Exploring the Alloy Operational Semantics for Case Management Process Modeling

Irina Rychkova
Centre de Recherche en Informatique
Université Paris 1 Panthéon - Sorbonne,
90 rue Tolbiac, 75013 Paris, France
Email: irina.rychkova@univ-paris1.fr

***Abstract*—Efficient case management in industry is hampered by attempts to deal with case management process the same way as with regular business process. Development of the specific approaches for case management process modeling and analysis is therefore an important endeavor that can improve the case management practice.**

In this work, we provide a mathematical model and a comprehensible formalism for reasoning about the meaning - the semantics - of case management process. We represent case management process as a finite state machine (FSM) and express its operational semantics in the Alloy specification language. The Alloy Analyzer tool allows us to define, simulate and validate a case management process model while efficiently managing its complexity. We illustrate our findings on the example of Mortgage Approval process.

***Index Terms*—case management, business process modeling, finite state machine, Alloy**

I. Introduction

Consider the following processes: crime investigation, mortgage processing, patience care, design of a new house. Being completely different by their objective, they have one important characteristic in common: unpredictability. Each of these processes unfold according to a particular *case* and emerging knowledge about this case rather than according to a predefined scenario.

Particular importance of case management processes (CMP) has been recognized since early 90x [1] [2]. Until now, however, case management processes remain largely "pen and paper".

Efficient case management in industry is hampered by attempts to deal with case management process the same way as with regular business process. Development of the specific approaches for CMP modeling and analysis is therefore an important endeavor that can improve the case management practice.

Lamport defines process as a sequence of events occurring in system [3], where each event is triggered by an (internal or external) action. According to this definition, a business process can be seen as a sequence of events triggered by the activities of (internal or external) business process actors.

The existing methods for business process modeling and analysis (e.g. BPMN - BPEL bundle) follow the *imperative* principles, implying that the order of events occurring during the process execution is *predefined*. For case management processes, however, it is not true [4]. In this work, we shift this traditional modeling paradigm and exploit the *declarative* principles for CMP modeling and analysis. We model CMP as a state-transition system that allows us to handle process events whose order of occurrence is *undetermined*.

The objective of this paper is to provide a mathematical model and a comprehensible formalism for reasoning about the meaning - the semantics - of case management process. In particular, we focus on (a) how a finite state machine (FSM) [5] [6] abstract model can be used to provide the operational semantics for CMP, (b) how CMP model can be formally specified in Alloy [7] and analyzed with Alloy Analyzer [8] and (c) how the results of this formal analysis can be interpreted and used by *business* specialists for process improvement.

We illustrate the proposed operational semantics on the example of Mortgage Approval process. First, we define a mortgage approval as a FSM: we specify the *states* of a mortgage case and the *transitions* between these states. The state transitions are *triggered* by the corresponding mortgage processing activities (e.g. application completion, validation of applicant's information, property appraisal, etc.). Their order in the model, however, remains undetermined. This FSM, describes how the mortgage case (application file) evolves from its submission to approval or rejection.

Once the state and state transitions are defined, we formalize the semantics of this FSM in Alloy and use the Alloy Analyzer tool [8] for model simulation and validation. Model simulation provides us with an instant visual feedback in a form of diagrams capturing the FSM execution scenarios. We use Alloy model checking for more specific reasoning about model properties: for example, we validate if our CMP model meets its business requirements.

Until now, Alloy specification language was successfully applied to design and verification of software and hardware systems. In this work, we position Alloy and Alloy Analyzer as a toolbox for both technical and business specialists. Along with UML, BPMN and other conventional diagrams, Alloy diagrams represent design artifacts and can be studied, referred to, and serve a mean for communicating and evaluating both business and technical design decisions.

Once the model validity at a given level of detail is checked - the model can be *refined*. This iterative approach allows us

to efficiently manage the model complexity.

The operational semantics defined in this work is a first step to using formal methods; it paves the road to automated design, validation and verification of CMP.

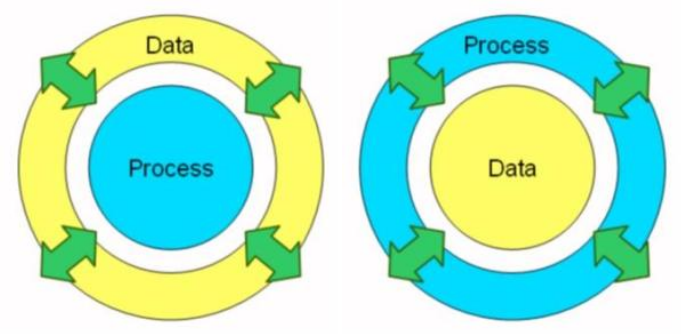
The reminder of this paper is organized as follows: In Section II, we provide a definition for case management process, discuss the challenges related to case management and introduce our example - the Mortgage Approval Process; In Section III, we introduce a finite state machine (FSM) abstraction for CMP modeling: first, we provide a mathematical model for FSM as defined by Plotkin [5], then we apply this model to our example. In Section IV, we formalize the FSM-based specification of the Mortgage Approval process in Alloy. In Section V, we illustrate how the CMP model can be simulated, analyzed and, eventually, improved using the Alloy Analyzer tool. We show the scenarios of mortgage approval generated by Alloy Analyzer and demonstrate how business properties of the Mortgage Approval process can be formally validated. In Section VI, we discuss the related works; Section VII presents our conclusions.

II. Case Management Process and Adaptive Case Management

Davenport [1], [2] defines case management process as a process that is not predefined or repeatable, but instead, depends on its evolving circumstances and decisions regarding a particular situation, *a case*. He discusses the need in specific approaches to handle such processes. The Case Management Process Modeling (CMPM) Request For Proposal released by OMG on September 2009 [9] expresses the practitioners' demand in the case management solutions. OMG defines case management as *"A coordinative and goal-oriented discipline, to handle cases from opening to closure, interactively between persons involved with the subject of the case and a case manager or case team"*. Case management processes (CMP) have multiple applications, including *"...licensing and permitting in government, insurance application and claim processing in insurance, patient care and medical diagnosis in health care, mortgage processing in banking..."* [9]. The main resource of a CMP is knowledge obtained as a result of communication between multiple actors/users. This knowledge is used for making decisions during the case handling.

A. Adaptive Case Management (ACM) vs. Business Process Management (BPM)

Business Process Management (BPM) and Adaptive Case Management (ACM) demonstrate conceptually different views on the system design. Process-centered view adapted by Business Process Management (BPM) implies that the data emerges and evolves within a process according to a predefined control flow (Fig. 1a) - similarly to a product evolving on a conveyor belt. This view suits to predictable and highly repeatable processes. One of the major challenges identified by both practitioners and researchers in the ACM field, is the attempts to deal with case management process in the industry the same way as with regular business process - i.e. applying the process-centered view. In this work, we implement the data-centered view (Fig. 1b) that is proposed by the Adaptive Case Management (ACM) practitioners [10]. This view implies that the process shall be adapted at run time, according to evolution of case circumstances and case-related data. This view suits to nondeterministic, knowledge-intensive processes like case management processes [11].



