*JOURNAL OF EMERGING TECHNOLOGIES IN ACCOUNTING*
Vol. 4
2007
pp. 161–182

# The Timeless Way of Building REA Enterprise Systems

**Guido L. Geerts**
*University of Delaware*

**Harry Jiannan Wang**
*University of Delaware*

**ABSTRACT:** In a continuously changing business environment, the need for enterprise systems that are more adaptable has been recognized by many. Several solutions are being suggested to improve the adaptability of enterprise systems, including service-oriented architectures, model-driven architectures, and reflective architectures. In this paper, we propose a timeless way of building enterprise systems that employs a reflective architecture with integrated Resource-Event-Agent (REA) enterprise ontology specifications. We show how the explicit recording of enterprise schema descriptions results in enterprise systems with increased adaptability. In addition, we demonstrate how explicitly recorded ontological specifications can further increase application reusability. We validate our research with a prototype system.

**Keywords:** adaptability; enterprise systems; REA enterprise ontology; reflective system architectures; reusability.

## INTRODUCTION

In a continuously changing business environment, the need for enterprise systems that are more adaptable has been recognized by many. Several solutions are being suggested to improve the adaptability of software systems. For example, service-oriented architectures enable organizations to transform their business processes into a set of automated business services with standard interfaces, resulting in more adaptive enterprise systems (Zhao et al. 2007). Model-driven architectures define an enterprise model at a high level of abstraction, and this is then transformed into models at progressively lower levels of abstractions until source code is generated (Frankel 2003; Kleppe et al. 2003). Changes in business practices are accommodated by enterprise model updates that are then automatically propagated into software code. Reflective systems record enterprise schema descriptions explicitly so that they can be manipulated at run time.

This paper explores the design of reflective systems where the explicitly recorded enterprise schema descriptions are further defined in terms of a domain ontology. The ontology chosen is the Resource-Event-Agent Enterprise Ontology or REA-EO (McCarthy 1982;

We acknowledge the helpful comments of Bob Haugen, Pavel Hruby, Mette Jaquet, Jesper Kiehn, Wim Laurier, the editor, three anonymous reviewers, and the participants of the OOPSLA'97 Business Object Workshop and the Fifteenth Annual Research Workshop on AI/ET in Accounting, Auditing, and Tax.

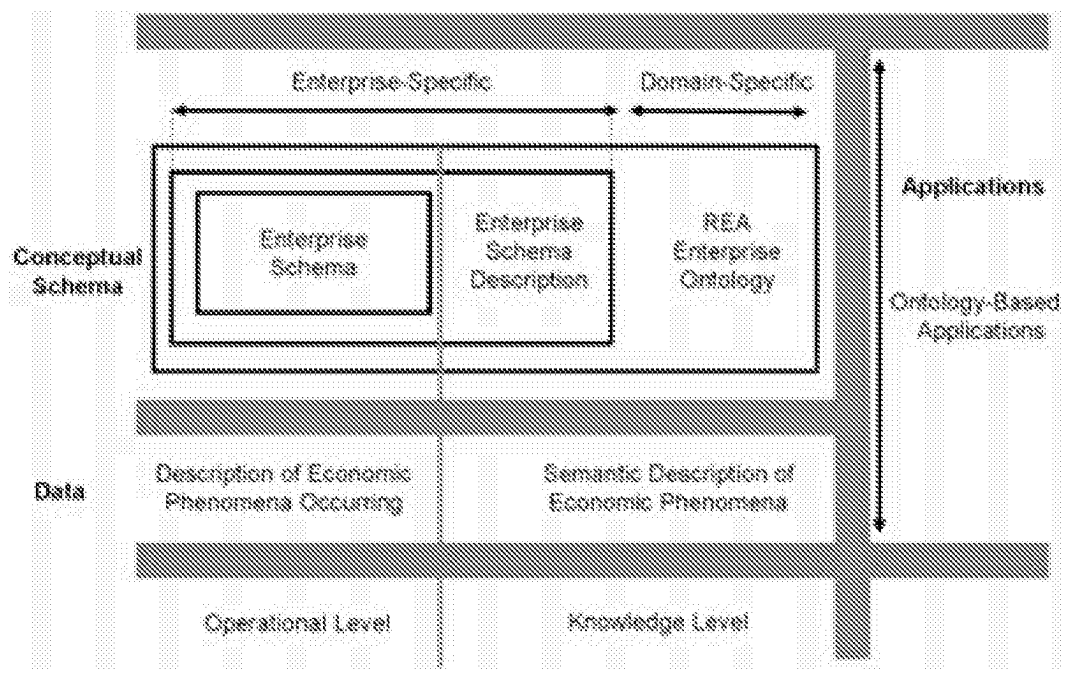
Corresponding author: Guido L. Geerts
Email: geertsg@lerner.udel.edu

Geerts and McCarthy 2004). The REA-EO is strongly rooted in accounting and economic theory and addresses the issue of what phenomena should be captured in an enterprise system. In addition, it provides structuring guidelines about the way economic phenomena should be assembled into business process and value chain specifications (Geerts and McCarthy 2001). The explicit recording of the semantics provided by the REA-EO enables increased reusability at the application level.

The architecture of a timeless REA enterprise system is illustrated in Exhibit 1. We follow Fowler (1997) in using the terms operational and knowledge level. An enterprise schema (operational level) describes the actual economic phenomena occurring in a company, such as information regarding a specific customer. Knowledge-level specifications, on the other hand, describe the economic phenomena represented in the enterprise system. For example, a definition of the characteristics of customers would represent metadata. Data dictionaries are primitive examples of metadata descriptions. The architecture in Exhibit 1 further differentiates between "enterprise schema descriptions" and "ontological specifications." Enterprise schema descriptions define the specific phenomena that occur in a company, such as the existence of customers and the characteristics to be communicated regarding customers. The ontological specifications, on the other hand, provide domain-specific definitions that are more general in nature. For example, customers would be classified as being of the REA-EO type agent. Applications have access to both operational- and knowledge-level specifications.

The objective of this paper is to explore how the architecture in Exhibit 1 can result in systems that are more adaptable and how it enables the engineering of applications that

**EXHIBIT 1**
**Architecture for Timeless REA Enterprise Systems**



are more reusable. In short, two main research issues addressed in this paper are: (1) building blocks and core design principles for timeless REA enterprise systems, and (2) implementation of a proof-of-concept prototype system that demonstrates how timeless REA enterprise systems improve adaptability and application reusability. The remainder of the paper is organized as follows. The second section discusses the main building blocks of timeless REA enterprise systems: reflective architectures and the REA enterprise ontology. The core principles of designing timeless REA enterprise systems are discussed in the third section. The fourth section presents a prototype implementation that illustrates the adaptability and reusability of timeless REA enterprise systems. We start with showing how to define an enterprise schema and its ontological specifications as part of a reflective architecture; next, we design two reusable applications that are expressed in terms of the REA-EO; finally, we show how a change in business practices is accommodated by the prototype implementation. We end the paper with some conclusions and future research directions.

## BUILDING BLOCKS

### Reflective Architectures

A reflective architecture is a means to implement enterprise systems that can be adapted to new user requirements at run time by retrieving and interpreting descriptive information. Yoder and Johnson (2002) present the adaptive object model (AOM) architecture as a particular approach to reflective architectures. An AOM represents classes, attributes, associations, and behavior as metadata which are stored in a database and then interpreted at runtime. Consequently, the descriptive information can be modified by nonprogrammer users to cope with new requirements.

Improved adaptability is the main promise of systems built with a reflective architecture. However, it should be noted that reflective architectures are more complex. The complexity results from semantics that are spread across metadata and that are harder to understand. Studies that aim at a better understanding and representation of metadata include the work on analysis patterns done by Hay (1996) and Fowler (1997). Further, a decrease in performance results from the fact that metadata need to be interpreted at run time, which typically includes access to the database system that stores them. Some research has been undertaken that focuses on improving the performance of reflective systems by leveraging different technologies, including caching, composite queries, and incremental computation (Nakamura and Johnson 1998; Yoder and Razavi 2000; Yoder and Johnson 2002).

Research issues needing further exploration comprise the nature of the semantics to be recorded as part of a reflective architecture including integrated ontological specifications and using such enhanced semantics as part of enterprise applications. These issues are the focus of this paper with the REA-EO being used as domain ontology.

