# A Formalization of UML AD Refinement Patterns in Event B

Ahlem Ben Younes
Research Unit of Technologies of Information and Communication (UTIC)- ESSTT
Tunisia
Ahlem.benyounes@fst.rnu.tn

Leila Jemni Ben Ayed
Research Unit of Technologies of Information and Communication (UTIC)- ESSTT
Tunisia
Leila.jemni@fsegt.rnu.tn

***Abstract*— In this paper, we propose a specification and verification technique using the combination UML Activity Diagrams (AD) and Event B to both improve graphical representation of the workflow application structure and its functions. Its required properties are also verified. The workflow is initially expressed incrementally graphically with UML AD, then translated into Event B and verified using the B powerful support tools. The Event-B expression of the UML AD model allows us to give it a precise semantics. We propose a workflow applications constructive approach in witch Event B models are built incrementally from UML AD models, driven by UML AD refinement patterns. The use of the B formal method and its refinement mechanism allows the verification of the correction of the UML AD refinement patterns.**

***Keywords-Progressive Development; Workflow Applications; Specification; Refinement UML AD; Patterns; Event-B; Formal Verification.***

## I. INTRODUCTION

The Workflow Applications (WA) are characterized by a high complexity. Increasingly, they became omnipresent in the critical calculation domain (natural or industrial disasters) and they have to obey to the realiabity and safety requirements. Thus, the need of an adequate software specification technique and a suitable development method is increased. The used specification formalisms need to be comprehensive, expressive, and precise.

A workflow is a set of activities (tasks/ process) that are ordered according to a set of procedural rules to achieve a result or a goal. A workflow model (workflow specification) is the definition of a workflow. A workflow is either an atomic task, known as elementary task/activity or a sub-workflow (nesting), a composite activity/task.

Traditional workflow models have obvious shortcomings in describing complex workflows. Such complexity is due not only to the hierarchical property of business process, but also to the complicated dependencies among tasks. Composition is an important approach to model larger and more complex workflow application. Task refinement is one kind of workflow composition approaches.

Indeed, specifying a complex system is a difficult task, which cannot be done in one step. The stepwise refinement technique facilitates the understanding of complex systems by dealing with the major issues before getting involved in the details. It consists of developing the system through different levels of abstraction. The system under development is first described by a specification at a very high level of abstraction. A series of iterative refinements may then be performed with the aim of producing a specification, consistent with the initial one, in which the behavior is fully specified and all appropriate design decisions have been made. Stepwise software development can be fully exploited only if the language used to create the specifications is equipped with formal refinement machinery, making it possible to prove that a given specification $S_C$ is a refinement of another specification $S_A$.

The Unified Modeling Language Activity Diagrams (UML AD) [3] are considered as an Object Management Group (OMG) [15] standard notation in the area of workflow applications modelling. The idea of one standard language for modelling provides many advantages to software development, such as simplified training and unified communication between development teams.

In our work, an UML activity diagrams approach based on stepwise refinement technique for the workflow specification is proposed. The refined workflow is presented in UML using the hierarchical capabilities of the UML Activity Diagram notation [4][5]. Workflow's hierarchy comes from the hierarchy of processes goals (Task). Goals or activities of workflow applications are organized in a hierarchy obtained from the Sequence/And/Or refinement of higher level activities (goals) into lower-level activities (atomic task). To describe the decomposition of the activity, we propose some patterns that allow to model some refinement of activity: sequence, choice (OR), parallel (AND), loop. Thus, the description of the workflow at different levels of abstraction becomes possible. In addition, our objective is to provide a specification and verification technique for workflow applications using UML AD, which give readable models and an appropriate formal method allowing verification of required properties (such no deadlock) to prove the correctness of the workflow specification.

Indeed, the main problem with UML activity diagrams is that they have no formal semantics and in consequence UML AD does not allow the formal verification of functional workflow applications properties (safety, deadlock-inexistence, liveness, fairness, etc) and the correction of the patterns.