(a) Process-centered view (b) Data-centered view

Fig. 1: BPM vs. ACM systems, from [10]

The body of knowledge on knowledge-intensive processes and ACM has been extensively developed by practitioners and published in the Internet blogs (see http://www.adaptive-case-management.com/, http://www.column2.com, and many others). The group of researchers at IBM proposes an approach that incorporates process- and data-centered perspectives and is based on concept of business artifacts [12], [13], [14]. Recently it has been complemented by academical research and gave a rise to a series of publications ([15], [10], [16], etc.)

In the following sections, we define an operational semantics for case management process and illustrate this semantics on the example of the Mortgage Approval process.

B. Example: Mortgage Approval process

A mortgage is a loan for buying a house. Mortgage Approval process is a typical example of a case management process.

The objective of mortgage approval process is to come up with a right decision about the applicants request: provide him/her with a loan or reject the application. In order to make this decision, the potential money lender (e.g. a bank) must ensure (i) that the applicant is credible and will be able to pay the loan back and (ii) that the amount of loan is justified with respect to the real cost of a property. [1]

In [17], we provide a detailed description of mortgage approval process as defined by different financial institutions in the USA. In this work, however, we model a simplified version of this process. We summarize the main steps and business rules defined for Mortgage Approval process in the list below.

[1]In the USA, a bank will remain the owner of the property until the mortgage is fully paid back. Thus the bank shall validate that the property in question costs at least as much as the seller requests.

Mortgage Approval.:

- An applicant submits a mortgage application where he/she requests for a loan.
- A bank can process only one application from a given applicant at a time.
- The application should provide a list of documents including the applicant's name, address, employment information, Internal Revenue Service (IRS) forms, recent pay-stubs, medical certificate, information about the property to buy[2] etc. (The exact list may vary depending on the financial institution and the particular situation of an applicant.)
- The application revision starts with a validation of the applicant's credibility (i.e. the employment information must be complete); the information about a property to buy can be provided later in the process.
- When the information on the property to buy is submitted, the lender sends an appraisal agent to evaluate (appraise) this property. Negative appraisal result is a sufficient condition for the mortgage application rejection.
- Preapproval - is an intermediate status of the application that is given upon the confirmation of applicant's credibility.
- The proof of the applicant's credibility and positive appraisal results are necessary conditions for application approval.
- Some elements of the application (e.g. a medical certificate, family situation, etc.) may trigger the specific loan approval conditions (e.g. demand supplementary insurance) or affect the loan type or duration.

As follows from the description, the process above should be able to handle (in general, unpredictable) sequences of events (i.e. submission of application elements, receiving evaluation results, etc) and cannot follow a predefined scenario.

III. Case Management Process: Abstract Model and Operational Semantics

Theoretical computer science uses a concept of *operational semantics* in order to define the meaning of a computer program. The operational semantics explains the meaning of a program "in terms of a hypothetical computer which performs the set of actions which constitute the elaboration of that program." [18]

In this work, we define the meaning of the Mortgage Approval process by modeling it as a nondeterministic finite state machine and by formalizing its operational semantics in Alloy specification language.

A. Finite State Machine

A finite state machine (FSM) or finite automaton - is a model of computation that specifies a machine that can be at one state at a time and can change its state (perform a state transition) as a result of a triggering event or condition. A FSM is defined by a (finite) set of its states and a set of triggering conditions for each of its transitions.

Mathematical model:

A FSM can be defined as a quintuple $(M = \langle Q, \Sigma, \delta, q_0, F \rangle)$ where:

Q is a finite set of states;

Σ is a finite set of triggering events or conditions (input alphabet);

$\delta : Q \times \Sigma \to P(Q)$ is the state transition relation;

$q_0 \in Q$ is the initial state;

$F \subset Q$ is the set of final states.

One can distinguish deterministic and nondeterministic FSM: for each state and triggering event, a deterministic FSM specifies only one next state, whereas a nondeterministic FSM can specify multiple possible next states. Using the mathematical model above, for deterministic FSM, we can write the state transition relation as $\delta : Q \times \Sigma \to Q$ - a function that returns a single state. For nondeterministic FSM it returns a set of states.

It is not difficult to prove that nondeterministic finite automata (NFA) are equivalent (i.e. can be translated) into deterministic finite automata (DFA). This translation results to more complex model (with increased number of states and transitions) therefore in many cases (including our example) the use of nondeterministic model is justified.

Application to the Case Management Process:

Case management process can be modeled as a finite state machine where

- a FSM *state* represents "a status" of the case (e.g. case-related documents, evidences, examination results, reports, decisions etc.) at a given moment of time;
- FSM *triggering events* represent events occurring during the case (e.g. document submitted, results arrived, decision made by a manager, request canceled, etc) and resulting from activities performed by human agents or systems involved into the case;
- FSM *transitions* can be associated with one (or multiple) activities and their specific outcomes (triggering events);
- The possible case management outcomes (accepted or rejected claim, recovered patient etc) can be associated with as a FSM set of final states.

In the next section, we model the Mortgage Approval process as a finite state machine and express its operational semantics in the Alloy specification language.

B. FSM for the Mortgage Approval process

The Mortgage Approval process manages an applicant's case until a decision on the mortgage application is made. Here the current state of the case is represented by a current state of a submitted mortgage application. We model the evolution of the mortgage application state over time as a FSM illustrated in Fig. 2. We define the following abstract states for a mortgage application:

$$Q_{App} = \{Incomplete, Complete, Preapproved, Approved, Rejected\}$$

The state transitions represent the activities carried out by an applicant (ex.: complete application) or by a bank (ex.: revise

[2]By the property information we understand all the data that would serve for further property appraisal.

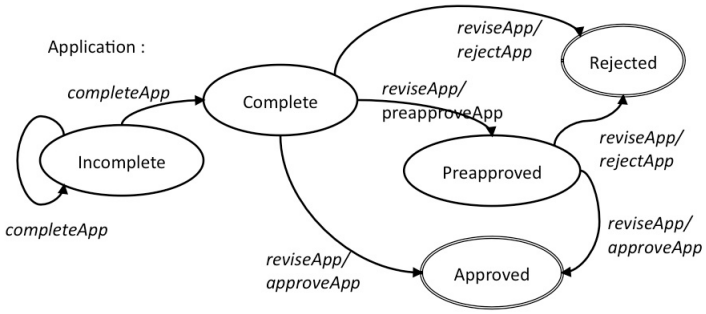


Fig. 2: A FSM for mortgage application handling: FSM_{app}

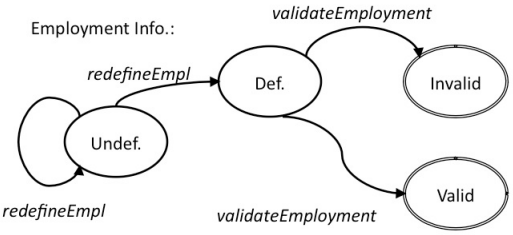


Fig. 3: A FSM for Employment Info handling: FSM_{empl}

application). These activities are shown as transition labels in Fig. 2.

The resulting FSM is nondeterministic: the same transition can define more then one resulting state. For example, the complete application (*completeApp*) activity in Fig. 2 can either trigger the application transition from *Incomplete* to *Complete* state or can result in no transition, keeping the application *Incomplete*. This reflects the real nature of the bureaucratic procedure behind.