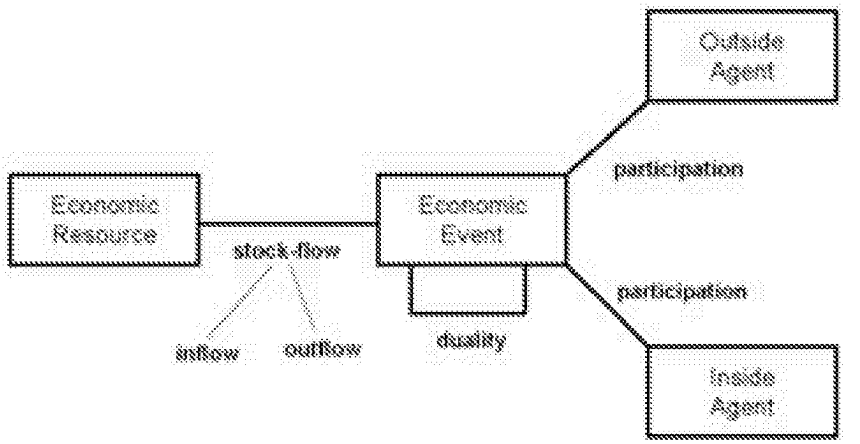
### The REA Enterprise Ontology

Domain ontologies define the things that are relevant to a specific application domain. The REA-EO defines the concepts that are relevant for enterprise systems and the relationships between them. The concepts and relationships defined in an ontology are its primitives. Exhibit 2[1] shows the primitives defined by the REA-EO (McCarthy 1982).

The three core concepts of the REA-EO are resources, events, and agents. Resources represent objects that are scarce, have utility, and are under the control of an enterprise (Ijiri 1975). Events represent a class of phenomena that reflect changes in scarce means

[1] We use the Unified Modeling Language (UML) (Booch et al. 1999) as notation in this paper.

**EXHIBIT 2**
**REA Enterprise Ontology (Adapted from McCarthy 1982)**



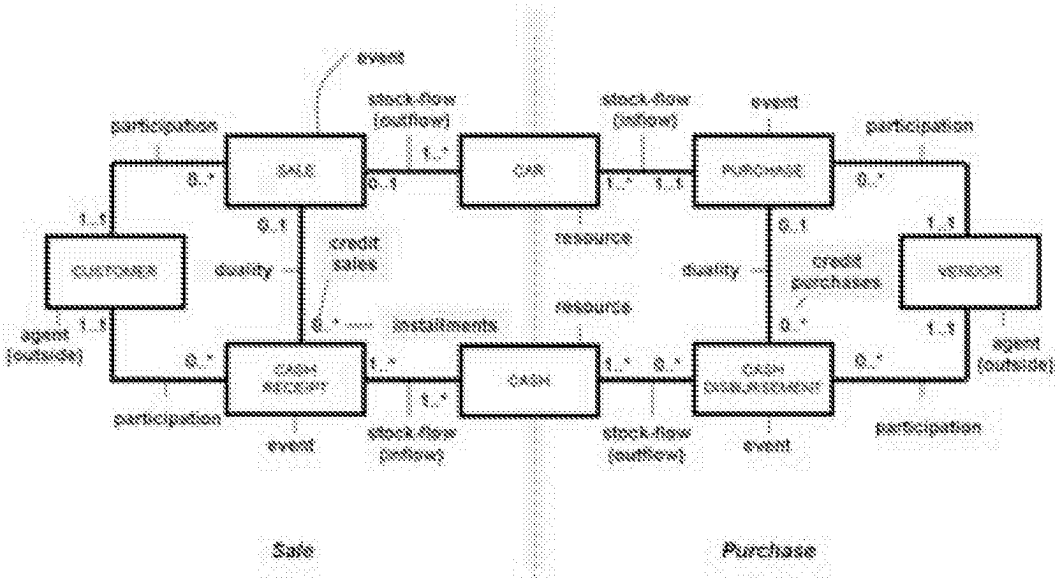

(economic resources) resulting from production, exchange, consumption, and distribution. Agents represent persons and agencies who participate in the economic events of the enterprise. The REA-EO further distinguishes between outside and inside agents. Outside agents are economic agents outside the enterprise who participate in the enterprise's economic events. Inside agents are economic agents who are accountable for the economic events that take place. For example, a customer is an outside agent for both sale and cash receipt, while a salesperson is an inside agent for sale and a cashier is an inside agent for cash receipt.

In essence, the REA-EO template in Exhibit 2 is a generic description of an economic event. The REA-EO recognizes that an economic event should participate in three different associations: duality, stock-flow, and participation. First, a duality association defines the association between economic events where the increment economic events are paired with decrement economic events. For example, sales (decrement of resources) are paired with cash receipts (increment of resources). Second, a stock-flow association describes the relationship between an event and a resource. An explicit distinction is made between the inflow of resources and the outflow of resources. For example, a sale results in an outflow of products, and a cash receipt results in an inflow of cash. Sale and cash receipt are economic events while product and cash are economic resources. Third, a participation association describes an event's association with outside and inside agents. Duality, stock-flow (inflow and outflow), and participation are REA-EO's association primitives.

Exhibit 3 shows an REA enterprise schema that defines the economic activities for a car dealer[2] that we will use as an example throughout the rest of this paper. The left side of Exhibit 3 shows the car dealer's sale business process: the exchange of cars for cash with customers. The right side shows the car dealer's purchase business process: the exchange of cars for cash with vendors. In addition, multiplicities are used to define business rules that express the car dealer's policies. Connolly and Begg (2005) define multiplicity as follows: "The number (or range) of possible occurrences of an entity type that

[2] We left inside agents out for simplicity purposes.

**EXHIBIT 3**
**REA Enterprise Schema: Economic Activities of a Car Dealer**
**No Trade-Ins**

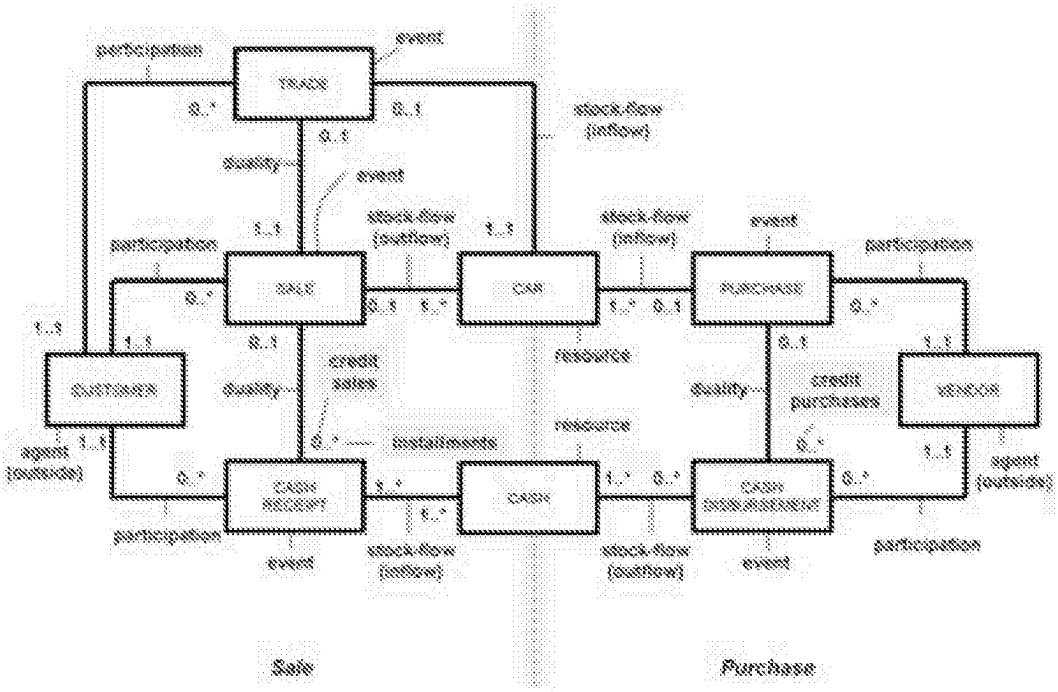



may relate to a single occurrence of an associated entity type through a particular relationship." For example, the multiplicities for the duality relationship between sale and cash receipt define a policy stating that credit sales and installments are acceptable while the multiplicities for the duality relationship between purchase and cash disbursement define a policy stating that credit purchases are acceptable.

Enterprise schemas such as the one in Exhibit 3 have a dual purpose. First, they result in a better understanding of the meaning of data and, for enterprise systems, in a better understanding of the economic activities of a company. The REA-EO further improves such understanding by providing guidance in the form of its structuring rules: the way the events are described and aggregated into business processes and the way the business processes are aggregated into value chains (Geerts and McCarthy 2001). Second, the enterprise schemas are mapped into design structures that can then be implemented with technologies such as relational databases, object-oriented platforms, and XML.

The enterprise schema in Exhibit 3 represents business practices as they occur at a specific point in time. In today's highly dynamic business environment, the economic activities of a company and its business policies change frequently in order to cope with new customer demands and market opportunities. For example, the car dealer whose economic activities are represented in Exhibit 3 does not allow trade-ins. Suppose that the car dealer changes its business policies and wants to accept trade-ins to increase sales. The existing enterprise schema must be adapted to the new practice resulting in the enterprise model shown in Exhibit 4. As a result, the underlying enterprise system must also be recompiled and redeployed, which is often costly and inefficient. Next, we explore how timeless REA

**EXHIBIT 4**
**REA Enterprise Schema: Economic Activities of a Car Dealer**
**Trade-Ins**




enterprise systems are able to absorb such changes and make enterprise systems more adaptive.

