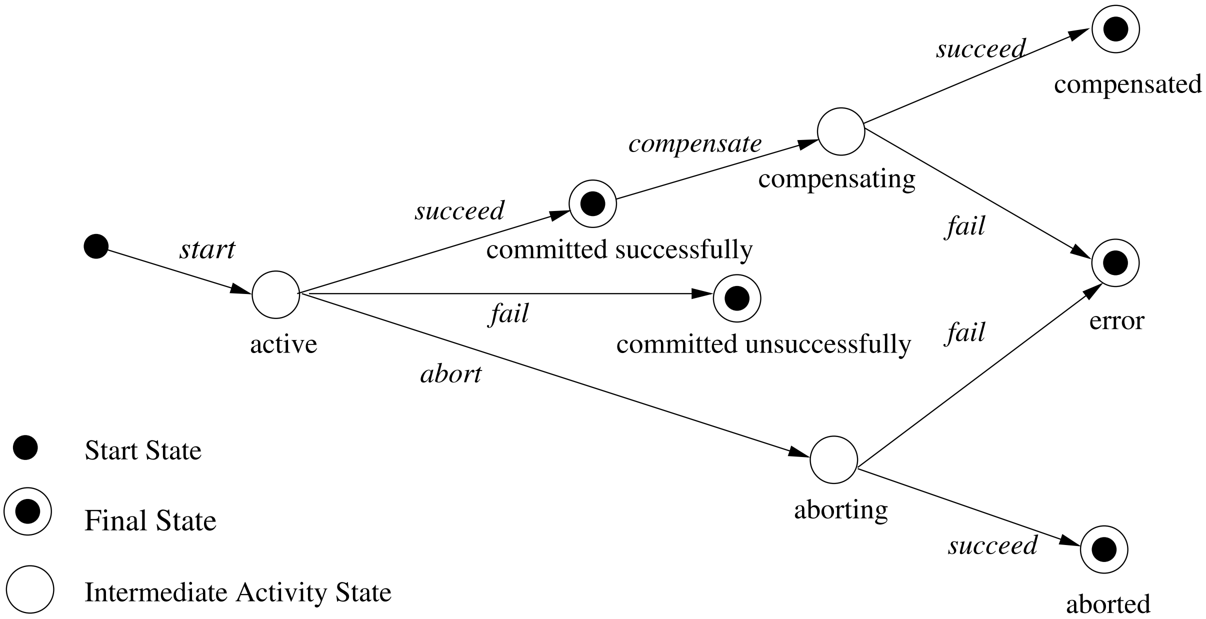

Figure 2: Event-state diagram for activities

The invocation of a transaction management primitive is termed a significant event in ACTA. The following significant events are valid in WAMO:

- Initiation events: start, compensate
- Termination events: succeed, fail, abort

An activity can be started if it is in the initial state *startable*. By triggering the event *start*, the activity changes its state from startable to *active*. An active activity can be terminated by the following events:

Succeed:	This event is equal to commit successfully or commit with a positive result. The corresponding termination state is *committed successfully*.
Fail:	This event is equal to commit unsuccessfully, commit with a negative result or abort semantically. The corresponding termination state is *committed unsuccessfully*.
Abort:	An active activity can be aborted by the user or by the system (e.g., if the client

for whom the trip reservation is arranged, falls ill then the whole reservation should be interrupted). An abort may be a complex process with two possible results: the process either succeeds (normal case) or fails (then a manual intervention is necessary). Additionally, it must be mentioned, that an abort is not always possible because it may endanger the safety of the overall process.

An important feature of WAMO is its compensation concept. Normally, a compensation is initiated if a vital activity fails. Only successful committed activities can be compensated. A compensation may succeed (normal case) or fail. If the compensation fails then a manual intervention will be necessary.

Manual intervention depends on the current situation and means to correct the failure, to restart the task or to change the execution state of the task in order to support process progress.

3.1 Structural dependencies in WADL

For the formalization of structural dependencies between activities (transactions) the following abbreviations are used:

PA: Parent Activity
CA: Child Activity
VCA: Vital Child Activity; the relation between the CA and its PA is vital
CompA: Compensation Activity; because of implementation aspects, each activity has a corresponding compensation activity which controls the compensation of A.
T: Task
n: number of last activity in a control structure

Additionally, there are two *invariant constraints* for the compensation process:

- If an activity is triggered by an compensate event then the corresponding compensation activity CompA is started.
- CompA can only be started if the corresponding (original) activity has committed successfully before.
- If CompA is started then it must commit successfully! Otherwise a manual intervention is necessary.

The structural dependencies in the rest of this section are presented not only in a short hand notation but also in a textual description.

3.1.1 Structural dependencies in a sequence

(a) CA_1 BD PA % Begin Dependency
(b) PA CTD CA_n % Commit-on-Termination Dep.
(c) PA AD VCA_j % Abort Dependency
(d) CA_i SD CA_{i-1} : $1 < i \leq n$ % Serial Dependency
(e) CA_i BCD VCA_{i-1}: $1 < i \leq n$ % Begin-on-Commit Dep.