Based on the process description, the current state of a mortgage application depends on the states of elements constituting this application: the applicant's employment information and the property information. We define the abstract states of these elements as follows:

$$Q_{EmpInfo} = \{Undefined, Defined, Invalid, Valid\}$$

$$Q_{PropInfo} = \{NotFound, Found, AppraisedOK, AppraisedKO\}$$

Our objective is to design a process for mortgage approval that will *adapt* to the evolving circumstances regarding the concrete applicant's case while *respecting* the business rules of the organization.

To ensure adaptability, we define a model where the mortgage application elements (i.e. employment info and property info) can emerge and evolve independently and with no predefined order (e.g. due to modification, validation, appraisal). For our example, we model the evolution of Employment info. and Property info. with separate FSMs (Fig. 3, 4). The transitions of these machines are labeled with the names of the corresponding activities. These FSMs are also nondeterministic.

The state of the mortgage approval process itself can be now considered as a combination of states of the three FSM defined above (Fig. 5.)

To guarantee that our model will respect the business rules defined by the organization (a bank), we express these rules

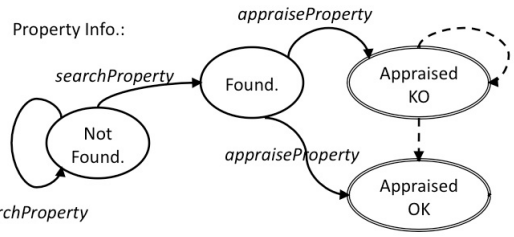


Fig. 4: A FSM for Property Info handling: FSM_{prop}

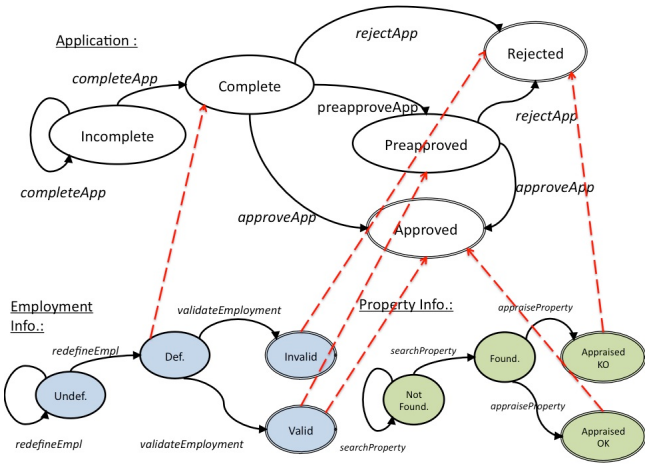


Fig. 5: A FSM for a mortgage approval process as a combination of $FSM_{app}, FSM_{empl}, FSM_{prop}$

as dependencies between the state machines (red dashed lines in Fig. 5).

To specify how these machines will work together, we introduce the Alloy operational semantics for our Mortgage Approval process.

IV. Alloy Operational Semantics for Mortgage Approval Process

Alloy [7] is a declarative specification language developed by the Software Design Group at MIT. Alloy is a language for expressing complex structural constraints and behavior based on first-order logic.

Alloy Analyzer [8] is a tool for the automated analysis of models written in the Alloy specification language. Given a logical formula and a data structure that defines the value domain for this formula, the Alloy Analyzer decides whether this formula is satisfiable. Mechanically, the Alloy Analyzer attempts to find a model instance - a binding of the variables to values - that makes the formula true. [19]

Specification of CMP in Alloy can be divided into four steps as follows:

- **Definition of data structure.** In this step, the case elements, their structure and relations between them are formalized with Alloy signatures (sig) and constraints.
- **Definition of FSM states.** In this step, the abstract states for each case element are identified. These states will later constitute the component FSMs.

- **Definition of triggering conditions.** Here, the conditions that will trigger a state transition in FSM are identified and formalized with Alloy predicates.
- **FSM assembling.** In this step, the component FSMs, specified for each case element are assembled into a composite FSM of the CMP. We formalize the relations between these FSMs with Alloy predicates.

FSM-based operational semantics provides the meaning for a CMP model and allows for further formal analysis to be carried out on this model. In the reminder of this section, we formalize our example - mortgage approval process - using Alloy specification language. In the further section we demonstrate how the formal specification can be analyzed using formal model checking techniques.

A. Definition of data structure.

Data structures in Alloy are represented with signatures labeled by the keyword `sig`. Alloy signatures can be abstract or concrete, can have explicit cardinalities (i.e. `one` or `set`), and can contain one or multiple fields. A field can be seen as analogy of a class attribute in OO languages.

Alloy specification language is based on the set theory. From the set-theoretical perspective, each Alloy signature defines a *set* of element of a given kind whereas its fields define the relations with another sets (signatures).

We define the Employment Info and Property Info data objects as Alloy signatures extending the generic `Obj` data type:

```
sig EmploymentInfo extends Obj{}
sig PropertyInfo extends Obj{}
```

At this level of detail, we do not precise the structure of these elements (this can be elaborated at refinement if needed).

We define the Application signature by specifying the relations between the mortgage application, employment information and property information:

```
sig Application extends Obj{
/*information about a house to buy */
property : one PropertyInfo,
/*information about the applicant employment */
employment : one EmploymentInfo
 }
```

B. Definition of FSM states.

Based on the set-theoretical perspective, we model a *state* of a data object as a set of such objects exhibiting the same properties: for example, the state `complete` below defines a set of mortgage applications that are complete.

```
sig State {
undef, def, valid, invalid: set EmploymentInfo,
incomplete, complete : set Application,
preapproved, approved, rejected : set Application,
found, notFound, appraisedOk,
appraisedKo: set PropertyInfo}
```

Accordingly, we model a state transition as removing an element from one set and adding it to another, for example:

```
complete = complete + e &&
incomplete = incomplete - e
```

The data structure `State` defines a state space for the FSM illustrated in Fig. 5. This finite state machine models the Mortgage Approval case as a whole.

We elaborate our FSMs for mortgage application and its components by introducing constraints on their states and on their possible state transitions:

```
EmploymentInfo = undef + def
undef & def = none
valid & invalid = none
(valid + invalid) & undef = none
```

This example illustrates the constraints defined for the FSM_{empl} state machine. The first two statements specify that any employment info element can be either *defined* or *undefined* but not in both states at a time; The third statement specifies that any element cannot be both *valid* and *invalid*; the last statement implements a business constraint derived from the process description in section II-B: it stands that employment info can be validated (and eventually can become *valid* or *invalid*) only if it is *defined*.

Other business properties of the mortgage approval process can be also modeled as Alloy constraints:

```
/*uniqueness of employment info */
fact { all a1, a2:Application
{a1.employment = a2.employment <=> a1=a2}}
```

This expression stands that *no two applications currently processed by the bank can have the same employment info*. It paraphrases and implements the business rule from the process description: *"A bank can process only one application from a given applicant at a time."*

C. Definition of triggering conditions.

We formalize the conditions that trigger state transitions for FSM_{app}, FSM_{empl} and FSM_{prop} shown in Fig. 2, 3, 4 with Alloy predicates. The triggering conditions are directly related to the (internal or external) activities and their outcomes. In our example, one activity can trigger different transitions depending on its outcome (e.g. completeApp). The converse is not true: each state transition in mortgage approval can be triggered by one activity only.