## CORE DESIGN PRINCIPLES

Information systems have a data component or information base and a data structure or conceptual schema that defines the semantics of the data and the constraints that apply to them (ISO 1982). Data structures are typically hardwired into the information system, and changes to them are costly and time-consuming, often requiring the creation or modification of table or object class definitions and additional programming. On the other hand, data manipulation such as adding a new customer is easy to do and the cost is minimal. The objective of timeless enterprise systems is to enable the accommodation of normal changes in the economic activities and business rules of an organization without affecting the underlying data structure and programs. As discussed in more detail below, this can be accomplished by defining semantics and business rules as part of the data component and thus allows users, especially nonprogrammers, to change them at runtime.

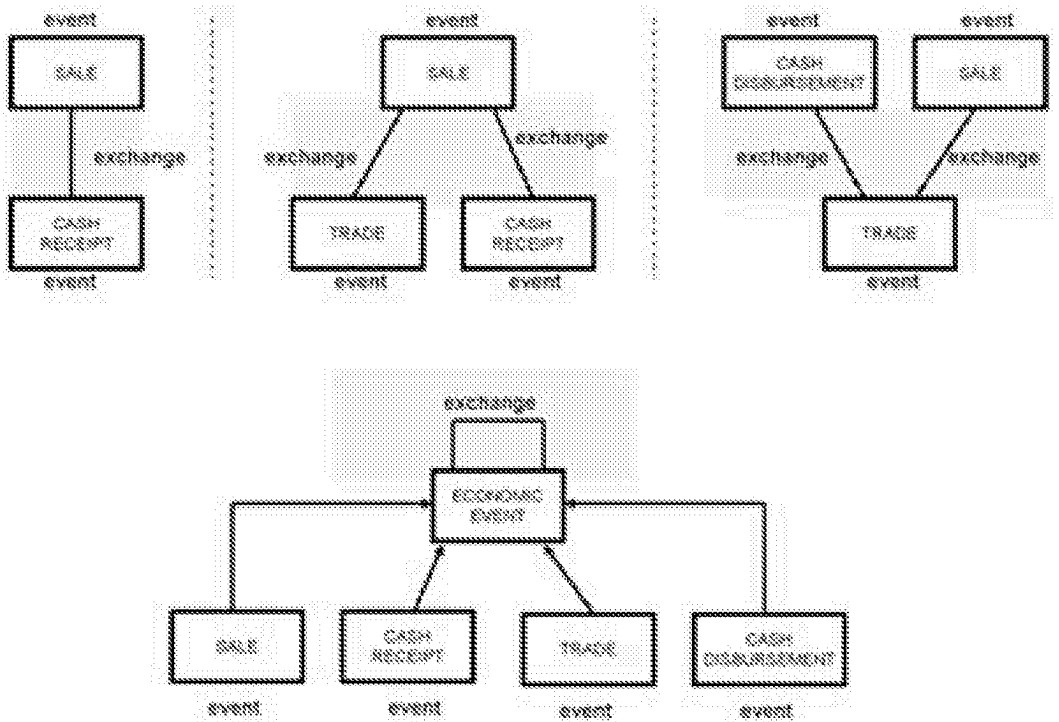
The conceptual schema in Exhibit 3 defines the following semantics and business rules. First, object classes define what economic phenomena can be part of the enterprise system. Information about sales, but not information about trade-ins or leases, can be recorded in the enterprise system. Second, associations define how concepts can interact. For example, vendors can be linked with purchases and cash disbursements but not with sales. Third,

while not shown in Exhibit 3 for simplicity purposes, properties define what data elements can be used for communication about the object classes. For example, communication regarding sales can be limited to the following data elements: ID, amount, and date. Fourth, business rules (multiplicities) define additional constraints. For example, the business rules in Exhibit 3 express the company's policy of accepting credit sales, installments, and credit purchases.

The definitions of the object classes, associations, properties, and constraints are an integral part of the data structure. In this paper, we define an enterprise schema as the part of the data structure (or conceptual schema) that represents the economic activities actually taking place in a business organization. The following statement holds when all semantics are defined as part of the enterprise schema, as is the case for Exhibit 3: "data structure = enterprise schema." Changes in the company's business practices, e.g., a new economic activity such as trade-ins or a new business rule such as requiring down payments by customers, would require changes in the data structure and would thus result in costly implementation work.

A continuously changing business environment requires ongoing changes to object classes, associations, properties, and business rules. The upper part of Exhibit 5 shows some alternative revenue generating practices for a car dealer. A car dealer can sell for cash only, can consider trade-ins, and can consider trade-ins where the value of the traded car is higher than the value of the new car. Other practices that could be considered but

**EXHIBIT 5**
**A Timeless Enterprise Model for Economic Exchanges**

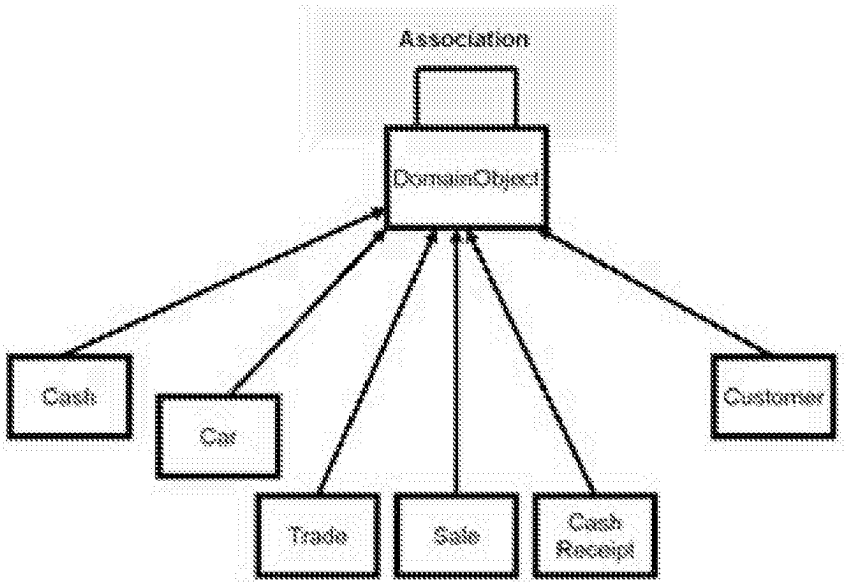


are not shown in Exhibit 5 are leasing cars and renting cars. Similar to Exhibits 3 and 4, the models in the upper part of Exhibit 5 hardwire specific business practices into their data structure, and any changes would result in major modifications of the enterprise system.

Timeless enterprise modeling starts with analyzing the core concepts underlying the economic activities. For example, there are commonalities among all associations in the upper part of Exhibit 5: they all represent specific configurations of the more generic "exchange"[3] concept. The lower part of Exhibit 5 shows an enterprise schema that replaces the specific exchanges in the upper part of Exhibit 5 by a generic "exchange" association. As a result, the user can record any current or future exchange between two economic events without making any change to the data structure. However, the increased flexibility comes at a price: reduced semantics. For the model in the lower part of Exhibit 5, exchanges can be defined between any two events; for example, a trade could be linked with a cash receipt or a sale with a purchase. Further, the business rules explicitly defined in the models in Exhibits 3 and 4—such as credit sales, installments, and credit purchases—are no longer defined in the enterprise schema in the lower part of Exhibit 5.

As illustrated by the diagram in Exhibit 6, further abstraction results in a further increase of flexibility and a further decrease of semantics. Exhibit 6 is an alternative representation for the sales business process for a car dealer and thus the left side of Exhibits 3 and 4. All object classes are now modeled as subtypes of DomainObject and all associations are recorded as Association instances. The domain classes in Exhibit 6 (Cash, Car, Trade, Sale, CashReceipt, Customer) are modeled exactly the same way as in Exhibit 4. However, domain-specific associations such as Sale-Customer and their business rules are no longer