On the other hand, the Event B method [2] is a variant of the B formal method [1], proposed by Abrial to deal with

 ISBN: 978-1-61208-158-8 

distributed, parallel and reactive systems [2]. The concept of refinement is the key notation for developing B models. The refinement of a formal model allows one to enrich the model in step by step approach. The last refinement gives the implementation machine, which map directly to a programming language such as C or ADA. B models provide an automatic proof, which convinces the user that the system is effectively correct and satisfies properties, which are presented as invariants/assertions. The strong point of B is support tools like as AtelierB [6] or B4free [7], an academic version of AtelierB. Most theoretical aspects of the method, such as the formulation of Proof Obligations (PO), are carried out automatically. The automatic and interactive provers are also designed to help designer to discharge the generated proof obligations. All of these points make B well adapted to large scale industrial projects [8]. However, B is still difficult to learn and to use. In addition, there is a lack of methodological studies related to the incremental development of complex system using the refinement mechanisms.

This is why a graphical representation of B models is required. For that purpose, we propose a constructive approach in witch Event B models are built incrementally from UML AD models, driven by UML refinement patterns.

Our work presents a specification and verification technique using the combination UML AD and Event B to both improve graphical representation of the workflow application structure and its functions such as the complex properties, and also verify required properties.

In our approach, the workflow is initially expressed incrementally graphically with UML AD refinement patterns, then translated into Event B and verified using the B powerful support tools.

The Event-B expression of the UML AD model allows us to give it a precise semantics. In this context, there have been efforts for defining semantics for activity diagram in the works of Eshuis [10][11]. However, these works not consider the hierarchical decomposition of activities in UML AD. In addition, from the validation point of view, in our approach, the verification of WA is based on a proof technique and therefore it does not suffer from the state space explosion occurring in classical model checking as in the cases of works in [9] [10] [11] and [12].

Our contribution, in this context, consists of using Event B method and its associate refinement process to encode the hierarchical decomposition of activities in UML AD: Each decomposition of a complex activity in UML AD is translated into Event B by refining the event corresponding to this activity. This refinement introduces the decomposition defined in the original UML AD workflow specification. Thus, a step by step UML AD workflow description and validation is performed in parallel.

Refinement allows the developer to express the relevant properties at the refinement level where they are expressible. Then, further refinements will preserve these properties avoiding reproving them again.

The use of the B formal method and its refinement mechanism allows the verification of the correction of the UML AD refinement patterns by the use of the B support tools.

This paper continues our previous works [4][5] by addressing the Event-B formalization of UML AD patterns (sequence, parallel, choice) for workflow applications with additional studies, results and proofs. In [4][5], we have only proposed translations rules for UML AD notation into Event B models. In these innovative works, we propose a formal framework to define refinement patterns for UML AD. We define an Event-B semantic for each UML AD refinement pattern for WA by constructing set-theoretic mathematical models (See Section 4 and 5). Based on the classical set of inference rules from Event-B [13], we identify the systematic proof obligations for each UML AD activity refinement pattern. The use of the B formal method and its refinement mechanism allows the verification of the correction of the patterns. The Event-B formalization of the other UML AD models is a work in progress.

The remainder of the paper is organized as follows: Section 2 presents a brief overview of the semi-formal UML activity diagrams notation. Section 3 presents a brief overview the formal Event B method. Section 4 details our proposed approach that consists in expressing a UML AD model with Event-B. Section 5 illustrates the approach by presenting the Event-B formalization of the sequence refinement pattern. Finally, a summary of our work concludes the paper.

## II. UML ACTIVITY DIAGRAMS

An activity diagram is a variation of a state machine in which the states represent the execution of actions or subactivities and the transitions are triggered by the completion of the actions or subactivities. We use activity diagrams to model computational, communication, and synchronization operations/process of parallel and distributed applications. Moreover, we use the hierarchical decomposition (thanks to refinement) offered by UML activity diagrams to model complexes applications gradually in incremental way on several levels (see Figure 2). An action state is used to model a step in the execution of an algorithm (atomic action), or a workflow process (Subactivity represents a composed activity). A subactivity state invokes an activity diagram. When a subactivity state is entered, the activity diagram nested in it, which corresponds to the refined activity, is executed. A subactivity state is shown in the same way as an action state with the addition of an icon in the upper left corner depicting a nested activity diagram (see Figure 1.(b)). Transitions are used to specify that the flow of control (the token) pass from one action to the next. An activity diagram expresses a decision when guard conditions are used to indicate different possible transitions (see Figure 1.(a)). A guard condition specifies a condition that must be satisfied in order to enable the firing of an associated transition. A merge has two or more incoming transitions and one outgoing transition. It can be used to merge decision branches back together. Fork and join are used to model parallel flows of control (see Figure 1.(b)). The initial and final state are, respectively, visualized as a

 ISBN: 978-1-61208-158-8 

solid ball and a solid ball inside a circle. Figure 1.(a) illustrates how to model a loop by employing an activity diagram, whereas Figure 1.(b) shows one option for modeling the parallel execution of two activities.