If a vital child activity (VCA) fails then the compensation process is activated. All previous executed activities which have committed successfully are compensated in *inverse* order. At last the parent activity fails.

(f) $CompA_i$ BAD VCA_j : $j > i$ % Begin-on-Abort Dep.
(g) $CompA_{j-1}$ BCD $CompA_j$ % Begin-on-Commit Dep.

The compensation of a sequence (the successfull committed parent activtiy) consits of the following dependencies:

(h) $CompA_1$ BD CompPA % Begin Dependency
(i) see rule (g)
(j) CompPA SCD CompA % Strong-Commit Dependency

Description:
(a) The first child activity CA_1 of a sequence cannot start before its parent activity PA has been started.
(b) PA succeeds as soon as the last CA in the sequence terminates (succeeds or fails).
(c) PA fails if vital child activity *VCA* fails (then the whole sequence fails).
(d) If the sequence consists of several child activities, then the activity CA_i in the sequence is started as soon as the previous activity CA_{i-1} has terminated.
(e) If the previous child activity is a vital activity then the next child activity CA_i in the sequence is started only if VCA_{i-1} succeeds.

(f) A_i is the last successfull committed activity immediately before VCA_j. $CompA_i$ is started as soon as VCA_j fails.
(g) If there are several compensatable activities in the sequence then the compensation activity $CompA_{j-1}$ is started as soon as $CompA_j$ has committed successfully.

(h) The sequence is compensated in inverse order. The last successfull committed subactivity A_1 is compensated by starting $CompA_1$. $CompA_1$ cannot be started before CompPA has begun.
(i) see rule (g)
(j) The sequence is compensated successfully as soon as the last CompA succeeds.

3.1.2 Structural Dependencies in a ranked choice:

(a) CA_1 BD PA % Begin Dependency
(b) PA SCD CA_j : $1 \leq j \leq n$ % Strong-Commit Dependency
(c) PA AD CA_n % Abort Dependency
(d) CA_j BAD CA_{j-1} : $1 < j \leq n$ % Begin-on-Abort Dependency

The compensation of a ranked choice consits of the following dependencies:

(e) $CompA_j$ BD CompPA % Begin Dependency
(f) CompPA SCD $CompA_j$ % Strong-Commit Dependency

Description:

(a) The first child activity CA_1 of a ranked choice cannot start before its parent activity PA has been started.
(b) PA succeeds as soon as the first CA (ranked choice activity) succeeds.
(c) PA fails if the last child activity in the choice fails (then the whole choice fails).
(d) CA_j is started only if CA_{j-1} fails.
(e) If a ranked choice is compensated then there is exactly one activity A_j to compensate. $CompA_j$ cannot be started before CompPA has begun.
(f) The ranked choice is compensated successfully as soon as $CompA_j$ succeeds.

3.1.3 Structural dependencies in a free choice:

The dependencies between activities in a free choice are very similar to the dependencies in a ranked choice. The only difference is the dynamic execution order of activities in a free choice. The execution order is evaluated at run-time through a certain condition.

(a) CA_i BD PA % Begin Dependency
(b) PA SCD CA_j : $1 \leq j \leq n$ % Strong-Commit Dependency
(c) PA AD CA_n % Abort Dependency
(d) CA_j BAD CA_i : $i,j \leq n$ % Begin-on-Abort Dependency

Description:

(a) The first selected child activity CA_i of a free choice cannot start before its parent activity PA has been started.
(b) PA succeeds (commits successfully) as soon as one CA of the free choice succeeds.
(c) PA fails if *all* child activities in the choice fail (then the whole choice fails).
(d) CA_j is started if the previous executing CA_i fails.

The compensation dependencies in a free choice are the same as the compensation dependencies of a ranked choice.

3.1.3 Structural dependencies in a limited choice:

In a limited choice (conditional construct) exactly one child activity is started at run-time, depending on a certain condition.

(a) CA$_{\mathrm{i}}$ BD PA % Begin Dependency
(b) PA SCD CA$_{\mathrm{i}}$ % Strong-Commit Dependency
(c) PA AD CA_i % Abort Dependency

Description:
(a) Child activity CA_i which satisfies a certain condition cannot start before its parent activity PA has been started.
(b) PA succeeds as soon as CA_i succeeds.
(c) PA fails if CA_i fails.

The compensation dependencies in a free choice are the same as the compensation dependencies of a ranked choice.

3.1.4 Dependencies between AND-parallel activities

(a) CA$_{1..n}$ BD PA % Begin Dependency
(b) PA CTD CA$_{1..n}$ % Commit-on-Termination Dependency
(c) PA AD VCA % Abort Dependency

If a vital CA fails then all sibling activities which have committed successfully are compensated and all active brother activities are aborted. Activities which have not already started are not allowed to start.