**EXHIBIT 6**
**A Timeless Enterprise Model for Associations**



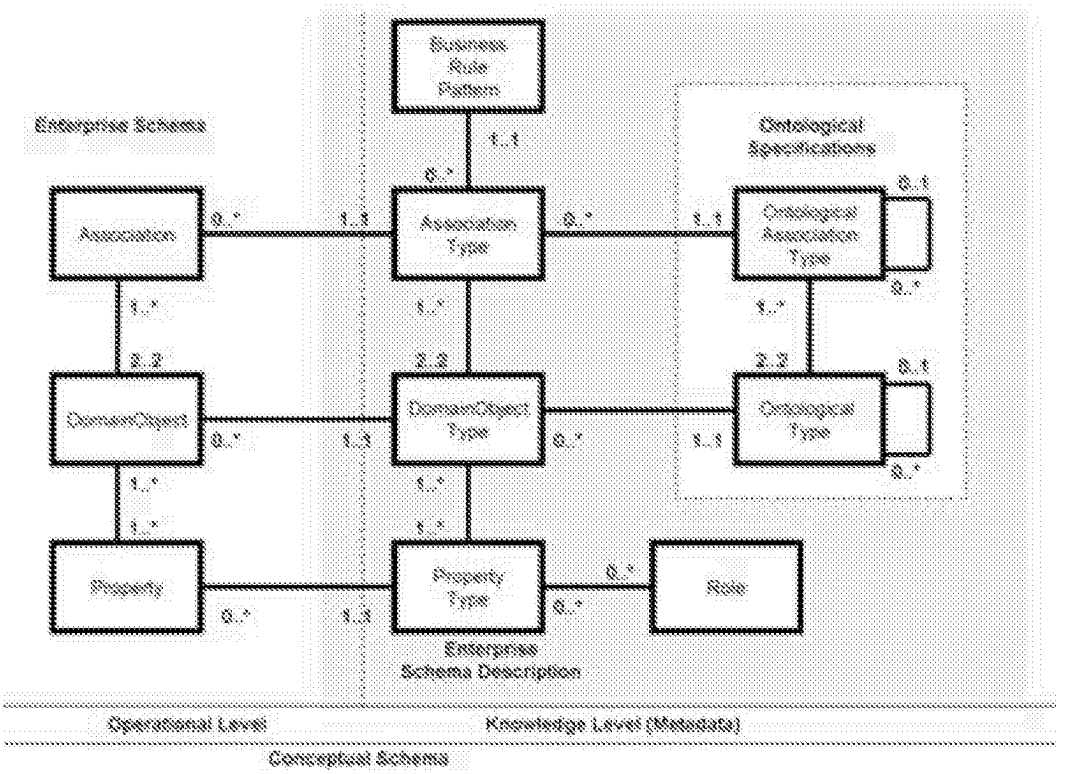

[3] Geerts and McCarthy (2004) declare exchanges and transformations as subtypes of the duality primitive. We ignore this distinction here and use the terms duality and exchange interchangeable.

explicitly defined as part of the enterprise schema. The flexibility is increased since any association between two domain objects can now be recorded in the information base. For example, there would be no problem extending the car dealer's business operations with leases and recording associations between a lease and a customer or a lease and a cash receipt.

The increase in flexibility further reduces the semantics. For the model in Exhibit 6, there are no restrictions on associations between objects; e.g., a Sale instance could be connected with a Vendor instance. Further, the multiplicities and thus the business rules defined for the associations in Exhibits 3 and 4 are no longer expressed in Exhibit 6. The conceptual schemas in Exhibits 5 and 6 still record the actual business activities of an enterprise and the "data structure = enterprise schema" equation still holds.

The examples in Exhibits 5 and 6 illustrate the trade-off between flexibility and semantics. The ideal situation would be where the specification of the semantics is preserved or even extended while the increased flexibility is maintained. One approach is the use of a reflective architecture where the descriptions of the enterprise schema are integrated as part of the conceptual schema. Exhibit 7 illustrates such an extended conceptual schema where "data structure = enterprise schema + enterprise schema description." The enterprise schema description (middle part of Exhibit 7) does not define domain semantics but provides a framework to describe the semantics of the enterprise schema as part of the

**EXHIBIT 7**
**Conceptual Schema for Timeless REA Enterprise Systems**



information base. Users define an enterprise's current configuration of economic activities and business rules. Instances of DomainObjectType define the economic phenomena that are currently part of the enterprise schema: Cash, Car, Sale, CashReceipt, Customer, and Trade. Instances of AssociationType define what associations can exist between the domain objects types (Sale-Customer, Sale-CashReceipt, CashReceipt-Customer, Sale-Car, etc.) and replace the associations hardwired as part of the data structures in Exhibits 3 and 4. Instances of PropertyType define what properties domain object types can have.

The following could be properties for the Sale object type: ID, amount, and date. A Role instance defines the role a property plays in a specific application. For example, by declaring date as having role DOO (Date-Of-Occurrence), applications would know that the date property defines when an event has occurred. Instances of BusinessRulePattern represent multiplicity patterns that define the business rules that apply to an association. For simplicity purposes, we limit the lower multiplicity values to 0 and 1 and the upper multiplicity values to 1 and * resulting in 16 business rule patterns. Being recorded in the information base, the semantic descriptions can be updated easily at run time; i.e., no changes to the data structure are required.

Exhibit 7 further extends the conceptual schema with ontological specifications resulting in the following equation: "data structure = enterprise schema + enterprise schema definitions + ontological specifications." Again, the conceptual schema extension at the right side of Exhibit 7 does not define the REA-EO but provides a framework to describe the semantics of the REA-EO or any other ontology as part of the information base. Instances of OntologicalType define the REA-EO's concept primitives: resource, event, and agent. The recursive association is needed to define that inside and outside agents are specializations of agent. Instances of OntologicalAssociationType define the REA-EO's association primitives: stock-flow, duality, and participation. The recursive association is needed to define that inflow and outflow are specializations of stock-flow.

Extending the conceptual schema with ontological specifications adds value in different ways. First, the architecture in Exhibit 7 supports continuously changing enterprise schema descriptions as defined by the user. The ontological specifications make sure that the enterprise schema descriptions adhere to REA-EO's structuring rules and best practices. Stated differently, the ontological specifications can be used for validation purposes (Geerts et al. 1996). Second, adherence to the REA-EO results in interoperability. Information systems grounded into the same ontology can be integrated more easily. Third, ontologies themselves are subject to changes, and such changes can be accommodated easily by the architecture in Exhibit 7. Also, an enterprise can adapt the ontological specifications to its own needs or the needs of its industry. Fourth, the explicit definition of the ontology provides us with additional knowledge that can be used as part of applications. Ontology-driven applications are discussed in the next section.

## A PROTOTYPE IMPLEMENTATION

As mentioned above, two issues with reflective architectures and their applications are increase in complexity and decrease in performance. For example, if we want a list of all sales, for the conceptual schema in Exhibit 3, we can easily get that list by finding all the instances of the Sale object class. However, this task becomes more complex when using an enterprise system that implements the conceptual schema in Exhibit 7. We first need to find all domain objects whose domain object type is Sale, then we need to use the information captured by the DomainObject, Property, and PropertyType object classes and the associations among them to determine the properties for each of the Sale instances. Similarly, when we need to generate a report that lists all payments for a specific sale using the

conceptual schema in Exhibit 3, the Sale–CashReceipt association makes that information readily available. On the other hand, for an enterprise system that implements the conceptual schema in Exhibit 7, we need to find all associations in which the sale participates and for which the AssociationType connects the Sale and CashReceipt domain object types. In other words, applications built with reflective systems need to access metadata at runtime and are more complex to write and need more processing resources and time.

At the same time, reflective enterprise systems provide substantial benefits over conventional enterprise systems in terms of adaptability. We have developed a prototype to illustrate the inner workings of a reflective enterprise system. The prototype's database that stores all data and metadata was implemented with MS SQL Server 2005 while MS Visual Basic 2005 was used for application development. Both technologies were chosen because they are widely available. The structure of the prototype's database is directly mapped from the conceptual schema in Exhibit 7 and shown in Exhibit 8. However, we made the following decisions when creating the database structure in Exhibit 8: (1) the recursive association for OntologicalType has not been implemented since it is not required by any of the applications, (2) binary associations represented by 2..2 multiplicities in Exhibit 7 have been implemented by two foreign keys, and (3) Exhibit 7 shows that a many-to-many relationship exists between PropertyType and Role, meaning that a property type can have multiple roles and the same role can be assigned to multiple property types. However, to simplify the implementation in Exhibit 8, we assume that a property type can only have one role.

The remainder of this section illustrates the inner workings of a reflective enterprise system in three steps. We first illustrate how enterprise schema semantics can be implemented with a reflective architecture. More specifically, we illustrate how the enterprise schema in Exhibit 3, i.e., the car dealer model without trade-ins, is implemented using the data structure shown in Exhibit 8. The data in Exhibit 9 represent the actual implementation of the enterprise schema in Exhibit 3. Second, we discuss the design of ontology-driven applications and their implementation with a reflective architecture. A key element of such applications is the explicit recording of ontological specifications as part of the metadata. We chose the following two applications for illustration purposes: (1) describe resource acquisitions, i.e., show how and when a resource has been acquired, and (2) claims, i.e., determine the existing exchange imbalances for an economic event. Third, we illustrate the adaptability of our prototype. More specifically, we extend the car dealer's business operations with trade-ins and show how we can adapt the prototype enterprise system to such a new business practice by manipulating the metadata in Exhibit 9.[4] The changes resulting from the company considering trade-ins are shown by the shaded areas in Exhibit 9. The data and metadata in the shaded areas should be ignored when we discuss the implementation of the enterprise model in Exhibit 3. Ontology-driven enterprise applications are highly reusable and are able to handle changes in business practices. We will discuss in more detail how the resource acquisition and claims applications deal with the trade-ins extension.

**Implementing Enterprise Schema Semantics with a Reflective Architecture**

As pointed out above, the conceptual schema in Exhibit 7 represents a reflective architecture that describes the semantics of an enterprise schema as part of the information base. Exhibit 9 shows how such a reflective architecture implements the enterprise schema in Exhibit 3. Following the distinction made in Exhibit 7, the implementation in Exhibit 9

[4] For simplicity, we ignore the modified multiplicities for the Car-Purchase association: not all cars are purchased.