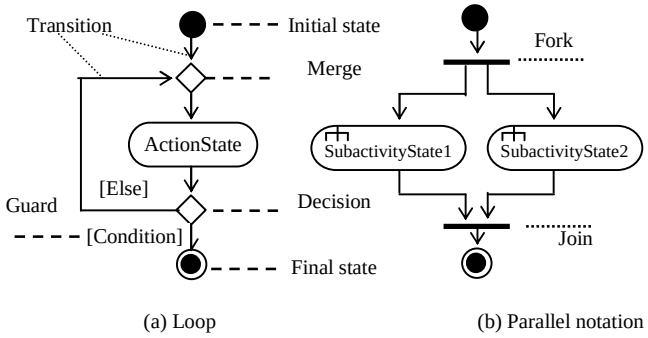


Figure 1. UML activity diagram notation

One of the main features of UML is the refinement with hierarchical decomposition of activities in UML AD, which permits to obtain a detailed specification from an initial specification. Figure 2 illustrates how to model complex application like distributed and parallel application, on several levels, by employing a refinement technique of UML AD.

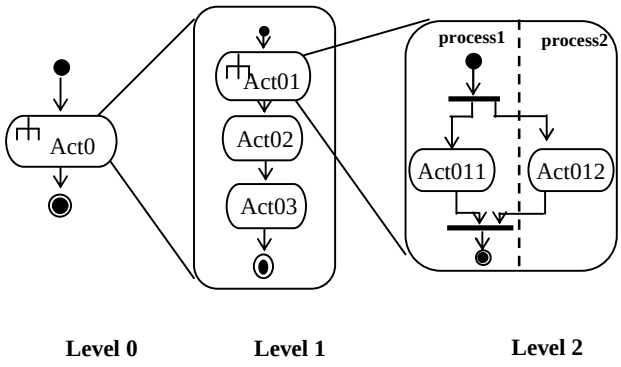


Figure 2. Hierarchical decomposition of activities in UML activity diagrams

Our choice of UML AD is motivated by the fact that workflow modelling is strongly supported by UML through activity diagrams [1]. Moreover, UML is easy to read and understand by human beings.

In the Section 4, we show how the semantics of activity diagrams can be formally described in Event B.

## III. EVENT B METHOD

We use the B method [1] and its event-based definition [2] to formalize UML AD models of workflow application.

**Event B model**: Development in the Event B method is based on the concept of model [2]. A model shown below is composed of:

**MODEL** < *name*>
**VARIABLES**
<*variables*>
**INVARIANT**
< *invariant*>
**ASSERTIONS**
< *assertion*>
**INITIALISATION**
<*initialization of variables*>
**EVENTS**
< *events*>
**END**

- Descriptive specification, which describes what the system does using a set of variables, constants, properties over constants and invariants which specify required properties to be verified in each state. This constitutes the static definition of the model.

- Operational specification, which describes the way how the system operates, it is composed of an initial state and various transitions (events) which show how the set of variables of the descriptive specification can move in time.

An Event B model is composed of set atomic events described by particular generalized substitution (**ANY**, **BEGIN** and **SELECT**). Each event Evt is fired if the guard P associated to this event is true. For the purpose of this paper, we will only use the **SELECT** substitution Evt= **SELECT** P **THEN** G **END**. Moreover, a B model contains a set of properties i.e invariants, liveness, safety and reachability properties which can be prove during the development thanks to the embedded proof system associated to B and the tool supported by B4free [6].

**Refinement of Event B models:** Each Event B model can be refined. A refined model is defined by adding new events, new variables and a gluing invariant. Each event of the abstract model is refined in the concrete model by adding new information describing how the new set of variables and the new events evolve. All the new events appearing in the refinement refine the skip event of the refined model.