In the example below, we specify a triggering condition for FSM_{empl} - a *redefineEmpl* activity - a submission of new information about the employment by an applicant. The corresponding Alloy predicate is defined with two parameters: a current state `st` (the sate of a mortgage case before the activity is carried out) and a next state `st'` (the state of a mortgage case after the activity is terminated).

```
pred redefineEmpl [st, st':State] {
(some e : st.undef & Application.employment |
     st'.def = st'.def + e &&
     st'.undef = st'.undef - e ) ||
(Application.employment = st.def )
  }
```

The FSM mathematical model defines a state transition as a relation $P(Q)$. This relation is specified in the predicate's body. In our example, it states that:

some (arbitrary) instances `e` of employment info that are currently undefined for their applications (i.e. belong to the set `undef` at the current state `st`), will be defined upon redefineEmpl termination (i.e. will be moved from the set `undef` to the set `def` in the next state `st'`).

We proceed with formalizing all the state transitions for our FSMs shown in Fig. 2, 3, 4 . This results in the following Alloy predicates: *validateEmployment, searchProperty, appraiseProperty, completeApp, approveApp, rejectApp, preapproveApp* (for the complete Alloy model, please, contact the author of this paper).

We specify the final states for FSMs by introducing the following constraints:

```
fact finalValid{all s: State, s': ord/next[s],
            e:EmploymentInfo
{e in s.valid => e in s'.valid }}
fact finalApprKo{all s: State, s': ord/next[s],
            p:PropertyInfo
 {p in s.appraisedKo => p in s'.appraisedKo }}
```

The example above states that `valid` and `appraisedKo` are the final states of the corresponding FSMs (see also Fig.3, 4). The Alloy expressions can be red as follows: *once the employment info is validated, it remains valid*; *once the property got a negative appraisal - it remains so (cannot be re-appraised)*.

These constraints are derived from the business specification for mortgage approval (Section II-B). Once a new rule is introduced: *once negatively appraised, the property, can be re-appraised* (dashed transitions in Fig.4) - the `finalApprKo` invariant shall be removed.

D. Assembling the component FSMs into the Mortgage Approval FSM.

Once the component FSMs are specified, we assemble them into the Mortgage Approval FSM as illustrated in Fig. 5 by specifying the relations between their state transitions. The Alloy invariant `run` below represents such an assembly and can be seen as a global state transition function for a mortgage approval process:

```
Definition of state transitions
fact run{
  all s: State, s': ord/next[s] {
   { completeApp [ s.incomplete, s,s']
&& s'.approved = s.approved &&
...
|| {reviseApp[s.complete-s.approved-s.rejected,s, s']
   && ....
}
|| {redefineEmpl [s,s']  &&  ....
|| {searchProperty[s,s'] &&  ....
 }  }  }
```

This expression specifies two groups of activities that can be carried out during the mortgage approval: the internal activities (complete application and/or revise application) and external activities (redefine employment, search property). The corresponding predicates are related with inclusive "or" operation (`||`) allowing for concurrent state transitions. From the mortgage approval perspective, that means that during the process, the activities can be executed in any order - the essential property of CMP.

This `run` expression uses two variables - `s` and `s'` - that represent respectively a current and a next states of our mortgage approval FSM. Here ord/next is a predefined function in Alloy that specifies the 'next' state as a function of a current state in the sequence of states to be generated by the Alloy Analyzer.

Following the definition of FSM_{app}, *completeApp* represents an activity for application completion and specifies a triggering condition for the applications, which are currently in the state `s.incomplete`; similarly, *reviseApp* is applicable to complete mortgage applications for which the final decision (approved or rejected) has not been made yet. The `reviseApp` predicate specifies the revision activity with more details:

```
pred reviseApp [a: set Application, st,st':State]{
(some a1: a | preapproveApp[a1, st, st']
       && st.valid=st'.valid
      && st.invalid = st'.invalid
&& st.appraisedOk = st'.appraisedOk
&& st.appraisedKo = st'.appraisedKo) ||
(some a2: a | approveApp[a2, st, st']
    && st.valid=st'.valid
    && st.invalid = st'.invalid
&& st.appraisedOk = st'.appraisedOk
&& st.appraisedKo = st'.appraisedKo) ||
(some a3: a | rejectApp[a3, st, st']
    && st.valid=st'.valid
    && st.invalid = st'.invalid
&& st.appraisedOk = st'.appraisedOk
&& st.appraisedKo = st'.appraisedKo ) ||
(some a5: a | validateEmployment[a5, st,st']
    && st'.approved = st.approved
    && st'.rejected = st.rejected
    && st'.preapproved=st.preapproved
&& st.appraisedOk = st'.appraisedOk
&& st.appraisedKo = st'.appraisedKo) ||
(some a4: a | appraiseProperty[a4, st,st']
    && st'.approved = st.approved
    && st'.rejected = st.rejected
    && st'.preapproved=st.preapproved
&& st.valid=st'.valid
&& st.invalid = st'.invalid)
 }
```

This predicate can be red as follows: *for all applications considered for revision, the following activities can be carried out concurrently: some applications can be approved, preaproved or rejected, whereas for some applications employment can be validated and/or property can be appraised.* Here we explicitly specify that *preapproveApp*, *approveApp* and *rejectApp* do not affect the employment validity or appraisal results; *validateEmployment* does not affect the application approval status neither it does the appraisal results etc.

E. Refinement of the partial model.

Once a CMP is specified at the abstract level, we can progressively *refine* our model by defining new case elements, their corresponding states and triggering conditions. The objective of step-wise refinement is to evolve from the partial (abstract) to concrete model of the process that would fully meet its business specification. According to our formalism, the CMP model refinement consists of two parts: refinement of FSM (adding states, transitions and triggering conditions) and corresponding refinement of the Alloy model (extending data structures, adding or relaxing constraints).

In its current definition, our Mortgage Approval process model omits a lot of details: besides the employment info and property info, applicant's bank records, tax forms, medical information, loan type and duration and many other elements shall be handled during the real mortgage approval. The idea

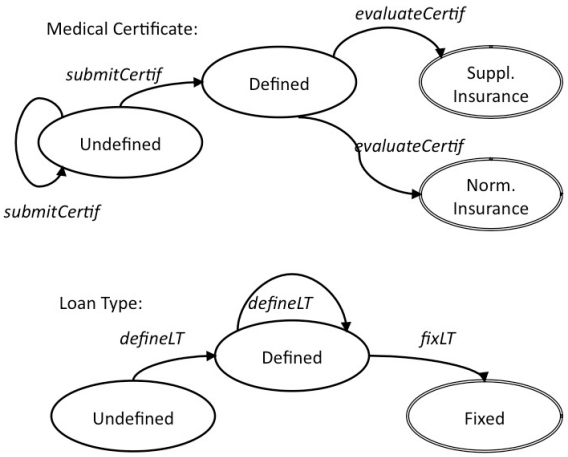


Fig. 6: A FSM for handling a medical certificate (top); A FSM for handling a loan type definition (bottom)

