Classification of REA Patterns using Formal Specifications

Vahid R. Karimi, Donald D. Cowan
David R. Cheriton School of Computer Science
University of Waterloo, Waterloo, ON, Canada
vrkarimi@cs.uwaterloo.ca, dcowan@csg.uwaterloo.ca

1 Abstract

This paper describes an approach, which uses formal specifications, for the classification of REA patterns and their building blocks. In general, classification is essential – especially when the number of patterns of a collection increases. Budgen [Bud03] stresses the need and importance of exploring and evaluating "the most effective ways to index and classify patterns." The number of patterns increases over time, and there is always the possibility of having either duplicate patterns or patterns whose relationships can be interpreted differently if these relationships are not clearly defined. In this short paper, we describe our approach in which REA patterns are specified formally using Alloy prior to being classified in the Zachman framework. For convenient references, these REA specifications are assigned numbers.

Key Words: Alloy, (Business) Patterns, Formal Specifications, Logic, Resource-Event-Agent (REA), REA Ontology, Unified Modeling Language (UML), Zachman Framework

2 Introduction

To specify REA patterns, we use Alloy [Jac00, Jac], based on a relational logic (i.e., a logic that combines the operators of the relational calculus with the quantifier of first-order logic [Jac06]). Table 1 shows a sketch of the Zachman framework [Zac87, SZ92], used for classifying REA patterns, entities, and their relationships. Our focus is on the designer view of this framework as highlighted in Table 1.

2.1 Formal REA Pattern Specifications

We first provide a brief introduction to our specification by describing a general REA exchange pattern in Alloy (Figure 1). More details of Alloy can be found in several sources (e.g., [Jac00, Jac, Jac06]).

The first line in Figure 1 represents the relationship "inflow" between "IResource" and "IncrementEevent." This relationship is indicated by an arrow (->) with the name "inflow." The resource, connected by an inflow relationship, is prefixed with an "I" and identified as IResource. "One" and "some" are relationship multiplicities;

	Data	Function	Network	People	Time	Motivation
Planner						
Owner						
Designer						
Builder						
Subcontractor						

Table 1: A Sketch of the Zachman Framework (Our focus is highlighted.)

Alloy has four relationship multiplicities: *set* (any number), *one* (exactly one), *lone* (zero or one), and *some* (one or more) [Jac06]. Similarly, the "outflow" relationship between "OResource" and "DecrementEvent" is represented in the second line. (The resource is prefixed with an "O" because it is connected to an outflow relationship.) The "exchange" relationship shows duality between "IncrementEvent" and "DecrementEvent." Thus, the exchange relationship is the union of two simpler event relationships as represented by the plus (+) sign. The "from" and "to" relationships are also unions as both involve a simpler relationship between two events (Incrementevent and Decrementevent) and two agents (IAgent and OAgent).

The multiplicity specifications (in Figure 1 and other figures) are mainly based on translating from the multiplicities of UML classes provided by Hruby et al. [HKS06] into Alloy.

```
                    Pattern 1
inflow: IResource one -> some IncrementEvent
outflow: OResource one -> some DecrementEvent
from: (IncrementEvent set -> one IAgent) + (DecrementEvent set -> one OAgent)
to: (IncrementEvent set -> one OAgent) + (DecrementEvent set -> one IAgent)
exchange: (IncrementEvent some -> some DecrementEvent) +
          (DecrementEvent some -> some IncrementEvent)
```

Figure 1: A Formal Specification of a General Exchange Pattern

REA Exchange, Conversion, and Value Chain Patterns: The exchange pattern, previously presented in Figure 1, is assigned number 1. The conversion pattern is very similar to the exchange pattern; unlike an exchange pattern that models the exchanges of resources by agents, a conversion pattern models the creation or maintenance of resources by agents. Figure 2 shows the conversion pattern (numbered 2).

Figure 3 is the REA value chain pattern (numbered 3) connecting exchanges and conversions. The union operator (+) describes the combinations (e.g., Exchange + Conversion), and the or operator (||) shows possible combinations.

REA Type and Group Patterns: Typification and grouping apply to Resource, Event, Agent, Commitment,

```
                    Pattern 2
create: IResource one -> some IncrementEvent
(use: OResource one -> some DecrementEvent) ||
(consume: OResource one -> some DecrementEvent)
from:(IncrementEvent set -> one IAgent) + (DecrementEvent set -> one OAgent)
to:(IncrementEvent set -> one OAgent) + (DecrementEvent set -> one IAgent)
exchange: (IncrementEvent some -> some DecrementEvent) +
          (DecrementEvent some -> some IncrementEvent)
```

Figure 2: REA Conversion Pattern

```
                    Pattern 3
(Exchange + Exchange) || (Exchange + Conversion) || (Conversion + Conversion)
```