 ISBN: 978-1-61208-158-8 

The gluing invariant ensures that the properties expressed and proved at the abstract level (in the **ASSERTIONS** and **INVARIANTS** clauses) are preserved in the concrete level. Moreover, **INVARIANT, ASSERTIONS** and **VARIANT** [14] clauses express deadlock and livelock.

1. They shall express that the new events of the concrete model are not fired infinitely (no livelock). A decreasing variant is introduced for this purpose.
2. They shall express that at any time, an event can be fired (no deadlock). This property is ensured by asserting (in the **ASSERTIONS** clause) that the disjunction of all the abstract events guards implies the disjunction of all the concrete events guards.

At every step of the refinement, proof obligations ensure that events and initialization preserve the system invariant. A set of proof obligations that is sufficient for the correctness must be discharged when a refinement is postulated between two B components [2] [14].

A strong point of the B method is that the B support tools like B4free [7] provides utilities to discharge automatically the generated proof obligations (of the invariant preservation and the refinement correctness). Analyzing the non-discharged proof obligations with the B support tools is an efficient and practical way to detect errors encountered during the specification development.

Moreover, in the refinement, it is not needed to reprove these properties again while the model complexity increases. Notice that this advantage is important if we compare this approach to classical model checking where the transition system describing the model is refined and enriched.

Finally, the choice of Event-B is due to its similarity and complementarity with UML AD: both Event-B and UML AD have the notion of refinement (constructive approach).

## IV. THE PROPOSED APPROACH

### A. *Presentation*

Our approach relies on the following steps:

- Step 1: Initially, the workflow is modeled graphically with UML AD refinement patterns.
- Step 2: For current decomposition level, the resulting graphical readable model is translated into Event B applying the approach described in [4][5].
- Step 3: This Event B model is enriched by relevant properties (no deadlock, no livelock, etc) which are defined in the INVARIANTS and ASSERTIONS clauses. These properties will be proved using the B4free tool [7].
- Step 4: We isolate the events of the Event B model whose POs, associated to the introduced invariant of step 3, are not provable.
- Step 5: The UML AD model of step1 is re-design by introducing a UML AD scope embedding the events identified at step 4 and a compensation/fault handler component.
- Step 6: Apply step 2.

This step-based approach is applied until the associated Event B model is free of unproved PO.

### B. *Formalization of UML AD*

To achieve our objective, we formalize with Event-B the UML AD refinement patterns that analysts use to generate a UML AD workflow hierarchy.

The UML AD language [3] offers two categories of activities:

**1) Atomic activities (action)** representing the primitive operations performed by the process. They are defined by action node in UML AD.

**2) Composed activities** representing the sub-workflows (nesting), obtained by composing primitive activities and/or other composed activities using the sequence, parallel (For/Join), choice (Decision/ Merge) control constructs in UML AD.

In the remaining of this paper, we refer to the decomposition (refinement) of a composed activity by the activities it contains as a result of the refinement operation using refinement patterns.

In this innovative work, the formal assertion defining an activity A is written in first-order logic. Thus, the general form of the assertions associated to the activities is A-Pre => A-post where A-Pre and A-Post are predicates associated to an activity A (See Figure3). Symbol => denotes the classical logical implication. Such assertions state that from a state in which A-Pre holds, we must reach another state in which A-Post holds.

If we refer to the concepts of guard and postcondition that exists in Event-B, a UML AD activity can be considered as a postcondition of the system, since it means that a property must be established. Following our pervious works [4], we have proposed to express each UML AD activity as a B event, where the action represents the achievement of the activity. Then, we will use the Event-B refinement relation and additional custombuilt proof obligations to derive all the subactivity of the system by mean of B events.

At the high level of abstraction, there is only one event for representing the parent activity. In accordance with the Event-B semantics, if the guard of the event is true, then the event necessarily occurs. For the new events built by refinement and associated to the subactivity, we guarantee by construction that no events prevent the postconditions to be established. For that, we have proposed an Event-B semantic for each UML AD refinement pattern by constructing set-theoretic mathematical models. Based on the classical set of inference rules from Event-B [13], we have identified the systematic proof obligations for each UML AD activity refinement pattern.

 ISBN: 978-1-61208-158-8 

To better illustrate the approach, the next section presents just the Event-B refinement semantics related to the sequence refinement pattern.