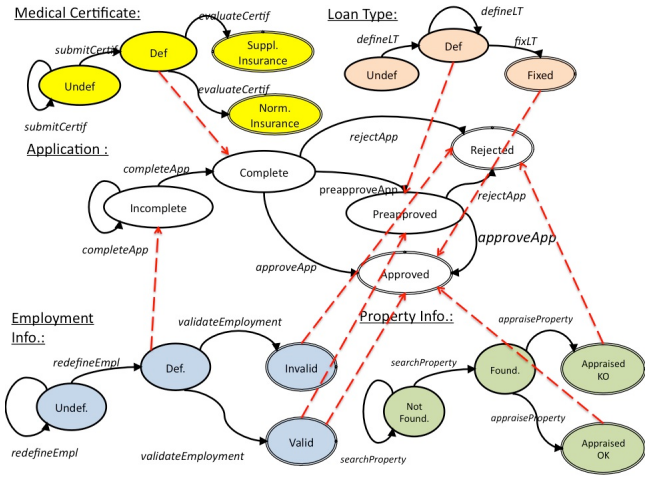


Fig. 7: A FSM for a refined mortgage approval process

is to iteratively refine our model by adding new FSMs for handling these elements and by providing their corresponding operational semantics in Alloy.

Fig. 6, 7 show a possible refinement scenario according to the following business specification parts:

- A medical certificate must be provided by the applicant upon completion of the application file. According to the evaluation results, the mortgage lender can insist on the supplementary insurance for the applicant.
- The loan type shall be defined at preapproval and fixed upon the application approval.

The refined state machine for the mortgage approval process is illustrated in Fig.7.

The choice of state space and the "relevance" of case elements is a separate topic that will not be discussed in this paper.

After each design iteration, we *simulate* and *validate* the Alloy model with Alloy Analyzer. The main objective of the step-wise refinement is to maintain the model validity while increasing its complexity.

V. Using Alloy Model Checker for Simulation and Analysis of Mortgage Approval Process Model

Once a CMP model is formalized in Alloy, it can be simulated with Alloy Analyzer.

A. Simulation

During the simulation, Alloy Analyzer generates the set of model instances representing random CMP execution scenarios. These instances are shown in a form of diagrams and provide a modeler with visual feedback. These diagrams can be examined and analyzed by both technical and domain specialists in order to detect unintended or implicit process behavior that might indicate the errors in the business specification. Along with UML, BPMN and other conventional diagrams, Alloy diagrams represent design artifacts and can serve a mean for communicating and evaluating the design decisions.

For the Mortgage Approval process model, we define a predicate `example` and configure the `run` statement for its execution:

```
pred example  { ord/last.preapproved+
ord/last.approved +
ord/last.rejected =Application }
run example for exactly 1 Application,
6 State, 1 EmploymentInfo, 1 PropertyInfo
```

The expressions above specifies a simulation of the Mortgage Approval FSM for 1 application, 1 employment info and 1 property info. The Alloy Analyzer tool will search for a scenario that

- starts in the initial state as specified by the `initState` invariant,
- follows the state transition rules for the FSM defined in the `run` invariant,
- involves exactly 6 states,
- terminates in the final state, where all the applications are at one of the 3 states: preapproved, approved or rejected (as specified in the `example` predicate).

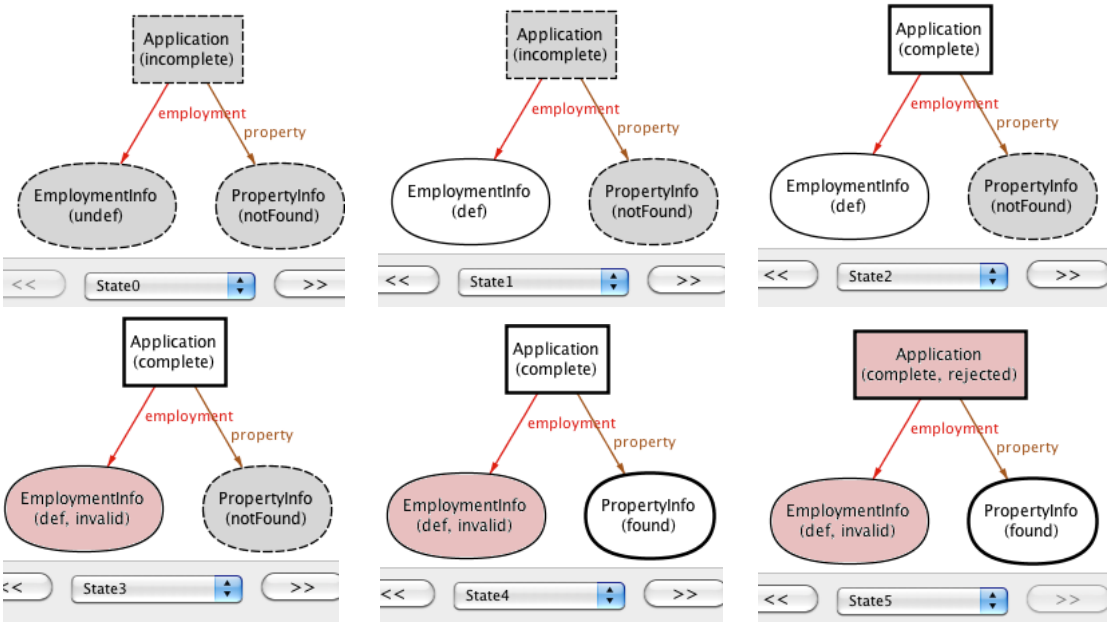
This simulation produces model instances (meaning that the model is consistent) - possible scenarios - and can be reconfigured for different number of elements and states.

Table I shows one instance generated by the Alloy Analyzer tool. The instance is projected over State in order to reproduce a sequence of state transitions generated by Alloy Analyzer. The scenario illustrates the case where the employment info is *defined* (State 1) completing the application (State 2); then the revision invalidates the employment info (State 3) and, even though the property info is submitted (State 4), the application eventually is rejected (State 5). This scenario seems correct with respect to the business specification of the mortgage approval.

The same simulation can be repeated multiple times producing different scenarios. TableII illustrates a model instance where two applications are handled at the same time:

```
run example for exactly 2 Application,
8 State, 2 EmploymentInfo, 2 PropertyInfo
```

TABLE I: Mortgage processing scenario for a single application and six states



This scenario (some states are omitted) describes simultaneous processing of two mortgage applications: one (Application0) progresses until its approval, whereas another one (Application1) is rejected. Running the simulations and analyzing the resulting scenarios - is an efficient technique for detecting errors (manifesting as abnormal scenarios) in the model.

B. Validation

Using the Alloy model checker, one can check if a CMP model corresponds to its business specification or if it has some specific business properties or meets specific requirements. We express the properties to validate as Alloy assertions.

An assertion is a logical expression that "asserts" a certain condition to be *True* in the model. The Alloy Analyzer tool examines all model instances (within a configured scope) in order to find at least one for which this condition will not hold (i.e. a counterexample). As for simulation, the counterexample is an Alloy diagram that shows the corresponding process execution scenario. This diagram can be examined and analyzed by technical and domain specialists in order to detect the problem source and correct the specification.

Below, we show several examples of model validation and improvement using Alloy Analyzer.