## EXHIBIT 8
## Database Structure for Timeless REA Enterprise Systems

### Panel A: Enterprise Schema

**Association**

| A_ID | DO_ID1 | DO_ID2 | AT_ID |
|---|---|---|---|
| Association ID: Identifier for Associations | References (DO_ID) to the two Domain Objects connected by the association | | Defines the Association's Type (its ID) |

**DomainObject**

| DO_ID | DOT_ID |
|---|---|
| Domain Object ID: Identifier for Domain Objects | Defines the Domain Object's Type (its ID) |

**DomainObjectProperty**

| DO_ID | P_ID |
|---|---|
| Links a Domain Object (DO_ID) to a Property (P_ID) | |

**Property**

| P_ID | PT_ID | P_VALUE |
|---|---|---|
| Property ID: Identifier for Properties | Defines the Property's Type (its ID) | Defines the Property's value |

### Panel B: Enterprise Schema Description

**BusinessRulePattern**

| BRP_ID | M1 | M2 |
|---|---|---|
| Business Rule Pattern ID: Identifier for Business Rule Patterns | Permissible multiplicity patterns for binary Association Types | |

**AssociationType**

| AT_ID | DOT_ID1 | DOT_ID2 | OAT_ID | BRP_ID |
|---|---|---|---|---|
| Association Type ID: Identifier for Association Types | References (DOT_ID) to the two Domain Objects Types connected by the Association Type | | Defines the Association Type's Ontological Association Type (its ID) | Defines the Association Type's Business Rule Pattern (its ID) |

**DomainObjectType**

| DOT_ID | NAME | OT_ID |
|---|---|---|
| Domain Object Type ID: Identifier for Domain Object Types | The name of the Domain Object Type | Defines the Domain Object Type's Ontological Type (its ID) |

**DomainObjectTypePropertyType**

| DOT_ID | PT_ID |
|---|---|
| Links a Domain Object Type (DOT_ID) to a Property Type (PT_ID) | |

**PropertyType**

| PT_ID | NAME | TYPE | PT_ROLE |
|---|---|---|---|
| Property Type ID: Identifier for Property Types | The name of the Property Type | The data type of the Property Type | Defines the Property Type's Role |

### Panel C: Ontological Specifications

**OntologicalAssociationType**

| OAT_ID | OA_NAME | OT_ID1 | OT_ID2 | OAT |
|---|---|---|---|---|
| Ontological Association Type ID: Identifier for Ontological Association Types | The name of the Ontological Association Type | References (OT_ID) to the two Ontological Types connected by the Ontological Association Type | | Reference (ID) to the Ontological Association Type. Implements the recursive association. |

**OntologicalType**

| OT_ID | OT_NAME |
|---|---|
| Ontological Type ID: Identifier for Ontological Types | The name of the Ontological Type |

## EXHIBIT 9
## Data for the Prototype Implementation

### Panel A: Enterprise Schema

**DomainObjectProperty**

| DO_ID | P_ID |
|---|---|
| 1 | 1 |
| 1 | 15 |
| 1 | 36 |
| 2 | 2 |
| 2 | 16 |
| 2 | 37 |
| 3 | 3 |
| 3 | 17 |
| 3 | 38 |
| 4 | 4 |
| 4 | 18 |
| 5 | 5 |
| 5 | 19 |
| 6 | 6 |
| 6 | 20 |
| 7 | 7 |
| 7 | 21 |
| 8 | 8 |
| 8 | 22 |
| 9 | 9 |
| 9 | 11 |
| 9 | 23 |
| 10 | 10 |
| 10 | 12 |
| 10 | 24 |
| 11 | 13 |
| 11 | 14 |
| 11 | 25 |
| 12 | 26 |
| 12 | 27 |
| 12 | 39 |
| 13 | 28 |
| 13 | 29 |
| 13 | 54 |
| 14 | 30 |
| 14 | 31 |
| 15 | 32 |
| 15 | 33 |
| 15 | 40 |
| 16 | 34 |
| 16 | 35 |
| 16 | 41 |
| 17 | 42 |
| 17 | 43 |
| 17 | 44 |
| 18 | 45 |
| 18 | 46 |
| 18 | 47 |
| 19 | 48 |
| 19 | 49 |
| 19 | 50 |
| 20 | 51 |
| 20 | 52 |
| 20 | 53 |

**Property**

| P_ID | PT_ID | P_VALUE |
|---|---|---|
| 1 | 1 | 10000 |
| 2 | 1 | 20000 |
| 3 | 1 | 35000 |
| 4 | 5 | 5000 |
| 5 | 5 | 5000 |
| 6 | 5 | 7500 |
| 7 | 5 | 1250 |
| 8 | 5 | 35000 |
| 9 | 2 | Ford Focus |
| 10 | 2 | Acura MDX |
| 11 | 3 | 2002 |
| 12 | 3 | 2006 |
| 13 | 2 | Ford Fusion |
| 14 | 3 | 2005 |
| 15 | 4 | 101 |
| 16 | 4 | 102 |
| 17 | 4 | 103 |
| 18 | 4 | 101 |
| 19 | 4 | 102 |
| 20 | 4 | 103 |
| 21 | 4 | 104 |
| 22 | 4 | 105 |
| 23 | 4 | C01 |
| 24 | 4 | C02 |
| 25 | 4 | C03 |
| 26 | 1 | 25000 |
| 27 | 4 | 104 |
| 28 | 6 | 10000 |
| 29 | 4 | T01 |
| 30 | 4 | 101 |
| 31 | 8 | 30000 |
| 32 | 4 | P01 |
| 33 | 7 | 10000 |
| 34 | 4 | P02 |
| 35 | 7 | 20000 |
| 36 | 9 | 6/1/2007 |
| 37 | 9 | 6/2/2007 |
| 38 | 9 | 6/3/2007 |
| 39 | 9 | 6/4/2007 |
| 40 | 9 | 5/5/2007 |
| 41 | 9 | 5/6/2007 |
| 42 | 4 | C04 |
| 43 | 2 | Mustang |
| 44 | 3 | 2006 |
| 45 | 4 | C05 |
| | | Nissan |
| 46 | 2 | Altima |
| 47 | 3 | 2003 |
| 48 | 4 | P03 |
| 49 | 7 | 35000 |
| 50 | 9 | 5/7/2007 |
| 51 | 4 | P04 |
| 52 | 7 | 25000 |
| 53 | 9 | 5/8/2007 |
| 54 | 9 | 6/2/2007 |

**DomainObject**

| DO_ID | DOT_ID |
|---|---|
| 1 | 2 |
| 2 | 2 |
| 3 | 2 |
| 4 | 7 |
| 5 | 7 |
| 6 | 7 |
| 7 | 7 |
| 8 | 7 |
| 9 | 1 |
| 10 | 1 |
| 11 | 1 |
| 12 | 2 |
| 13 | 9 |
| 14 | 8 |
| 15 | 3 |
| 16 | 3 |
| 17 | 1 |
| 18 | 1 |
| 19 | 3 |
| 20 | 3 |

**Association**

| A_ID | DO_ID1 | DO_ID2 | AT_ID |
|---|---|---|---|
| 1 | 1 | 4 | 9 |
| 2 | 1 | 5 | 9 |
| 3 | 2 | 6 | 9 |
| 4 | 2 | 7 | 9 |
| 5 | 3 | 8 | 9 |
| 6 | 1 | 9 | 2 |
| 7 | 2 | 11 | 2 |
| 8 | 3 | 10 | 2 |
| 9 | 2 | 13 | 11 |
| 10 | 14 | 15 | 16 |
| 11 | 14 | 16 | 16 |
| 12 | 12 | 17 | 2 |
| 13 | 13 | 18 | 6 |
| 14 | 15 | 9 | 4 |
| 15 | 16 | 11 | 4 |
| 16 | 19 | 10 | 4 |
| 17 | 20 | 17 | 4 |

(*continued on next page*)

**EXHIBIT 9 (continued)**

**Panel B: Enterprise Schema Description**

**AssociationType**

| AT_ID | DOT_ID1 | DOT_ID2 | OAT_ID | BRP_ID |
|---|---|---|---|---|
| 1 | 1 | 2 | 5 | 13 |
| 2 | 2 | 1 | 5 | 4 |
| 3 | 1 | 3 | 4 | 14 |
| 4 | 3 | 1 | 4 | 8 |
| 5 | 1 | 9 | 4 | 5 |
| 6 | 9 | 1 | 4 | 2 |
| 7 | 2 | 5 | 2 | 10 |
| 8 | 5 | 2 | 2 | 7 |
| 9 | 2 | 7 | 1 | 3 |
| 10 | 7 | 2 | 1 | 9 |
| 11 | 2 | 9 | 1 | 5 |
| 12 | 9 | 2 | 1 | 2 |
| 13 | 3 | 6 | 2 | 10 |
| 14 | 6 | 3 | 2 | 7 |
| 15 | 3 | 8 | 1 | 3 |
| 16 | 8 | 3 | 1 | 9 |
| 17 | 4 | 7 | 4 | 16 |
| 18 | 7 | 4 | 4 | 16 |
| 19 | 4 | 8 | 5 | 15 |
| 20 | 8 | 4 | 5 | 12 |
| 21 | 5 | 7 | 2 | 7 |
| 22 | 7 | 5 | 2 | 10 |
| 23 | 5 | 9 | 2 | 7 |
| 24 | 9 | 5 | 2 | 10 |
| 25 | 6 | 8 | 2 | 7 |
| 26 | 8 | 6 | 2 | 10 |

