Implementation of Workflows as Finite State Machines in a National Doctoral Dissertations Archive

Nikos Houssos, Dimitris Zavaliadis, Kostas Stamatis and Panagiotis Stathopoulos

[1] *National Documentation Centre / National Hellenic Research Foundation, 11635, Athens, Greece.*
nhoussos (at) ekt.gr, dimzava (at) ekt.gr, kstamatis (at) ekt.gr, pstath (at) ekt.gr

Abstract: *Workflows for submission and processing of digital material is one of the important aspects of open access repositories and the major concerns in their implementation. The subject of the present paper is the modelling of processing workflows in the Hellenic National Archive of Doctoral Dissertations as Finite State Machines and their implementation through an original software library that enables client applications to create and utilise FSMs.*

Keywords: *Repositories, Workflows, Finite state machines, Electronic theses and dissertations, Object-oriented design.*

I. INTRODUCTION

In this article we present the implementation of workflows for cataloguing and processing doctoral theses in the Greek National Archive of Doctoral Dissertations (HEDI) based on an original software library for Finite State Machines that has been developed by the Hellenic National Documentation Centre (EKT).

EKT has been granted by law 1566/1985 the responsibility of developing and maintaining the Greek National Archive of doctoral theses. The archive contains the doctoral dissertations produced in Higher Education Institutions as well as a number of PhD theses awarded to Greek scholars by universities outside Greece (USA, UK, Canada, and Germany, among others), in total about 24.500 theses as of February 2010, 2.75M pages of digitised and born-digital dissertations, with 1200 -1400 new dissertations arriving every year.

HEDI is supported by IT systems since 1986 when EKT developed the bibliographic database 'National Archive of PhD Theses" employing for cataloguing the home-grown library automation software, ABEKT **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**. Initially, EKT has been collecting from individual universities and cataloguing theses solely in print form. The database was made available to the public via the mainframe host computer 'Hermes' for more than a decade from 1986 until 1999. Thereafter, a new version of ABEKT has been used, including support for metadata standards like UNIMARC, UNIMARC Authorities and ISO 2709 and the Z39.50 protocol for search and retrieval of bibliographic records. This system, later integrated into the ARGO digital library portal (argo.ekt.gr) (Sfakakis, 2003) that is still in operation, provides free web-based access to metadata as well as advanced services to librarians, through a library catalogue-like user interface. Meanwhile, EKT proceeded with executing a major digitisation project for the majority of the dissertations in the – until then – print-only archive, which enabled open access to theses full text for Web users – realised through a specialised presentation application (Loverdos, 2001). Furthermore, in later years universities have been submitting theses to the archive, in both print and electronic form. As a result, today more than 75% of the theses in HEDI are available online in full text.

In 2009, a decision was made to re-build and modernise the information infrastructure supporting HEDI – a project that was completed in 2010. The following main choices were made:

- Create an e-theses repository on the DSpace platform, mainly targeting end-users.
- Create a separate IT application for the administration and management of the EKT internal workflows that are necessary for the processing of the material submitted to HEDI. The output of these workflows for every thesis is a quality-controlled metadata record and a searchable full-text file (or set of files). An important aspect to consider is that these workflows should be able to handle dissertations in both print and electronic form – note that the print archive is still maintained although a thesis is normally submitted in both print and electronic form. This application supports processing workflows for theses, acting as a service consumer for the workflow engine presented in this article.
- Maintain and constantly update the ARGO version of HEDI as a bibliographic system of choice for librarians offering to the latter additional important services like record export and transformation among various formats (e.g., MARC21 to UNIMARC). Ceasing to provide the HEDI database through this portal was considered to be a non sound option, given the popularity of ARGO among the Greek library community as well as the support of UNIMARC and related services.
- Select and reuse open source main software components, from the operating system to the repository and the middleware/database layer, while exploiting EKT's virtual infrastructure (Stathopoulos, 2009) in order to provide HEDI services with higher availability, scalability and total cost of ownership. The same software

components and technology have been previously used by EKT while developing institutional and subject repositories.