Example 1. Conformance to business specification.: The assertion below expresses the correct approval condition. It can be interpreted as follows: *for all applications, the fact that an application is approved implies that its corresponding employment info is valid and that its corresponding property is appraised with positive result.*

```
applApprovalOk: check {
    all st:State, a:Application  |
    a in st.approved =>
    (a.employment in st.valid &&
    a.property in st.appraisedOk)
} for  1 Application,  10 State,
4 EmploymentInfo, 1 PropertyInfo
```

We check this assertion for our current model with Alloy Analyzer and find no counterexamples. This validates the corresponding business rule for our model. Along those lines, rejection, preapproval and other business rules can be validated.

Example 2. Valid appraisal scheduling: According to the process description, a property appraisal is scheduled upon the submission of the property information by the applicant. However, carrying out the appraisal for the applicants who's employment info is already known as *invalid* does not make sense for a bank. Thus, we want to ensure that the mortgage approval process modeled above prohibits an appraisal for the applications with invalid employment info. This is a business property that we express as an Alloy assertion:

```
propAppraisalOk: check {
   all st:State,  st':ord/next[st], a:Application  |
  ((a.property not in st.appraisedOk
   + st.appraisedKo
  && a.property in st'.appraisedOk
   + st'.appraisedKo) =>
  a.employment not  in st.invalid)
} for 1 Application, 6 State,
1 EmploymentInfo, 1 PropertyInfo
```

The Alloy Analyzer tool searches for an instance that would illustrate the violation of this condition (a counterexample). This generated instance (Table III) illustrates the scenario where the appraisal is done (State 5) whereas the employment info was invalidated in the previous state (State 4) - it invalidates our assertion.

To correct the model, we revise the *appraiseProperty* predicate:

```
pred  appraiseProperty[a: set Application,
st, st':State]{
 some ap:a,  p:ap.property &
 (st.found - st.appraisedOk - st.appraisedKo) |
```

TABLE II: Mortgage processing scenario for two applications processed concurrently.

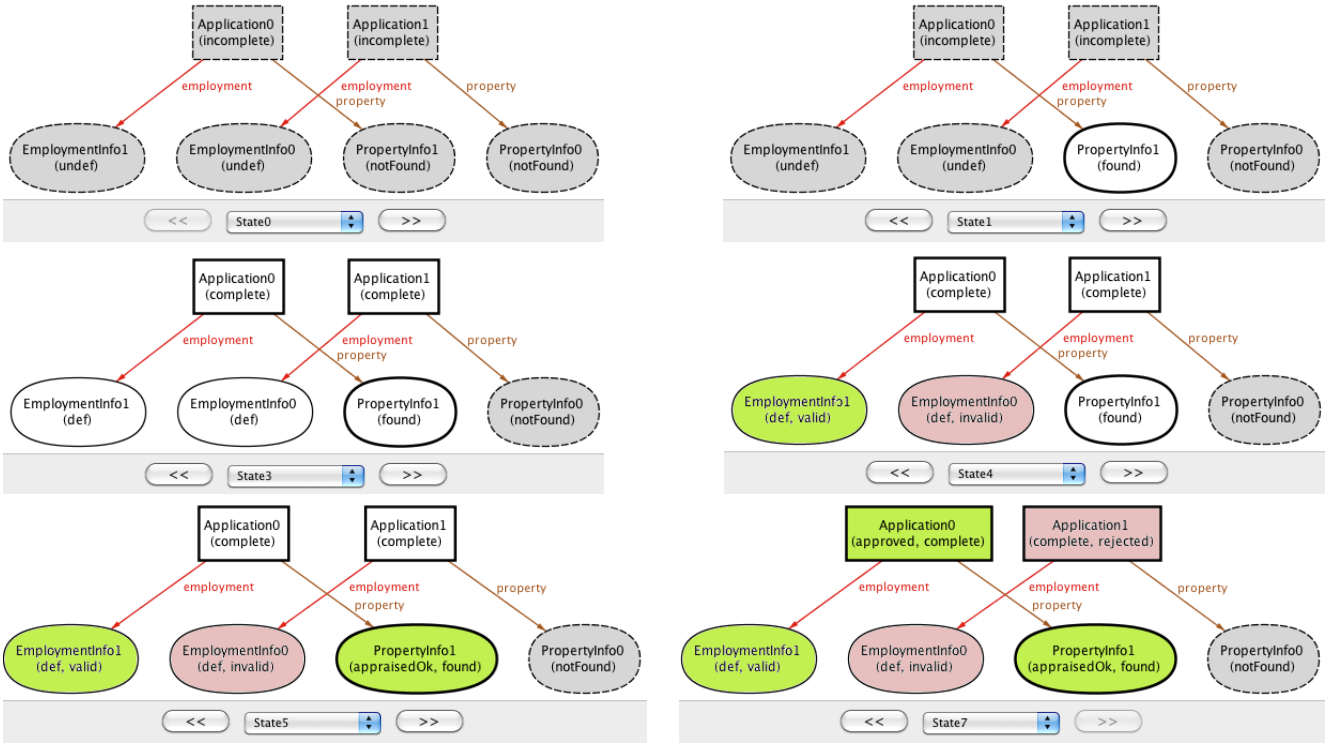
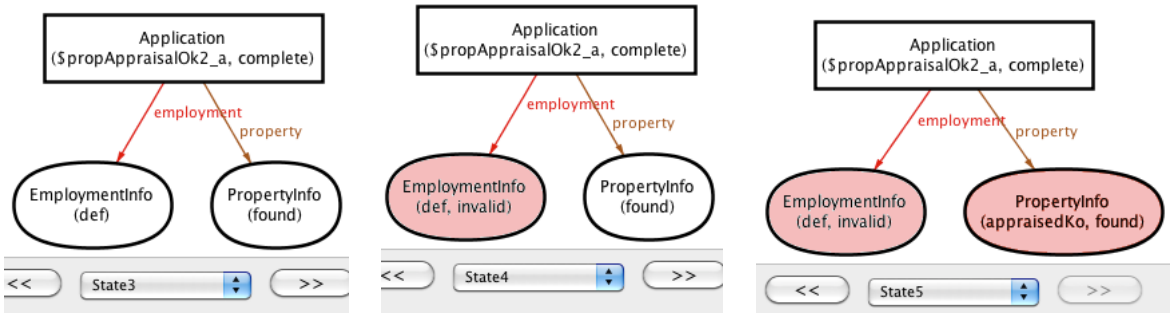


TABLE III: Counterexample: wrong appraisal scheduling.




```
//either the appraisal results are positive
(( st'.appraisedOk = st.appraisedOk + p
&& st'.appraisedKo =st.appraisedKo ) ||
// or negative
( st'.appraisedOk = st.appraisedOk
&&  st'.appraisedKo =st.appraisedKo + p ) ) ||
// or no conclusion has been made
(st'.appraisedOk = st.appraisedOk
&&  st'.appraisedKo =st.appraisedKo)
}
```

and add the corresponding constraint:

```
pred  appraiseProperty1[a: set Application,
st, st':State]{
some ap:a,  p:ap.property &
(st.found - st.appraisedOk - st.appraisedKo) |
ap.employment in st.valid  => ...
}
```

Now the assertion is validated by the Alloy Analyzer.