Figure 3: REA Value Chain Pattern

and Contract entities. Typification is "a-kind-of" relationship, whereas grouping is "a member of" relationship. Furthermore, typification is transitive, but grouping is not. Despite these differences, the distinction between typification and grouping are not always definite [GM06]. In Figure 4, the set multiplicities next to Resource, Agent, Event, Commitment, and Contract indicate that type and group apply to a number of instances and members. An exclusive classification for the type pattern (e.g., a resource can be of one type – the multiplicity of one) is assumed; for a shared classification, the multiplicity of one changes to set. Conversely, a shared classification for the group pattern (i.e., the multiplicity of set on the left side of the group pattern) is specified; for an exclusive classification, the multiplicity of set on the left of pattern five changes to the multiplicity of one. Figure 4 shows the specification of REA type pattern (numbered 4) and REA group pattern (numbered 5).

```
                    Pattern 4
Pattern 4a      specify: ResourceType one -> set Resource
Pattern 4b      specify: AgentType one -> set Agent
Pattern 4c      specify: EventType one -> set Event
Pattern 4d      specify: CommitmentType one -> set Commitment
Pattern 4e      specify: ContractType one -> set Contract
  ------------------------------------------------------------

                    Pattern 5
Pattern 5a      grouping: ResourceGroup set -> set Resource
Pattern 5b      grouping: AgentGroup set -> set Agent
Pattern 5c      grouping: EventGroup set -> set Event
Pattern 5d      grouping: CommitmentGroup set -> set Commitment
Pattern 5e      grouping: ContractType set -> set Contract
```

Figure 4: REA Type and Group Patterns

REA Commitment and Contract Patterns: Commitment and contract patterns (numbered 7 and 8) are two additional REA patterns, shown in Figure 5.

```
                        Pattern 6 and 7
duality: (EconomicIncrementEvent some -> some EconomicDecrementEvent) +
         (EconomicDecrementEvent some -> some EconomicIncrementEvent)
reciprocal: (IncrementCommitment some -> some DecrementCommitment) +
            (DecrementCommitment some -> some IncrementCommitment)
fulfillment: Commitment set -> one EconomicEvent
reservation: Commitment set -> one Resource
intendedProvide: (DecrementCommitment set -> one OAgent ) +
                 (IncrementCommitment set -> one IAgent)
intendedReceive: (IncrementCommitment set -> one OAgent) +
                 (DecrementCommitment set -> one IAgent)
clause: (IncrementCommitment some -> one Contract) +
        (DecrementCommitment some -> one Contract)
```

Figure 5: REA Commitment and Contract Patterns

REA Schedule and Policy Patterns: Two other REA patterns are schedule (numbered 8) and policy (numbered 9).

REA Linkage, Association, and Custody: These REA patterns are simpler than others and can be viewed as relationships. The linkage relationship (numbered 10) shows the structure and dependency of a resource (i.e., the composition of a resource as many resources consist of parts). The rationale for having this relationship is to avoid having an entity of *part* because a *part* is also a resource [HKS06]. The association relationship (numbered 11) can be any of the following three relationships: 1. the responsibility relationship (i.e., the dependency between two internal agents); 2. the cooperation relationship (i.e., the relationship between two external agents); or 3. the assignment relationship (the dependency between internal and external agents) [GM00]. Association and custody (numbered 12) are the relationships that are independent of agent participation in events [DCH05]. Figure 6 presents linkage, association, and custody relationships.

2.2 Placing REA Patterns in the Framework

Table 2 shows the Zachman framework in which the design row is populated with previously assigned REA pattern numbers. Our approach to the classification of REA patterns differs from previous approaches because our organization is closely related to formal pattern specifications. Note that the first column contains only things (resources), a restricted version of data. Furthermore, the custody pattern (pattern 12) is not placed in this table because it does not fit in one column and spans two columns: agents and resources.

```
Pattern 10a    linkage: ResourceComponent set -> set ResourceParent
Pattern 10b    linkageType: ResourceGroupComponent set -> set ResourceGroupParent
--
Pattern 11  association
Pattern 11a     responsibility: AgentInternalSup set -> set gentInternalSub
Pattern 11b     cooperation: AgentExternal set -> set ExternalAgent
Pattern 11c     assignment: AgentInternal set -> set AgentExternal
--
Pattern 12      custody: Agent set -> set Resource
```

Figure 6: REA Linkage, Association, and Custody Patterns (Relationships)

	Data	Function	Network	People	Time	Motivation
Planner						
Owner						
Designer	4a,5a,10a,10b	1,2,3		4b,5b,11	4c,5c,8	4d,5d,4e,5e,6,7,9
Builder						
Subcontractor						

Table 2: A Sketch of the Zachman Framework with REA Patterns

3 Related Work

(REA) Patterns and Classifications: The Resource-Event-Agent (REA) model [McC82] was first presented as REA (enterprise) domain ontology in *The ontological Foundation of REA Enterprise Information systems* [GM00]. Geerts and McCarthy [GM02], based on Sowa's work, classified REA ontology into three categories of Firstness (Independent), Secondness (Relative), and Thirdness (Mediating). Hruby et al. [HKS06] presented REA patterns in two levels: the operational level consisted of exchange, conversion, and value chain patterns; the policy level comprised all other REA patterns. Geerts, McCarthy, and Rockwell [GMR96] referred to five different views (i.e., planner, owner, designer, builder, and subcontractor) of the Zachman framework and described how the accounting database design could be improved. They considered rows one to four of the data column of this framework roughly equivalent to requirement analysis, conceptual design, logical design, and physical design of databases, respectively.