(d) CompA$_{\mathrm{i..j}}$ BAD VCA % Begin-on-Abort Dependeny
(e) CA$_{\mathrm{i..j}}$ WD VCA % Weak-Abort Dependency

The compensation of an and-parallel control construct consits of the following dependencies:
(f) CompA$_{\mathrm{i..j}}$ BD CompPA % Begin Dependency
(g) CompPA SCD CompA$_{\mathrm{i..j}}$ % Strong-Commit Dependency

Description:
(a) The child activities cannot be started before their parent activity PA has been started.
(b) The parent activity PA succeeds as soon as *all* CAs have terminated (succeeded or failed)
(c) The parent activity PA fails if one of its *vital* child activities fails.
(d) A$_{\mathrm{i..j}}$ are successfull committed activities. The corresponding compensation activities of A$_{\mathrm{i..j}}$ (CompA$_{\mathrm{i..j}}$) are started as soon as a VCA fails.
(e) All active CAs are aborted as soon as a VCA fails.
(f) A$_{\mathrm{i..j}}$ are successfull committed child-activities. The corresponding compensation activities of A$_{\mathrm{i..j}}$ (CompA$_{\mathrm{i..j}}$) are started after CompPA has been started.
(g) If all CompA$_{\mathrm{i..j}}$ succeed then CompPA succeeds.

3.1.5 Dependencies between OR-parallel activities

(a) $CA_{1..n}$ *BD PA* % Begin Dependency
(b) *PA SCD* CA_i % Strong-Commit Dependency
(c) PA AD $CA_{1..n}$ % Abort Dependency
(d) CA_j ED CA_i % Exclusion Dependency
!(e)CA_j EBD CA_i % Exclusive Begin Dependency

In an OR-parallel construct no vital child activities are allowed. Therefore no compensation is triggered if a child activity fails.

The compensation of an or-parallel construct comprises the following dependencies:

(f) $CompA_i$ BD CompPA % Begin Dependeny
(g) CompPA SCD $CompA_{i..j}$ % Strong-Commit Dependency

Description:
(a) The child activities cannot be started before its parent activity PA has been started.
(b) The parent activity PA succeeds as soon as the *first* CA succeeds.
(c) The parent activity PA fails if all child activities fail.
(d) IF CA_i succeeds and CA_j has begun executing, then CA_j aborts (both CA_i and CA_j cannot succeed).
!(e) IF CA_i succeeds and CA_j has not begun executing, then CA_j cannot begin.

(f) A_i is the successfull committed child activity. The corresponding compensation activitiy $CompA_i$ is started as soon as CompPA has begun.
(g) If $CompA_i$ succeeds then also CompPA succeeds.

3.1.6 Dependencies of an iteration

(a) CA_1 BD PA % Begin Dependency
(b) CA_{i+1} SD CA_i % Serial Dependency
(c) PA CTD CA_i % Commit-on-Termination Dependency

Description:
(a) The first activation of the CA of an iterativPA cannot be started before the PA has been started.
(b) The i+1st activation of CA cannot be started before the i-th activation of CA has terminated.
(c) PA succeeds as soons as the last activation of CA terminates.

To simplify the semantics of an iteration there are three constraints:

- An iteration activtiy always succeeds: If an activity is repeated several times (until the condition is satisfied) then there may be several termination states (e.g. 2 times the activity may have succeeded and one time the activity may have failed). To compute the result of the iteration activity (PA) on the basis of the different termination states of the iterating CAs would be very difficult and probably ambiguous.

- The relation between the PA (iteration activity) and CA (activity which iterates) is always "non vital". This makes the handling of iterations easier and it also fits very well to the previous mentioned constraint (if a "vital" relation would be possible then the PA must fail if the iterating CA fails).

- The compensation of an iteration is a null operation.

3.1. Dependencies between activities and tasks

(a) T BD PA % Begin Dependency
(b) PA SCD T % Strong Commit Dependency
(c) PA AD T % Abort Dependency

Compensation of a task:

(d) CompT BD CompPA % Begin Dependency
(e) CompA SCD CompT % Strong Commit Dependency

Description:
(a) A task T cannot begin execution before the corresponding parent activity PA has begun.
(b) If task T succeeds then also its PA succeeds.
(c) If task T fails then also PA fails.
(d) The compensation of a task cannot begin before the corresponding parent compensation activity has begun.
(e) The compensating parent activity succeeds as soon as the compensation task succeeds.

Literature:

[CR92] Chrysanthis P. K., Ramamritham K.: ACTA: The Saga Continues. In: [Elm92].
[EL95] Eder J., Liebhart W.: The Workflow Activity Model WAMO. Proc. of the 3rd Int. Conference on Cooperative Information Systems, Vienna 1995.
[Elm92] Elmagarmid A. K.: Database Transaction Models for Advanced Applications. Morgan Kaufmann, 1992.
[GHS95] Georgakopoulos D., Hornick M., Shet A.: An Overview of Workflow Management: From Process Modeling to Workflow Automation. In: Distributed and Parallel Databases, Vol.3, No.2 1995.
[Jab94] Jablonski S.: Functional and Behavioral Aspects of Process Modelling in Workflow Systems. In: G. Chroust and A. Benczur (ed.): CON 94 Workflow Management: Challenges, Paradigms and Products, Oldenburg 1994.