**BusinessRulePattern**

| BRP_ID | M1 | M2 |
|---|---|---|
| 1 | 0..1 | 0..1 |
| 2 | 0..1 | 1..1 |
| 3 | 0..1 | 0..* |
| 4 | 0..1 | 1..* |
| 5 | 1..1 | 0..1 |
| 6 | 1..1 | 1..1 |
| 7 | 1..1 | 0..* |
| 8 | 1..1 | 1..* |
| 9 | 0..* | 0..1 |
| 10 | 0..* | 1..1 |
| 11 | 0..* | 0..* |
| 12 | 0..* | 1..* |
| 13 | 1..* | 0..1 |
| 14 | 1..* | 1..1 |
| 15 | 1..* | 0..* |
| 16 | 1..* | 1..* |

**DomainObjectTypePropertyType**

| DOT_ID | PT_ID |
|---|---|
| 1 | 2 |
| 1 | 3 |
| 1 | 4 |
| 2 | 1 |
| 2 | 4 |
| 2 | 9 |
| 3 | 7 |
| 3 | 4 |
| 3 | 9 |
| 7 | 5 |
| 7 | 4 |
| 8 | 8 |
| 8 | 4 |
| 9 | 6 |
| 9 | 4 |
| 9 | 9 |

**PropertyType**

| PT_ID | NAME | TYPE | PT_ROLE |
|---|---|---|---|
| 1 | S_Amount | int | Amount |
| 2 | Name | string | Name |
| 3 | Year | int | Year |
| 4 | ID | int | ID |
| 5 | CR_Amount | int | Amount |
| 6 | T_Amount | int | Amount |
| 7 | P_Amount | int | Amount |
| 8 | CD_Amount | int | Amount |
| 9 | Date | date | DOO |

**DomainObjectType**

| DOT_ID | NAME | OT_ID |
|---|---|---|
| 1 | Car | 1 |
| 2 | Sale | 2 |
| 3 | Purchase | 2 |
| 4 | Cash | 1 |
| 5 | Customer | 3 |
| 6 | Vendor | 3 |
| 7 | CashReceipt | 2 |
| 8 | CashDisbursement | 2 |
| 9 | Trade | 2 |

**Panel C: Ontological Specifications**

**OntologicalType**

| OT_ID | OT_NAME |
|---|---|
| 1 | Resource |
| 2 | Event |
| 3 | Agent |

**OntologicalAssociationType**

| OAT_ID | OA_NAME | OT_ID1 | OT_ID2 | OAT |
|---|---|---|---|---|
| 1 | Duality | 2 | 2 | |
| 2 | Participation | 3 | 2 | |
| 3 | Stock-Flow | 2 | 1 | |
| 4 | Inflow | 2 | 1 | 3 |
| 5 | Outflow | 2 | 1 | 3 |

contains three different types of information: enterprise schema (Panel A), enterprise schema description (Panel B), and ontological specifications (Panel C).

The enterprise schema in Exhibit 3 has eight enterprise-specific object types which are defined in the DomainObjectType table in Exhibit 9 (enterprise schema description): Car, Sale, Purchase, Cash, Customer, Vendor, CashReceipt, and CashDisbursement. Each domain object type is uniquely identified (DOT_ID), is given a name (NAME), and is classified in terms of the REA-EO (OT_ID). For example, the second row in the DomainObjectType

table defines an object type which is identified as "2" (DOT_ID), has name "Sale" (NAME), has ontological type "2" (OT_ID), and thus is an Event.

Each domain object type further has a number of property types. For example, Sale, a domain object type, has three property types: a property that identifies the sale (ID), a property that defines the sale's amount (S_Amount), and a property that defines when a sale occurred (Date). Information regarding property types is recorded in the PropertyType table (enterprise schema description). Row one (PT_ID = "1") defines the S_Amount property, row four (PT_ID = "4") defines the ID property, and row nine (PT_ID = "9") defines the Date property. It is worth noting that different object types may share the same property type. For example, the ID property type is used by both Car and Sale. Information regarding what property types are used to describe each of the domain object types is captured in the DomainObjectTypePropertyType table in Exhibit 9 (enterprise schema description). For example, rows 4, 5, and 6 in the DomainObjectTypePropertyType table define that the domain object type with ID 2 (DOT_ID = "2"), and thus Sale, has three property types: (1) the property type with PT_ID = "1" and thus S_Amount, (2) the property type with PT_ID = "4" and thus ID, and (3) the property type with PT_ID = "9" and thus Date.

As was illustrated in Geerts (1993) and Geerts (2004), the design of ontology-driven, reusable applications further requires the recognition of stereotypical roles to be assigned to property types. For example, the claims application needs to know which event property type contains the amount to be used for calculation purposes. In Exhibit 9, the Sale. S_Amount, CashReceipt.CR_Amount, Purchase.P_Amount, and CashDisbursement.CD_ Amount property types are declared as having the role "Amount" (PT_ROLE).

The enterprise schema in Exhibit 3 also has ten associations: Sale-Car, Purchase-Car, Sale-Customer, Sale-CashReceipt, Vendor-Purchase, Purchase-CashDisbursement, Cash-Receipt-Cash, Cash-CashDisbursement, CashReceipt-Customer, and CashDisbursement-Vendor. All ten associations are defined as instances of the AssociationType table (enterprise schema description) and the following information is defined for each of them: a unique identifier (AT_ID), the two domain object types that participate in the association type (DOT_ID1 and DOT_ID2), the ontological association type (OAT_ID), and the business rule pattern that applies to the association type (BRP_ID). For example, the first row in the AssociationType table in Exhibit 9 has "1" as identifier, defines an association between the Car (DOT_ID1 = "1") and Sale (DOT_ID2 = "2") domain object types, is of type outflow (OAT_ID = "5"), and has business rule pattern 13 (BPR_ID = "13") applying to it: (1..*) – (0..1). Applied to the Car-Sale association, the business rule pattern with ID 13 implies that not all cars have been sold, but a car can be sold only once while all sales include one or more cars. The BusinessRulePattern table (enterprise schema description) contains the sixteen business rule patterns that we recognized above.

The fact that each association type is defined twice in the AssociationType table is an implementation issue. For example, the first row in the AssociationType table defines an association between Car and Sale while the second row defines an association between Sale and Car. Although the information contained in one row can be deduced from the other row, the use of both rows makes the implementation easier.

According to the REA-EO (McCarthy 1982), there are three types of domain objects in enterprise systems: resource, event, and agent. The REA-EO further recognizes three types of domain associations in enterprise systems: duality, participation, and stock-flow. Stock-flow associations are further differentiated into inflow or outflow. Ontological specifications are explicitly recorded as part of the metadata in Exhibit 9 (ontological specifications). The definitions for resource, event, and agent are stored in the OntologicalType

table while the definitions for duality, participation, stock-flow, inflow, and outflow are stored in the OntologicalAssociationType table. Additional semantics are captured by the association between OntologicalType and OntologicalAssociationType: e.g., a duality association is defined between two economic events. This association is implemented by the OT_ID1 and OT_ID2 columns in the OntologicalAssociationType table. Such information can be used for validation purposes when defining an REA-structured enterprise schema (Geerts et al. 1996).

The remaining tables in Exhibit 9 define the actual data: i.e., the phenomena that actually occur in the enterprise (the enterprise schema). The semantics for the actual data are defined by the metadata, which are comprised of the enterprise schema description and the ontological specifications. For example, the first instance of the DomainObject table (DO_ID = "1") defines a sale (DOT_ID = "2"). The DomainObjectProperty table further defines that this sale has three properties: P_ID = "1," P_ID = "15," and P_ID = "36." The values for the P_ID column refer to the actual property description. For example, the property with P_ID = "1" is of type S_Amount (PT_ID = "1") and has value "10000." Further, the association table defines how the DomainObject instances are connected. For example, the first instance of DomainObject (the sale described above) participates twice in the Sale-CashReceipt association (AT_ID = "9") and once in the Sale-Car association (AT_ID = "2"). Consider the instance in the Association table with A_ID = "6." It defines an instance of the Sale-Car association (AT_ID = "2") between the sale represented as the domain object with DO_ID = "1" and the car represented as the domain object with DO_ID = "9."

It should be noted that the system portrayed in Exhibit 9 is not capable of dealing with more advanced semantic abstractions such as association classes. For example, the reflective architecture is not able to properly define the situation of a many-to-many duality association between Sale and CashReceipt with amount as association class property. These more complex scenarios are ignored in this paper for simplicity purposes.

