

Role-playing: A Mechanism for Bridging the Object-Oriented Design-Level Gap

Submitted to the OOPSLA-97 Workshop on
Object Technology, Architectures, and Domain Analysis

James P. Davis
HealthMagic, Inc.
1901 Main Street, Suite 1570, MS 504
Columbia, SC 29201 USA
jimd@healthmagic.com

Ronald D. Bonnell
Center for Information Technology
University of South Carolina
Columbia, SC 29208 USA
bonnell@ece.sc.edu

Abstract. In this position paper, we present *role-playing*--a data point in the technology space for creating synergy between the artifacts and activities associated with domain analysis, software patterns and architectures, and object-oriented frameworks. Role-playing partitions domain analysis into separate modeling activities for the business domain and the underlying application task. It involves creating explicit links between the conceptual “chunks” of reusable task objects (namely, instantiated design patterns or entire architectures) and business artifacts produced via domain modeling (namely business objects and their associations) on which these task-specific artifacts are to operate. Our intent is to support greater productivity in application construction (and greater subsequent code reuse) through methodology enhancements in domain modeling and architecture selection, and through process enhancements for the creation and maintenance of conceptually-defined patterns that are used as a basis for defining frameworks and architectures.

1. The Design-level Gap Defined. It is generally accepted that, given the complexity and size of software applications (or systems) today, we as practitioners are not able to keep up with demand for these applications, or with the corresponding rate of change in hardware technology (as governed by Moore’s Law [1]). Because of difficulties in managing complexity, compounded by the difficulty of collaborating in large teams, we are not making the gains we expected at this point in the evolution of the software industry.

Although Object-Oriented techniques, tools and languages (coupled with education) have allowed the industry to make great strides in productivity, many colleagues are looking for solutions to aid the move to the “next level”. Work in software architecture [2] [3] and in design patterns [4] [5] seeks to raise the level at which we enter the software design process, and the size of the design “chunks” that we manipulate. Rather than focusing on individual object instances and classes, the focus is shifting to components comprised of many classes, to instantiated design patterns comprised of many objects and components, and to software architectures built from patterns layered on application frameworks. We show this movement upward in *design level* in the following table, comparing it to the analogous levels in hardware design.

Level of Perception and Chunking	Software Design Level	Hardware Design Level
Application	Application environment or framework	Application (software layers)
Architectural	Software architecture: selection and evaluation.	System level: macro-architecture, selection and evaluation.
Conceptual	Software abstractions: domain modeling, OOA/OOD, software patterns	Behavioral level: state machines; graphical HDL tools and methods.
Logical	High-level language: C, C++, Smalltalk, etc.	Register level (RTL): high-level hardware description languages (HDLs), micro-architecture & datapath selection.
	Assembly level: compiled program to hardware (PC) architecture	Gate level: logic design, cell selection and device technology binding.
Physical	Machine code: invocation and integration	Geometry: layout and routing.

From the table, the Levels of Perception and Chunking loosely correspond to how we humans perceive these levels, using terminology that is common through many of the sub-disciplines of the computer industry. The hardware design levels correspond to those recognized in that industry, and follow the technology “waves” that have moved the hardware design community to higher levels of abstraction for design entry [6]. The levels associated with software design are not as well recognized. Consequently, this position paper conveys our attempt at organizing and naming them. Note that the levels on the software side are actually placed on top of the levels for hardware, corresponding to the layered “abstract virtual machine” model presented in [7]. The intent of presenting this comparison is to (i) establish some perspective as to what is happening in the software industry by comparing it with what has been happening with hardware design for the past few years; and, (ii) to establish a context from which to introduce some new ideas.

The relative success of the hardware design community in moving up the ladder of design levels can be attributed to the fact that each higher level is defined in terms of the level below it; in other words, there is technology available that allows hardware designers make the transformation. In this position paper, we propose one aspect of a solution for moving up the software levels (as they apply to object-oriented design and development), by allowing a connection to be retained with the adjacent levels below. Thus, we wish to bridge the gap between existing O-O practice, using objects and components, and the evolving higher-level design approaches, comprised of patterns, architectures and frameworks.

2. The Bridge Defined. Our interest is in defining a mechanism that allows us to create a “conceptual” bridge between *classes* and *components*, and the higher-design level artifacts *design pattern*, *architecture*, *framework* and *application*. We focus on the problem of reusing domain models across business applications through the use of components, patterns and frameworks. These high-level chunking elements are used to construct business applications by reusing and adapting application task models.