II. PROCESSING WORKFLOW FOR THESES

The internal workflow followed for the processing of the incoming dissertations material is depicted, at a high level of abstraction, in **Figure 2** and can be described as follows:

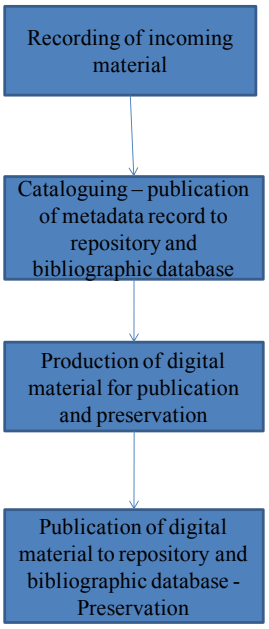

Figure 2.High-level overview of workflow phases.

1. The first step is the recording of the incoming material upon its receipt by EKT. This material includes the full-text dissertation document in print and electronic form (only print is currently mandatory); sometimes supplementary files (e.g., data sets) are provided. Furthermore, a declaration sheet by the doctoral degree holder about copyright issues is normally received – authors are being given the option to either make their full-text thesis publicly available immediately or after an embargo period (at most three years), as is common in other e-thesis systems. This sheet contains also basic descriptive metadata, comprising also author-defined keywords and abstract. Recording of physical (e.g., thesis hard copies) and digital object (files) is performed, as well as creating a very short metadata record including fields like author, degree awarding institution, date of degree award, date of submission to HEDI, public availability/copyright status as specified by the author, etc.
2. The second step in the workflow is the cataloguing of the thesis, to produce a detailed metadata record in the UNIMARC format; this is performed by qualified EKT/NHRF library personnel, starting either from scratch or from an ingested institutional repository record through a user-friendly interface. Furthermore, author-defined metadata is also taken into account, when available, and could be utilised after quality control. It is during the cataloguing phase that a unique identifier (number) is assigned to this particular thesis; this number is permanent and is used to identify all material related to the thesis within the archive. The assignment of the unique identifier is also a strict pre-requisite for starting the processing of the corresponding digital material (see step 3). After cataloguing, the metadata record (without the corresponding digital files – e.g. thesis full text) is published online in the repository and the bibliographic database.

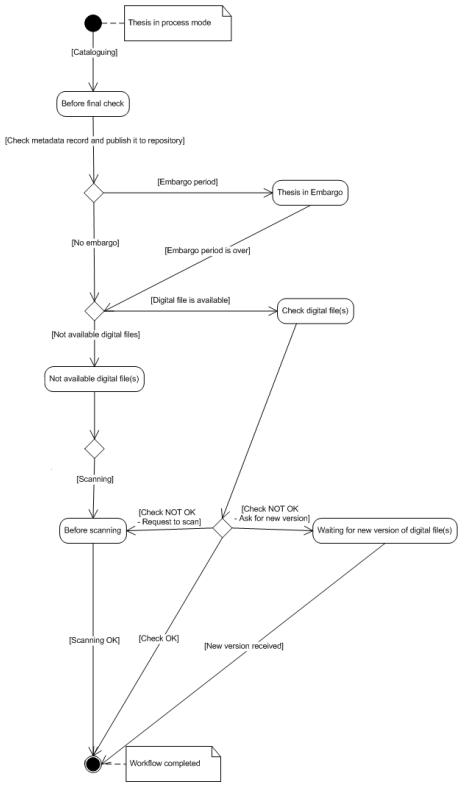

Figure 2.The workflow modelled as a Finite State Machine.