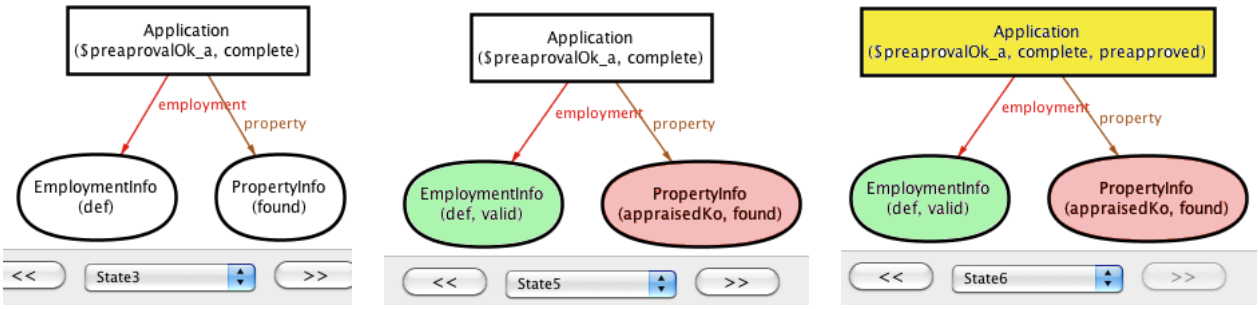
Example 3. Valid preapproval status: In this example, we check that an application with already negatively appraised property shall not be preapproved for the mortgage (even though its employment info got validated).

```
preaprovalOk: check {
   all st:State, st':ord/next[st], a:Application  |
   ( (a not in st.preapproved &&
   a in st'.preapproved) =>
   a.property not  in st.appraisedKo)
} for  2 Application,  10 State,
4 EmploymentInfo, 2 PropertyInfo
```

The counterexample illustrating the violation of this condition is shown in Table IV. We revise the *preapproveApp* predicate:

```
pred preapproveApp[a: set Application, s,s':State]{
  a.employment in s.valid &&
(s'.approved = s.approved &&
s'.rejected = s.rejected &&
s'.preapproved=s.preapproved +a)
}
```

TABLE IV: Counterexample: wrong preapproval decision



and add the constraint that states that *we do not preapprove the application if the property is appraised to Ko*.

```
a.property in (PropertyInfo - s.appraisedKo)
```

Now the assertion is validated by Alloy Analyzer.

VI. Related work

During the last decade, process flexibility and evolution support remains the central area of interest for many researchers: among others, numerous contributions of the groups at the University of Ulm (Dadam, Reichert et al) and the Eindhoven University of Technology (van der Aalst et al) can be emphasized.

The rigidity of work-flow based approaches as well as imperative modeling style is well recognized by both researchers and practitioners. In [20], van der Aalst presents a case handling paradigm to cope with business process flexibility. The differences between case handling and traditional workflow management are discussed and a metamodel for case handling is presented. Similarly to our approach, this work limits the formalization of case handling to activities, data objects, and their interrelationships. State-transition diagrams are used to specify the operational semantics of case handling systems. It proposes a definition of the case as a tuple CD = (A,P,D,dom,mandatory, restricted, free, condition) whereas we formalize the case management process using more general abstraction of finite state machine. Mandatory, restricted and free relations defined in this work can be expressed as conditions in or corresponding Alloy specifications.

The works of Pesic and v.d.Aalst attempt to change the paradigm and provide an automated support for modeling and analysis of loosely-structured processes based on the declarative principles. The Declare framework presented in [21], [22] is a constraint-based system and uses a declarative language grounded in temporal logic. The authors propose the deviation mechanisms to assure the flexibility while preserving the control over the process validity an correctness through verification at design time and performance analysis at run time: "... DECLARE can offer most features that traditional WFMSs have: model development, model verification (finding errors in models), automated model execution, changing models at run-time (i.e., adaptivity), analysis of already executed processes, and decomposition of large processes." [21]. In [23], the Adept workflow management system is presented. Adept enables the modifications on the predefined workflow at the run time. Though, becoming "more declarative", the resulting approach is still based on the imperative principles of workflow.

In the research reported in [24], the authors ask the question: "Do Workflow-Based Systems Satisfy the Demands of the Agile Enterprise of the Future?" and draw the conclusion that designing and putting into operation workflowable processes may neither be possible nor desirable in the enterprise of the future. In [25], process instances are represented as moving through state space, and the process model is represented as a set of formal rules describing the valid paths of the possible trajectories. This approach is grounded on the theory of automated control systems and implements declarative modeling principles (purely non-workflow). In [12], [14], an approach for process management based on business artifacts is presented. Business entities are changing as they move through a process. They are described with an information model and a life cycle model (i.e. a mortgage case in our example can be considered as a business entity). In [14], the operational semantics of Guard-Stage-Milestone is presented. This semantics explains the interactions between business artifacts which is formalized following declarative principles. In [26], semantics for a declarative process model based on Generic Process Model (GPM) is presented. GPM uses states as a leading element in process representation; it captures the process context and also reasons about process goals. Similarly to our approach, the proposed semantics is based on the notion of states and set theory. Formal validation and model checking, however, is not yet realized for this approach.

The claim of workflow-based, process-driven or other imperative design methodology proponents is that the imperative specifications can be validated, analyzed and controlled, assuring stable performance and predictable results. Similar results, however, can be assured by providing formal semantics for "unpredictable" declarative models and, eventually, applying the formal verification approaches.

There are two main approaches to formal verification: model checking [27] and a theorem proving based on logical inference [28]. Model checking is an approach for verifying requirements and design for a vast class of systems, including real-time embedded and safety-critical systems. Model check-

ers include such tools as [8], [29]. The major drawback of the model checking is a state explosion problem, which originates from the fact that for real systems the size of the state space grows exponentially with the number of processes [30].

The second approach is an automated theorem proving based on logical inference. Within this approach, the fact that the system specification (a model) satisfies a certain property is expressed as a logical formula. The task is to prove the validity of this formula, deducing it from a set of axioms exist for the underlying logic (e.g. first-, second-, higher-order logic etc), and hypotheses made about the system. Theorem proving for the first-order logic is well developed; higher order and other logics are more expressive and appropriate for wider range of problems then first-order logic; however the automated theorem proving for these logics is more complicated [31], [32] .

In spite of their effectiveness, approaches based on a formal semantics, model checking and verification using theorem proving are rarely used in practice due to the high cost. However, we agree with Colin Snook and Michael Butler that "Formal specification is the first step to using formal methods and is, in itself, a useful activity even if a fully formal development process is not followed." [33], [34].

VII. Conclusion

In theoretical computer science, the operational semantics explains the meaning of a program "in terms of a hypothetical computer which performs the set of actions which constitute the elaboration of that program." [18]

In this work, we explained the meaning of a case management processes (CMP) in terms of a finite state machine (FSM), which performs the set of actions which constitute the case handling. We formalized our operational semantics in the Alloy specification language.

We implemented the data-centered view (Fig. 1b) and proposed the model that prescribes no control flow but lets the process unfold according to a particular case circumstances.

We illustrated our findings on the example of Mortgage Approval process: first, we represented this process as an assembly of three independent FSMs handling the mortgage application, applicant's employment info. and property info.; then, we demonstrated how these FSMs and their assembly can be specified in Alloy and analyzed with the Alloy Analyzer tool.