Patterns have been classified several times, as early as the mid 1990s, if not earlier. Gamma et al. [GHJV95] classified their design patterns into behavioural, creational, and structural patterns. Zimmer [Zim95] suggested three categories: "uses," "is similar to," or "can be combined with" for the classification of Gamma et al.'s design patterns. Trowbridge and his colleagues at Microsoft [TCE+04] used a tabular classification, a modifi-

cation of the Zachman framework, to organize their patterns. They allocated one hundred and five patterns to the design and builder rows of the framework. Hafiz et al. [HAJ07] also classified security patterns.

4 Conclusion

We realized that the fundamental building blocks of REA (Resource, Event, Agent) match the columns of the Zachman framework. In addition, some authors (e.g., [HDC99]) added *location* to REA and obtained an acronym of REAL. (In this case, *location* matches the third column of this framework.) However, there are several questions that need to be answered: What should we do with patterns that span a few columns (i.e., the custody pattern)? Is multiple column spanning acceptable and even expected when more patterns are added? Do commitment and contract patterns belong to column six (as policy does) or to column two as the extensions to exchange and conversion patterns?

We showed an approach to classify REA patterns using their formal specifications in the Zachman framework but have not described how these patterns are related to each other. We plan to extend this classification by showing the possible combinations of REA patterns. Another possible extension is to describe REA patterns with regard to the builder view (row 4) of the Zachman framework.

References

[Bud03] David Budgen. *Software design*. Pearson/Addison-Wesley, 2 edition, 2003.

[DCH05] Cheryl Dunn, J. Owen Cherrington, and Anita Hollander. *Enterprise Information Systems: A Pattern-Based Approach*. McGraw-Hill Irwin, 3 edition, 2005.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley, 1995.

[GM00] Guido Geerts and William McCarthy. The ontological foundation of REA enterprise information systems. Paper presented to the American Accounting Association Conference, Philadelphia, 2000.

[GM02] Guido Geerts and William McCarthy. An ontological analysis of the economic primitives of the extended-REA enterprise information architecture. *The International Journal of Accounting Information Systems*, 3(1):1–16, March 2002.

[GM06] Guido Geerts and William McCarthy. Policy-level specifications in REA enterprise information systems. *Journal of Information Systems*, 20(2):37–63, 2006.

[GMR96] Guido Geerts, William McCarthy, and Stephen Rockwell. Automated integration of enterprise accounting models throughout the systems development life cycle. *Intelligent Systems in Accounting, Finance, and Management*, 5(3):113–128, September 1996.

[HAJ07] Munawar Hafiz, Paul Adamczyk, and Ralph Johnson. Organizing security patterns. *IEEE Software*, 24(4):52, 2007.

[HDC99] Anita Hollander, Eric Denna, and J. Owen Cherrington. *Accounting, Information Technology, and Business Solutions*. McGraw-Hill/Irwin, 2 edition, 1999.

[HKS06] Pavel Hruby, Jesper Kiehn, and Christian Vibe Scheller. *Model-Driven Design Using Business Patterns*. Springer, 2006.

[Jac] Daniel Jackson. http://alloy.mit.edu.

[Jac00] Daniel Jackson. Automating first-order relational logic. In *ACM SIGSOFT Symposium on Foundations of Software Engineering*, November 6-10, 2000.

[Jac06] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.

[McC82] William McCarthy. The REA accounting model: A generalized framework for accounting systems in a shared data environment. *The Accounting Review*, 57(3):54–78, July 1982.

[SZ92] John Sowa and John Zachman. Extending and formalizing the framework for information systems architecture. *IBM Systems Journal*, 31(3):590–616, 1992.

[TCE+04] David Trowbridge, Ward Cunningham, Matt Evans, Larry Brader, and Paul Slater. Describing the enterprise architectural space. Technical report, MicroSoft Corporation, June 2004.

[Zac87] John Zachman. A framework for information systems architecture. *IBM Systems Journal*, 26(3):276–292, 1987. There is a reprint of this article in IBM Systems Journal (vol.38 no 2/3) in 1999.

[Zim95] Walter Zimmer. Relationships between design patterns. In James Coplien and Douglas Schmidt, editors, *Pattern Languages of Program Design*, 1995.