3. The third phase is the processing of the material for the generation of a high quality, fully searchable digital file containing the dissertation full-text, as well as image files for individual pages to allow online page-by-page reading. This phase typically includes one of the following alternatives:
   - Digitisation, if the thesis document is available only in print form, which includes Optical Character Recognition (OCR) processing for making the resulting PDF file full-text searchable.
   - Production, based on the initially submitted digital file(s), of a thesis full-text file (in PDF format) ready for publication and subsequent preservation. This could entail, among others, transformation to PDF from other formats like .doc and .odp and checks on whether the dissertation text is fully searchable (otherwise, OCR is applied).

4. The last step in the workflow concerns the submission and storage of the digital material in a repository and the corresponding detailed record at the bibliographic database. Subsequent updates in the metadata and/or digital files are performed through the theses administration application and are propagated to both the repository and the bibliographic database. Future preservation actions are enacted on the digital files in the repository.

Figure 2 depicts the processing workflow modelled as a Finite State Machine (FSM) - a simplified version of the workflow is depicted for economy of presentation.

III. A SOFTWARE ENGINE FOR FINITE STATE MACHINES

To address the implementation of the aforementioned workflow in the system supporting HEDI, we have developed a software library, call FSM engine, for executing workflows modelled as Finite State Machines.

In particular, the FSM Engine is a Java API that allows definition and execution of workflows represented as state machines. It helps in cases where the behaviour of an object needs to be changed at runtime depending on its state, eliminating the need for extensive use of if/else and switch statements which make code unreadable and difficult to maintain. The FSM Engine is inspired by the State Design Pattern (Gamma, 1995) and has been developed by EKT.

The FSM Engine provides a set of reusable Java classes and interfaces that apply the abstract concepts of finite state machines at the source code level. By extending and implementing those classes and interfaces, a client application can systematically define how a system reacts to certain events avoiding at the same time cluttered case statements. This is achieved by specifying a set of valid states for an object along with possible transitions between states which are triggered by events. In addition, guard conditions can determine whether a transition should be executed or not and a set of possible actions can be associated with a given transition in order to be fired as a side effect of executing the transition. Branching is also supported, which enables multiple discrete transitions for a given event and at a given state – the evaluation of a boolean expression determines the specific transition to be followed in such a case.

The FSM engine domain model is depicted in Figure 3. The main classes comprising the engine are the following:

StateContext

Maintains a reference to the current state. It should be implemented by one or more client classes that can have different internal states and whose behavior changes depending on those states.

State

Represents the different states of the state machine. Each possible state of a class implementing the StateContext interface should correspond to a concrete implementation of this interface.

StateName

A Java enum acting as a bridge between StateContext and States..

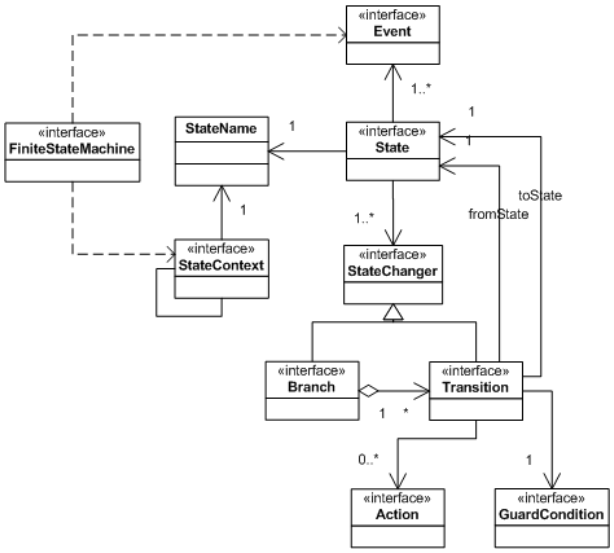

Figure 3.The FSM engine domain model.

Event

Represents an external or internal event such as a button click or a date expiration that can change the State of an object, thus leading to a state Transition in the system. Typically, there is only one Transition corresponding to a given Event but it is also possible to accommodate more than one Transitions for an Event via Branches and GuardConditions.