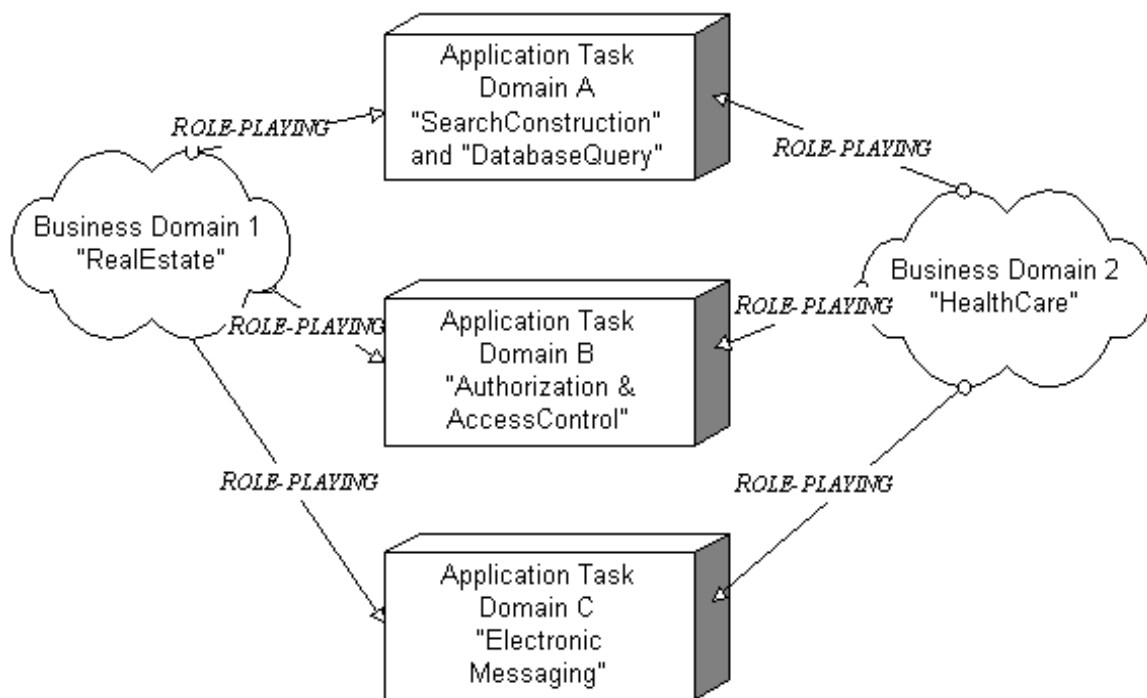


Figure 1. Bridging the Reuse Gap Between Business and Application Domains.

In Figure 1, we depict several business domains for which applications have been built--the first is a Multiple Listing Service (MLS) information system in the real estate industry, the second is a patient record management system for health care. Practitioners in the respective business domains identify the artifacts and objects that are native to these domains by specialized vocabulary and semantics. The relationships and "business rules" are specific each business domain, becoming more specialized within a given business enterprise in the domain.

However, within the business domains--including the two identified in Figure 1--there are other "embedded" domains, often shrouded in the analysis, that are specific to the *application task* to be implemented in software for use in the business enterprise. As shown in Figure 1, these application task domains appear as part of constructed software applications in different business domains. In a sense, the software industry already follows this partitioning--spreadsheet, word processing and DBMS application environments allow mapping from business domain semantics to an underlying application task semantics. We are advocating a higher-level mapping between business domain and application task, starting with a particular focus during domain modeling and analysis, carrying through to construction of the software application from reusable chunks of task functionality using components, patterns and frameworks.

Figure 2 presents a conceptual "world", modeled in UML, in which we define role-playing as a bridging mechanism in the context of how is it associated with components, patterns and architectures. We start with a static structure meta-schema representing the different design levels. At the bottom are the Object, Class, Component and Interface meta-classes. A Component meta-class Contains Classes and Interfaces. We define an Architecture meta-class, which Contains Class and Component meta-classes, or both. Note that the *{or}* constraint restricts an Architecture from not having at least one class or one component as part of its definition. In addition, an Architecture is implemented in a Framework, Contains zero or more InstantiatedPattern classes, HasDomains and is recursively related to other defined architectures via the HasSubArchitecture association. For purposes of illustration, we have modeled only a very small set of the characteristics of a software architecture. The point is that we are establishing a definition of an architecture in terms of its constituent artifacts. The definition can be embellished further at a later time (or during the workshop itself).