**Design and Implementation of Ontology-Driven Enterprise Applications**

An enterprise system such as the one portrayed in Exhibit 9 integrates metadata definitions and enables the design of ontology-driven enterprise applications as will be illustrated with the following two examples: (1) determine resource acquisitions, and (2) determine existing exchange imbalances for economic events (claims).

The requirement for the first application is to determine how and when each car was acquired. The enterprise schema in Exhibit 3 shows that all cars are purchased. However, in a future scenario, other ways of acquiring the same resource might be considered. For example, the enterprise schema in Exhibit 4 shows that cars can either be purchased or traded. A timeless enterprise system needs to recognize current and future business practices. This can be accomplished by using the following ontology-based (*italics*) requirement specification: "determine the *inflow events* for all instances of a *resource*." From an implementation perspective, the ontological primitives (inflow, event, resource) become placeholders or variables that can be substituted by actual domain objects and associations at run time. Next, we discuss how such an ontology-based application works when applied to the enterprise system in Exhibit 9 only considering the implementation of the enterprise schema in Exhibit 3.

A potential user interface for the resource acquisition application is shown in Exhibit 10. The application first accesses the information stored in the DomainObjectType and OntologicalType metadata tables and determines all domain object types with ontological type "Resource": Car and Cash. Both resources are shown in the user interface upon

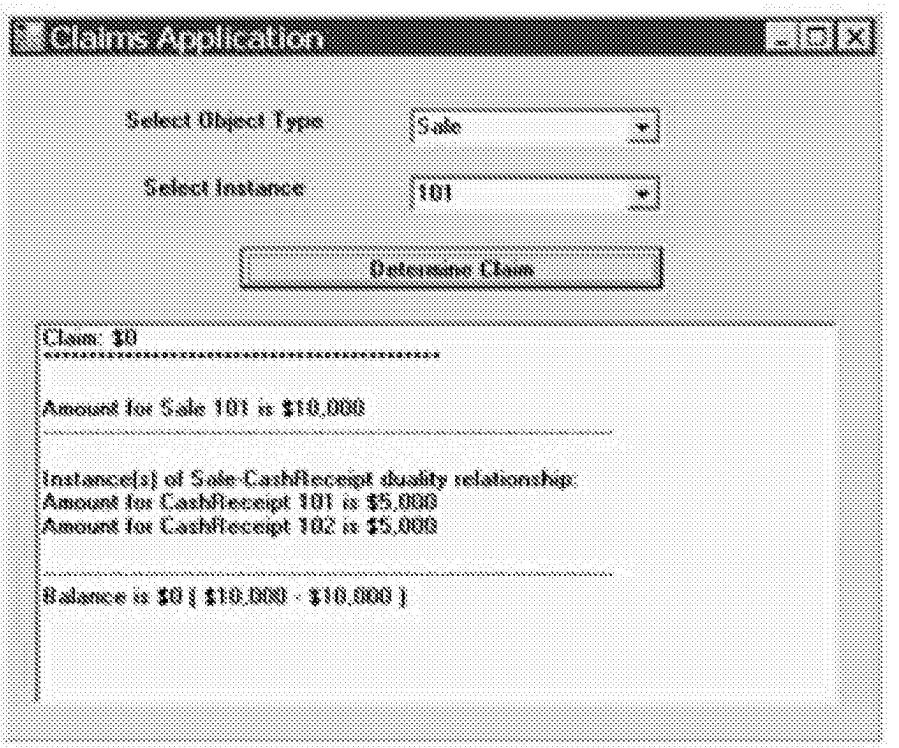
**EXHIBIT 10**
**Resource Acquisition Application: No Trade-Ins**


initialization. The user selects Car, and pushes the "Show Resource Details" button. At that point, the application finds all Association instances in which a Car instance participates and for which the AssociationType has "Inflow" as ontological type. Finally, the application generates the following information for each of the Resource instances: (1) all resource properties (ID, Name, and Year for Car), (2) the name of the event (Purchase), and (3) the ID, DOO, and Amount attributes for the inflow event (Purchase) where ID, DOO, and Amount represent roles. The property with role ID identifies the inflow event. The property with role DOO records the date the event occurred. The property with role Amount represents the amount of the event. All information regarding car acquisitions is shown in the lower part of the user interface in Exhibit 10.

The requirement for our second application is to determine the existing exchange imbalance or claim for an economic event. The following is a more generic definition of this requirement, expressed in terms of the REA-EO: "a claim is the existence of a flow of resources without the full set of corresponding instances of a dual flow" (Geerts and McCarthy 2000). The application should be able to deal with current and future configurations of exchange networks. For example, the same application should be able to accommodate the different duality network configurations in the upper part of Exhibit 5; e.g., an event participating in one duality association or in a network of duality associations. Next, we discuss how the claim application works when applied to the enterprise system in Exhibit 9. Again, we only consider the implementation of the enterprise schema in Exhibit 3 and thus ignore the shaded areas in Exhibit 9.

A potential user interface for the claims application is shown in Exhibit 11. The application first accesses the information stored in the DomainObjectType and OntologicalType metadata tables and determines all event domain object types: Sale, CashReceipt, Purchase, and CashDisbursement. Then, the user selects one of the economic events. For the example shown in Exhibit 11, the user selects Sale. Upon selection of the economic event by the user, the application populates the second drop down box with the instances of the domain object type selected. To determine the ID of the event instances, access to the DomainObjectProperty, Property, and PropertyType tables is required. For our example, the application finds four sales. However, the interface portrayed in Exhibit 11 only shows the first Sale instance, the one with ID "101." Next, the user selects an event

**EXHIBIT 11**
**Claims Application: No Trade-Ins**




instance and clicks the "Determine Claim" button. For our example, the user selects the instance with ID "101." The application then determines the claim amount as follows:

(1) The application determines the amount for the economic event by finding its property with role Amount. The sale (DO_ID = "1") has three properties and the one with role Amount is the property with P_ID = "1" (S_Amount). The amount for Sale 101 is $10,000 (P_VALUE).
(2) The application determines all Association instances of type duality in which the event (DO_ID = "1") participates and sums the amounts of its dual events. For the example in Exhibit 9, the application finds two such Association instances: A_ID = "1" and A_ID = "2." For both dual events (DO_ID2 = "4" and DO_ID2 = "5"), the application will use the information in the DomainObjectProperty, Property, and PropertyType tables to find the value of the property with role "Amount" (CR_Amount). The amount of both cash receipts associated with sale 101 is $5,000, and the total amount for the dual events is $10,000.
(3) The application uses the information from steps 1 and 2 to determine the balance. The remaining balance (claim) of sale 101 is $0 meaning it has been paid off.

Claim information for the selected event is printed as shown in Exhibit 11: "Claim: $0." The lower part of Exhibit 11 further illustrates that the metadata in Exhibit 9 can be

used for the creation of more elaborate reports. An example of metadata integrated in the output is the name of the dual events.

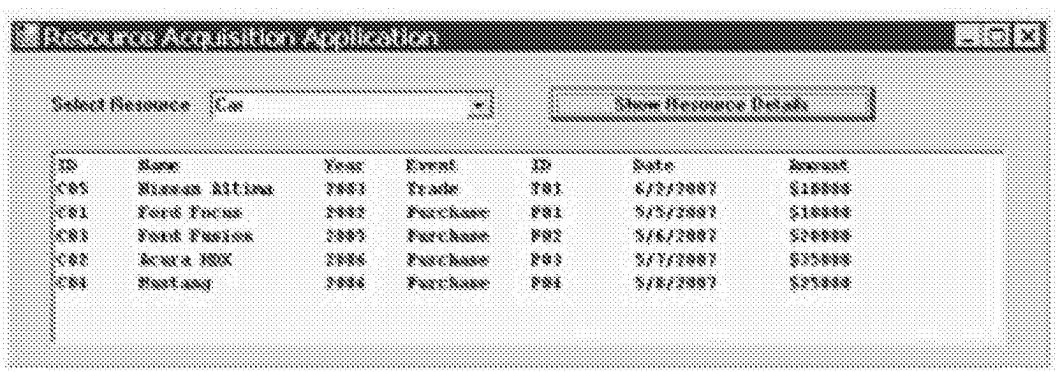
### Adapting Timeless REA Enterprise Systems and their Applications to Changes in Business Practices

Today, companies often change their business practices to cope with new customer demands and market opportunities, such as our car dealership considering trade-ins. With the reflective architecture shown in Exhibit 7, an enterprise system can be adapted to new business practices by manipulating the metadata. The shaded areas in Exhibit 9 illustrate how the enterprise schema in Exhibit 3 can be extended to the enterprise schema in Exhibit 4. A new domain object type with name Trade is added to the DomainObjectType table (DOT_ID = "9") and classified as an Event (OT_ID = "2"). The shaded areas in the DomainObjectTypePropertyType table link Trade to three properties: T_Amount, ID, and Date. T_Amount is defined as a new instance in the PropertyType table and is assigned the role "Amount." Further, three new associations are added to the AssociationType table: Trade-Car, Trade-Sale, and Trade-Customer. The definition of a new association type includes specifying its ontological type and the business rule pattern that applies to it. For example, we define the association between Car and Trade as an inflow (OAT_ID = "4") and specify the following business rules for it: 1..1 – 0..1 (BPR_ID = "5").