Transition

Defines the transition from one state to another as a response of the state machine to an occurrence of a specific Event. A Transition can be guarded by a GuardCondition which determines whether the execution of the Transition should proceed or not. In addition, a Transition can be associated with one or more Actions to be performed once the Transition has been executed.

GuardCondition

A GuardCondition is actually a Boolean expression that affects the behavior of the state machine by enabling Transitions only when it evaluates to true and disabling them when it evaluates to false. A GuardCondition can be associated with many Transitions but each Transition can have only one GuardCondition.

Action

Defines an activity that is to be performed when executing a certain Transition. An Action can be associated with more than one Transitions and a Transition can have more than one Actions. Also note that an Action can well trigger an Event having as a result another Transition.

Branch
Acts as a container of Transitions. It is used in cases where there is no one-to-one mapping between an Event and a Transition but instead there are more than one possible Transition for a given Event. In principle, the Transitions contained in a Branch are mutually exclusive, meaning that only one Transition will be executed each time and this will be determined once the GuardConditions of all Transitions have been evaluated.

FiniteStateMachine
An abstraction encapsulating the internal details of the state machine, presenting a single interface to the outside world. Implementation classes should be used as the main point where the lifecycle management of a StateContext takes place, whenever a triggering Event occurs.

In practice, to use the engine, the implementor of a workflow has to create concrete classes implementing interfaces like State, StateContext, Event, Transition, Action or the corresponding default classes offered by the library.

An important feature of the engine is that a specific workflow based on the implemented classes is specified outside the code in an XML configuration file – in particular an application context file of the Spring framework. The Spring dependency injection mechanisms are employed for an instantiation of a workflow/FSM engine. Therefore, modifications in a workflow (additions of states, transitions, events, actions, etc.) can be injected into the system without modifying source code – just by editing the configuration file.

III. SUMMARY AND FUTURE WORK

The main subject of the present article is the modelling of processing workflows in the Hellenic National Archive of Doctoral Dissertations as Finite State Machines and their implementation through an original software library that enables client applications to create and utilise FSMs. The adopted modelling and development approach proved to be appropriate for this use case.

Plans for further work regarding HEDI include the support of online submission workflows by authorised parties outside EKT (e.g. graduating doctoral students, university personnel) and automated ingest of records from external systems. These services require, among others, more sophisticated workflows for quality control both at the metadata and digital file level and further automation of digital file checking and processing.

Regarding the FSM engine, future plans include publising and maintaining it as an autonomous open source project under a license that facilitates re-use in third-party applications. In terms of features, a priority is to make the engine even less intrusive for client applications, probably through mechanisms like Java annotations, aspect-oriented programming, and potentially metaprogramming in cases of integration with other than Java languages running on the Java Virtual Machine such as Groovy. Furthermore, support of advanced workflow features that go beyond standard FSM functionality will be investigated, such as clustering and orthogonality as supported by schemes like statecharts (Harel, 1987).

REFERENCES

Gamma, E. et al., *Design patterns: elements of reusable object oriented software*, Addison Wesley Longman, Inc. (1995).

Harel D., "Statecharts: a visual formalism for complex systems", *Science of Computer Programming*, **8**(3), 231-274 (1987).

Loverdos, C., Kapidakis, S., "Flexible, service-based content presentation: The Hellenic Dissertations Presentation System", *Lecture Notes in Computer Science*, Darmstadt, Germany, **2163** (2001).

Sfakakis, M., Kapidakis, S., "An architecture for online information integration on concurrent resource access on a Z39.50 environment", *Lecture Notes in Computer Science*, Springer Verlag, Berlin, Heidelberg, **2769**, 288-299 (2003).

Stathopoulos, P., Soumplis, A., Houssos, N., "The case study of an F/OSS virtualization platform deployment and quantitative results", *5th International Conference on Open Source Systems*, Skövde, Sweden (2009).