## V. THE FORMALIZATION OF THE UML AD SEQUENCE REFINEMENT PATTERN

The sequence activity refinement pattern refines a composed activity by introducing intermediate sequence states A01,..., A0n for reaching a state satisfying the target condition (denoted by A0-Post) from a state satisfying the current condition (denoted by A0-Pre) as shown in Figure 3 (with just two sub-activity).

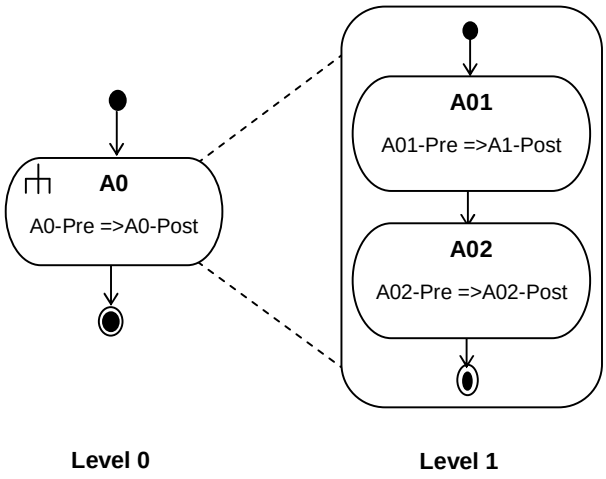


Figure 3. Sequence activity refinement pattern

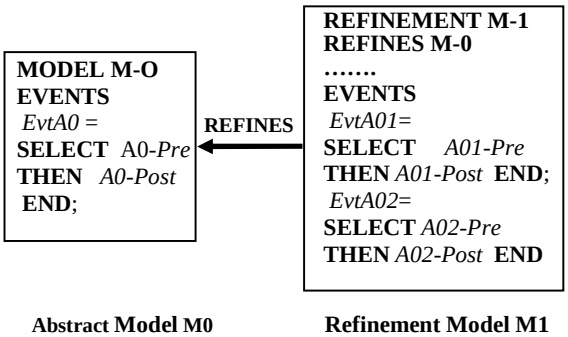


Figure 4. Overview of the Event-B representation of the UML AD model

### A. Description

The first sub-activity A0 is an activity with the sequence condition as target condition; it states that the sequence condition (denoted by A0-Pos) must hold if.

The specific current condition A01-Pre (which can be larger than the current condition A0-Pre of the parent activity) holds in the current state. The second sub-activity states that the specific target condition A02-Post (which can be larger than the target condition A0-Post of the parent goal) must hold if the specific sequence condition A02-Pre (derived from A01-Post) holds in the current state.

### B. Formal definition

As explained in the last section, each level i ($i \in [0..n]$) is represented in the hierarchy of the UML AD models as an Event-B model Mi that refines the model $Mi-1$ related to the level $i - 1$. Moreover, we represent each activity A j,i ($j \in$ [0,..,n] activity index ) as a B event EvtAi,j , where the guard is the transcription of A-Guard from the activity expression, and the **THEN** part is the translation into Event-B of A-Post (see Figure 4).

### C. Proof obligations identification

We are going to give systematic rules defining exactly what we have to prove for this pattern in order to ensure that each concrete event (EvtA01, EvtA02) indeed refines its abstraction EvtA0. In fact, we have to prove three different lemmas:

- The ordering constraint (PO1) expresses the sequence characteristic between the Event-B events. PO1 ensures that the target condition of the activity A01 implies the current condition of the activity A02.

$$A01\text{-}Post \Rightarrow A02\text{-}Pre \quad \textbf{(PO1)}$$

- The guard strengthening (PO2) ensures that the concrete guard is stronger than the abstract one. In other words, it is not possible to have the concrete version enabled whereas the abstract one would not. The term “stronger” means that the concrete guard implies the abstract guard.

$$A01\text{-}Pre \Rightarrow A0\text{-}Pre \quad \textbf{(PO2)}$$

- The correct refinement (PO3) ensures that the sequence of concrete events transforms the concrete variables in a way which does not contradict the abstract event.

$$A02\text{-}Post \Rightarrow A0\text{-}Post \quad \textbf{(PO3)}$$

 ISBN: 978-1-61208-158-8 

The Event-B refinement semantics of the sequence refinement pattern requires to prove three proof obligations ((PO1) (PO2) (PO3)) that could easily be discharged by the current version of B4free or Rodin automatic theorem prover.

