UML Activity Diagrams as a Workflow Specification Language

Marlon Dumas and Arthur H.M. ter Hofstede

Cooperative Information Systems Research Centre
Queensland University of Technology
GPO Box 2434, Brisbane QLD 4001, Australia
{m.dumas, a.terhofstede}@qut.edu.au

Abstract. If UML activity diagrams are to succeed as a standard in the area of organisational process modeling, they need to compare well to alternative languages such as those provided by commercial Workflow Management Systems. This paper examines the expressiveness and the adequacy of activity diagrams for workflow specification, by systematically evaluating their ability to capture a collection of workflow patterns. This analysis provides insights into the relative strengths and weaknesses of activity diagrams. In particular, it is shown that, given an appropriate clarification of their semantics, activity diagrams are able to capture situations arising in practice, which cannot be captured by most commercial Workflow Management Systems. On the other hand, the study shows that activity diagrams fail to capture some useful situations, thereby suggesting directions for improvement.

1 Introduction

UML activity diagrams are intended to model both computational and organisational processes (i.e. workflows) [14,15]. However, if activity diagrams are to succeed as a standard in the area of organisational process modeling, they should compare favorably to the languages currently used for this purpose, that is, those supported by existing Workflow Management Systems (WFMS).

In this paper, we investigate the expressiveness and adequacy of the activity diagrams notation for workflow specification, by systematically confronting it with a set of *control-flow workflow patterns*, i.e. abstracted forms of recurring situations related to the ordering of activities in a workflow, and the flow of execution between them. Many of these patterns are documented in [3,4], and a comparison of several WFMS based on these patterns is provided in [4].

Our evaluation demonstrates that activity diagrams support the majority of the patterns considered, including some which are typically not supported by commercial WFMS. This is essentially due to the fact that activity diagrams integrate signal sending and processing at the conceptual level, whereas most commercial WFMS only support them as a low-level implementation mechanism.

While activity diagrams compare well to existing WFMS in this respect, they exhibit the major drawback that their syntax and semantics are not fully defined in the standard's documentation[1]. Indeed, while the features inherited from Harel's statecharts [9] have a formal operational semantics, the features specific to activity diagrams are only partially formalised in the standard (through OCL statements), and their description in natural language leaves room for some ambiguities that we point out throughout the paper. We hope that some of these ambiguities will be clarified in future releases of the standard.

The rest of the paper is structured as follows. Section 2 discusses some semantical issues of the activity diagrams notation, focusing on control-flow aspects. Sections 3, 4, and 5 evaluate the capabilities of activity diagrams against different families of workflow patterns. The patterns considered in sections 3 and 4 are extracted from [4], while those discussed in section 5 are variants of the well-known producer-consumer pattern. Finally, section 6 points to related work, and section 7 discusses directions for improving the activity diagrams notation.

2 Overview of Activity Diagrams

The aims of the following overview are (i) to discuss the semantics and properties of critical constructs used in the rest of the paper, (ii) to identify some ambiguities in the standard, and (iii) to explain how this paper deals with these ambiguities. It is not intended as an introductory overview. Readers not familiar with activity diagrams may refer to e.g. [8].

2.1 States and Transitions

UML activity diagrams are special cases of UML *state diagrams*, which in turn are graphical representations of *state machines*. The *state machine* formalism as defined in the UML, is a variant of Harel's statecharts [9].

State machines are transition systems whose arcs are labeled by ECA (Event-Condition-Action) rules. The occurrence of an event fires a transition if (i) the machine is in the source state of the transition, (ii) the type of the event occurrence matches the event description of the transition, and (iii) the condition of the transition holds. The event (also called trigger), condition (also called guard), and action parts of a transition are all optional. A transition without an event is said to be *triggerless*. Triggerless transitions are enabled when the action or activity attached to their source state is completed.

A state can contain an entire state machine within it, leading to the concept of *compound state*. Compound states come in two flavours: OR and AND. An OR-state contains a single statechart, while an AND-state contains several statecharts (separated by dashed lines) which are intended to be executed concurrently. Each of these statecharts is called a *concurrent region*. When a compound state is entered, its initial transition(s) are taken. The execution of a compound state is considered to be complete when it reaches (all) its final state(s). Initial states are denoted by filled circles, while final states are denoted by two concentric circles: one filled and one unfilled (see figure 1).

[1] When discussing the syntax and semantics of activity diagrams, we take as sole reference the final draft delivered by the UML Revision Task Force 1.4 [14].