We illustrated how *model simulation* and *model checking* with the Alloy Analyzer tool can drive the case management process design:

The advantage of simulation is a possibility to get an instant visual feedback in a form of diagrams capturing the FSM execution scenarios. Model checking enables more specific reasoning about model properties.

Examining the model instances generated by Alloy Analyzer, we were able to reason about correctness of our partial model; specifying and checking the Alloy assertions, we were able to validate business properties of our model (e.g. correct conditions for application approval and preapproval, correct appraisal scheduling, etc.).

Once the model validity is checked at a given level of detail, the model can be *refined*. According to our formalism, the refinement consists of two parts: definition of the new FSMs and formalization of these FSMs in Alloy. This approach allows us to efficiently manage model complexity.

Alloy specification language was successfully applied in various domains including software and hardware system specification, automatic model completion, service testing, etc.[3] Its use in business domain, however, is rather limited. In this work, we position Alloy and Alloy Analyzer as a toolbox for both technical and business specialists: together with UML and BPMN diagrams, Alloy diagrams represent design artifacts and can be studied, referred to, and serve a mean for communicating and evaluating both business and technical design decisions.

In the mortgage approval example presented in this paper, we consider that a state transition can be triggered by a specific outcome of a *single* activity (i.e. no two activities can have the same outcome and trigger the same state transition). Possible effects of contextual events on state transitions (e.g. an economic crisis may affect a mortgage approval decision) are not considered by this example. In general case, a triggering condition can be seen as *a combination of contextual events and multiple outcomes of several (internal or external) activities*. Therefore, a user can have a choice of activity to execute. In [35], we discuss context modeling, formal concept analysis and Galois lattices as a background theory for context-aware and agile process management. Definition of automated user guidance for CMP is a subject of our future work.

The operational semantics defined in this work is a first step to using formal methods; it paves the road to automated design, validation and verification of CMP. This work, however, does not suggest any concrete (graphical) modeling notation for case management processes as we convinced that visual syntax can be freely chosen according to the modeler's taste as soon as the semantics that has to be encoded by this syntax is well established.

References

[1] Davenport, T., Nohria, N.: Case management and the integration of labor. Sloan Management Review **35** (1994) 11–11

[2] Davenport, T.: Thinking for a living: How to get better performances and results from knowledge workers. Harvard Business Press (2005)

[3] Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7) (July 1978) 558–565

[4] Rychkova, I.: Towards automated support for case management processes with declarative configurable specifications. [16] 65–77

[5] Plotkin, G.: A structural approach to operational semantics. (1981)

[6] Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. **3**(2) (April 1959) 114–125

[7] Jackson, D.: Software Abstractions- Logic, Language and Analysis. MIT Press (2011)

[8] Jackson, D.: Alloy Analyzer tool. http://alloy.mit.edu/alloy/ (2011)

[9] OMG: Case management process modeling (cmpm) request for proposal. http://www.omg.org/cgi-bin/doc?bmi/09-09-23 (2009)

[3] See `http://alloy.mit.edu/alloy/applications.html`.

[10] Swenson, K.: Mastering The Unpredictable: How Adaptive Case Management Will Revolutionize The Way That Knowledge Workers Get Things Do. Meghan-Kiffer Press (2010)

[11] Pucher, M.: The elements of adaptive case management. Mastering the Unpredictable (2010) 89–134

[12] Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal **42**(3) (2003) 428–445

[13] Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering **32**(3) (2009)

[14] Hull, R., Damaggio, E., De Masellis, R., Fournier, F., Gupta, M., Heath III, F.T., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., et al.: Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events. In: Proc. of DEBS. (2011)

[15] Herrmann, C., Kurz, M.: Adaptive case management: Supporting knowledge intensive processes with it systems. In Schmidt, W., ed.: S-BPM ONE - Learning by Doing - Doing by Learning. Volume 213 of Communications in Computer and Information Science. Springer Berlin Heidelberg (2011) 80–97

[16] Rosa, M.L., Soffer, P., eds.: Business Process Management Workshops - BPM 2012 International Workshops, Tallinn, Estonia, September 3, 2012. Revised Papers. In Rosa, M.L., Soffer, P., eds.: Business Process Management Workshops. Volume 132 of Lecture Notes in Business Information Processing., Springer (2013)

[17] Rychkova, I., Nurcan, S.: Towards adaptability and control for knowledge-intensive business processes: Declarative configurable process specifications. In: System Sciences (HICSS), 2011 44th Hawaii International Conference on, IEEE (2011) 1–10

[18] Wijngaarden, A.v.: Report on the algorithmic language ALGOL 68. Printing by the Mathematisch Centrum (1969)

[19] Rychkova, I.: Formal Semantics for Refinement Verification of Enterprise Models. PhD thesis, EPFL (2008)

[20] Van der Aalst, W., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. Data & Knowledge Engineering **53**(2) (2005) 129–162

[21] Pesic, M., Schonenberg, H., van der Aalst, W.: Declare: Full support for loosely-structured processes. In: Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International, IEEE (2007) 287–287

[22] van der Aalst, W., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science-Research and Development **23**(2) (2009) 99–113

[23] Reichert, M., Rinderle, S., Dadam, P.: Adept workflow management system. In: Business Process Management. Springer (2003) 370–379

[24] Bider, I., Johannesson, P., Perjons, E.: Do workflow-based systems satisfy the demands of the agile enterprise of the future? [16] 59–64

[25] Bider, I.: Towards a non-workflow theory of business processes. [16] 1–2

[26] Soffer, P., Yehezkel, T.: A state-based context-aware declarative process model. In: Enterprise, Business-Process and Information Systems Modeling. Springer (2011) 148–162

[27] Dijkstra, E., Dijkstra, E., Dijkstra, E.: Notes on structured programming. Technological University, Department of Mathematics (1970)

[28] Gordon, M.J.C., Melham, T.F., eds.: Introduction to HOL: a theorem proving environment for higher order logic. Cambridge University Press, New York, NY, USA (1993)

[29] Holzmann, G.J.: The SPIN Model Checker - primer and reference manual. Addison-Wesley (2004)

[30] Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Progress on the state explosion problem in model checking. In: Informatics, Springer (2001) 176–194

[31] Weidenbach, C., Brahm, U., Hillenbrand, T., Keen, E., Theobald, C., Topić, D.: Spass version 2.0. Automated DeductionCADE-18 (2002) 45–79

[32] Paulson, L.: Isabelle: A generic theorem prover. Volume 828. Springer (1994)

[33] Snook, C., Butler, M.: Uml-b: Formal modeling and design aided by uml. ACM Trans. Softw. Eng. Methodol. **15**(1) (January 2006) 92–122

[34] Snook, C., Savicks, V., Butler, M.: Verification of uml models by translation to uml-b. In: Proceedings of the 9th international conference on Formal Methods for Components and Objects. FMCO'10, Berlin, Heidelberg, Springer-Verlag (2011) 251–266

[35] Rychkova, I., Kirsch-Pinheiro, M., Le Grand, B.: Context-aware agile business process engine: Foundations and architecture. In: Proceedings of the 14th Business Process Modeling, Development, and Support (BPMDS'13) working conference. (2013) To appear