## VI. CONCLUSION

We have proposed a specification and verification technique for workflow applications using UML AD and Event B. The workflow is at first modelled with UML AD refinement patterns, which is understandable allowing communications with costumers, then translated the resulting model into Event B, which is enriched by relevant properties (Safety, nodeadlock) to be verified using powerful support tool B4free[7]. This approach allows to rigorously verify UML specifications by analysing derived B specifications and to prove that the modelled workflow using the AD respects all safety and reliability constraints by the formal verification of its properties. Analyzing derived B specifications (thanks to B4free tool) is a practical and rigorous way to improve initial UML AD specifications.

Our contribution consists in the use of the Event B method and its associate refinement process to encode the hierarchical decomposition of activities in UML AD and its tools for the formal verification of workflow applications. In addition, the strong point in our approach is that the validation can be performed at any development stage and particularly at early steps allowing saving at development. Thus, a step by step UML AD workflow description and validation is performed in parallel.

For an incremental development of AW using UML AD, we have proposed some activity refinement patterns (sequence, parallel, choice). In this paper, we have proposed a formal framework to define refinement patterns for UML AD. The use of the B formal method and its refinement mechanism allows the verification of the correction of the patterns by the B support tools.

In contrast to the works of Eshuis [10] [11], Karamanolis [12] and Van der Aalst [9], in our works, the verification is based on a proof technique and therefore it does not suffer from the state number explosion occurring in classical model checking as in the cases of their works.

Actually, we are actively working on the extension of our works to investigate new refinement patters presented in [4] and to generalize our method. In addition, in future work, we envisage the formal validation of our transformation rules.

## REFERENCES


[1] J.-R, Abrial, “The B Book. Assigning Programs to Meanings”. Cambridge University Press, 1996.

[2] J.-R.Abrial. ”Extending B without changing it” (for developing distributed systems)”. In H Habrias, editor, First B Conference. 1996.

[3] R. Johason, I. Jacobson, and G. Booch, “The Unified Modelling Language reference Manual” .Addison- Wesley. 1998.

[4] A. Ben Younes, and L.-Jemni,Ben Ayed “ Using UML Activity Diagrams and Event B for Distributed and Parallel Applications”. In 31st Annual IEEE International Computer Software and Applications Conference .COMPSAC 2007: pp, 163-170.

[5] A. Ben Younes, and L.-Jemni, Ben Ayed, “Specification and verification of Workflow Applications using Combination of UML Activity Diagrams and Event B’’. In The 5th International Conference on Software Engineering and Data Technologies. ICSOFT 2010: pp 312-316.

[6] Clearsy, “ System Engineering Atelier B”. Version 3.6. ,2001

[7] Clearsy“B4free,” Available at www.b4free.com . (June 30, 2011).

[8] P. Behm, P. Desforges, and J.-M. Meynadier. “MÉTÉOR: An Industrial Success in Formal Development”. April 1998. An invited talk at the 2nd Int. B conference, LNCS 1939.

[9] W.M.P. van der Aalst. “Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques”. In Business Process Management: Models, Techniques, and Empirical Studies, volume 1806 of Lecture Notes in Computer Science, pp 161-183. Springer-Verlag, Berlin, 2000.

[10] R. Eshui and R. Wieringa. “Tool Support for verifying UML Activity Diagram”. IEEE transaction on software Engineering , volume 30 , N°7; pp 437-447. 2004.

[11] R. Eshuis and R. Wieringa. “ A formal semantics for UML activity diagrams”. Technical Report TR-CTIT-01-04, Centre for Telematics and Information Technology, University of Twente, 2001.

[12] C. Karamanolis, D.Giannakopoulou, J. Magee, and S. M.Wheater “Formal verification of workflow schemas”. University of Newcastle, Technical Report. 2000.

[13] J.-R. Abrial. Chapter 2 of the forthcoming book: "Modeling in Event-B: System and Software Engineering Forthcoming book". http://www.event-b.org/A ch2.pdf.

[14] C. Metayer, J,-R. Abrial, and L. Voisin. ”Event B language’’, Technical Report D7, RODIN Project Delivrable. 2005.

[15] Object Management Group (OMG), http://www.omg.org. (June 30, 2011).


 ISBN: 978-1-61208-158-8 