With the metadata updated, we are now able to add data. The data entered describe the following transaction: for the sale with DO_ID = "2," the car dealer receives a car (trade-in), in addition to two payments (cash receipts), the value of the car traded is $10,000, and the trade event occurred on 6/2/2007. A new instance is added to the DomainObject table (DO_ID = "13") and declared as a trade (DOT_ID = "9"). In addition, three properties are defined for the newly created trade: its identifier is "T01" (ID), its trade amount is "10000" (T_Amount), and its date is "6/2/2007" (Date). All three properties are linked to the Trade instance in the DomainObjectTypePropertyType table. The trade is further linked to the sale (DO_ID = "2") and the car (DO_ID = "18") in the Association table. The car traded is specified as a 2003 Nissan Altima (ID = "C05").

Neither of the two applications discussed above needs to be recoded when trade-ins are considered. Exhibit 12 shows the new output for the resource acquisition application.

**EXHIBIT 12**
**Resource Acquisition Application: Trade-Ins**



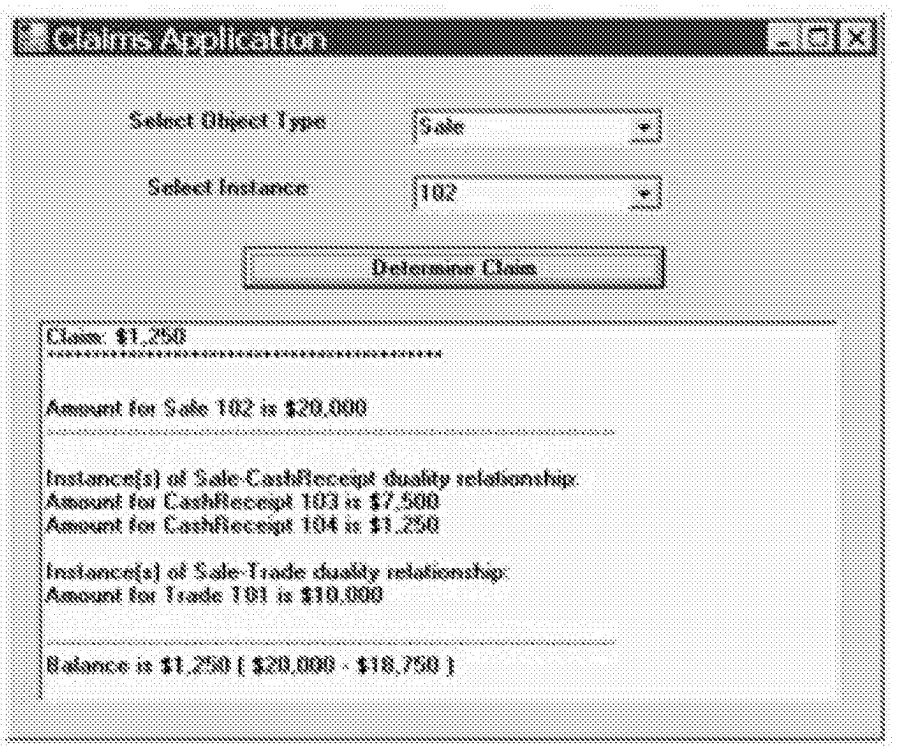
The application now recognizes the existence of an additional car (DO_ID = "18") that has been traded. The transaction's nature (trade) and its three properties are included in the application's output. Exhibit 13 shows the new output for the claims application. The application recognizes the extended set of duality association types in which the sale now participates and correctly calculates the new claim amount: $1,250.

The discussion above shows the adaptability of timeless REA enterprise systems. This type of flexibility is critical for enterprise systems to adapt to a changing business environment in a timely and cost-effective manner.

## CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

Timeless REA enterprise systems combine a leaner enterprise schema (fewer semantics) with extensive descriptions that capture domain-specific semantics and ontological specifications. The expected benefits of such systems are increased adaptability and reusability. On the other hand, their design is substantially more complex and applications are slower since the knowledge-intensive descriptions need to be interpreted. Many issues remain that need further research including the following.

**EXHIBIT 13**
**Claims Application: Trade-Ins**



First, criteria need to be developed that help determine the optimal trade-off among the adaptability, complexity, and efficiency of timeless REA enterprise systems. Second, more research is needed to fully understand the design and implementation of timeless REA enterprise systems. An example of an issue to be addressed is how to keep track of the changes made by a company to its economic activities and business policies over time. Third, possible extensions to the knowledge-intensive descriptions need to be studied, including advanced semantic abstractions, REA-EO extensions, and upper-level ontology specifications. An example of a semantic abstraction that should be considered in future research work is association classes. Examples of REA-EO extensions that should be considered in future research work include commitments, contracts, business events, and workflow (Geerts and McCarthy 2004). Upper-level ontologies capture primarily concepts that are basic to the human understanding of the world (Kiryakov et al. 2001). Examples of concepts taken from Sowa's upper-level ontology (Sowa 1999) that could be considered for extending the ontological specifications in Exhibit 9 include process, juncture, structure, and situation (Geerts and McCarthy 2002). Fourth, other approaches to timeless enterprise systems are being proposed such as service-oriented architectures and model-driven architectures; an in-depth comparison might help to better understand the pros and cons of the different approaches. In addition, the design of enterprise systems that combine elements of the different architectural approaches should be considered.

## REFERENCES

Booch, G., J. Rumbaugh, and I. Jacobson. 1999. *The Unified Modeling Language Reference Manual.* Reading, MA: Addison-Wesley.

Connolly, T. M., and C. E. Begg. 2005. *Database Systems: A Practical Approach to Design, Implementation and Management.* Reading, MA: Addison-Wesley.

Fowler, M. 1997. *Analysis Patterns: Reusable Object Models.* Reading, MA: Addison-Wesley.

Frankel, D. S. 2003. *Model Driven Architecture: Applying MDA to Enterprise Computing.* Indianapolis, IN: Wiley.

Geerts, G. L. 1993. Toward a new paradigm in structuring and processing accounting data. Doctoral dissertation, Free University of Brussels.

———, W. E. McCarthy, and S. R. Rockwell. 1996. Automated integration of enterprise accounting models throughout the systems development life cycle. *International Journal of Intelligent Systems in Accounting, Finance & Management* 5 (3): 113–128.

———, and ———. 2000. Augmented intensional reasoning in knowledge-based accounting systems. *Journal of Information Systems* 14 (2): 127–150.

———, and ———. 2001. Using object templates from the REA accounting model to engineer business processes and tasks. *The Review of Business Information Systems* 5 (4): 89–108.

———, and ———. 2002. An ontological analysis of the economic primitives of the extended-REA enterprise information architecture. *International Journal of Accounting Information Systems* 3: 1–16.

———, and ———. 2004. The ontological foundation of REA enterprise information systems. Working paper, Michigan State University. Available at: http://www.msu.edu/user/mccarth4/.

———. 2004. An XML architecture for operational enterprise ontologies. *Journal of Emerging Technologies in Accounting* 1: 73–90.

Hay, D. C. 1996. *Data Model Patterns.* New York, NY: Dorset House Publishing.

Ijiri, I. 1975. *Theory of Accounting Measurement.* Sarasota, FL: American Accounting Association.

ISO. 1982. *Concepts and Terminology for the Conceptual Schema and the Information Base*, edited by J. van Griethuysen. New York, NY: ANSI.

Kiryakov, A., K. Simov, and M. Dimitrov. 2001. OntoMap: Portal for upper-level ontologies. In *Proceedings of the Euroconference Recent Advances in Natural Language Processing*, 142–148, Tzigov Chark, Bulgaria.

Kleppe, A., J. Warmer, and W. Bast. 2003. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Boston, MA: Addison-Wesley.

McCarthy, W. E. 1982. The REA accounting model: A generalized framework for accounting systems in a shared data environment. *The Accounting Review* (July): 554–578.

Nakamura, H., and R. E. Johnson. 1998. Adaptive framework for the REA accounting model. Paper presented at the OOPSLA'98 Business Object Workshop IV, Vancouver, Canada. Available at: http://jeffsutherland.com/oopsla98/nakamura.html.

Sowa, J. 1999. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Pacific Grove, CA: Brooks/Cole Publishing.

Yoder, J. W., and R. Razavi. 2000. Metadata and adaptive object-models. In *ECOOP'2000 Workshop Reader, Lecture Notes in Computer Science*, 1964. New York, NY: Springer Verlag.

———, and R. Johnson. 2002. The adaptive object model architectural style. In *Proceedings of the 3rd IEEE/IFIP Conference on Software Architecture (WICSA3)*, 3–27, Montreal, Canada.

Zhao, J. L., M. Tanniru, and L. J. Zhang. 2007. Services computing as the foundation of enterprise agility: Overview of recent advances and introduction to the special issue. *Information Systems Frontiers* 9 (1): 1–8.