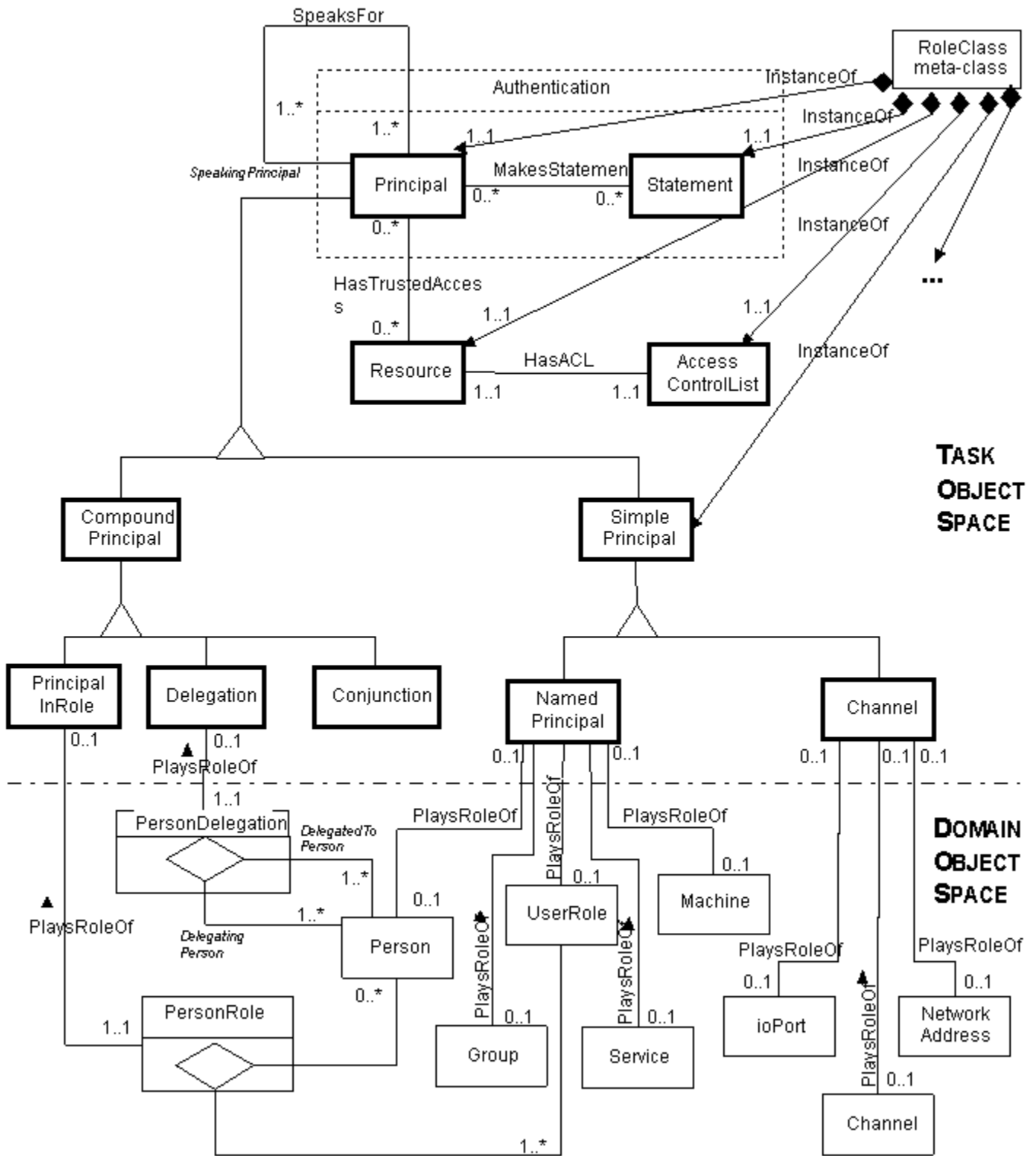
Note also that we have defined a meta-class for an InstantiatedPattern. The literature discusses that design patterns are textual artifacts and don't figure directly into architectures or frameworks; however, their subsequent implementations do [4]. We are attempting to capture the following: (1) patterns, once instantiated, are part of the architectural definition; and, (2) as part of the architecture, they are implemented in the underlying Framework. Once Patterns are implemented, we are restricting them from being implemented directly into the Framework but, rather, through the Architecture. This is an arbitrary constraint, which could be relaxed in our model. Note, also, that the InstantiatedPatterns meta-class has a self-decomposing association via HasSubPattern. We mean for patterns to be nested. Since we are modeling pattern instances, a pattern wouldn't appear as a component pattern in more than one super-pattern or Architecture. This is, again, a limitation in the depicted model; we might choose to model the PatternClasses (not shown) that may appear as components in more aggregate super-patterns or Architectures..

software architectures, it is RoleClasses that participate in construction of InstantiatedPatterns and Architectures. Roles are context-specific definitions of object structure and behavior that are defined solely from the specific point of view of the pattern or the architecture. They have no context outside of their definition in the pattern or architecture. By allowing (and constraining) objects to participate in roles as the basis for acquiring desired application functionality, we provide a mechanism that allows objects to adopt "personalities" without changing their underlying structural and behavioral definition.

The underlying domains for selected design patterns and task architectures are modeled exclusively as a set of role classes, their associations and their methods. The role class, association and method definitions depend upon the vocabulary, structure and semantics inherent in the abstracted design pattern or software task architecture. The primacy of roles in modeling is consistent with the semantics of roles in all the object models (namely, roles are the conceptual artifacts that participate in associations in the domain). However, we make the role playing explicit by defining roles as first-class objects.

Once a pattern or architecture is defined (in terms of its role classes, associations and methods), all connections through the framework from abstract or concrete business object classes, in one or more application domains, are made through the PlaysRoleOf association. In the meta-model, the example is Class meta-class PlaysRoleOf RoleClass meta-class. A role's existence is defined in terms of the collaborations it has with other role classes in the pattern or architectural model. Thus, in order for classes defined outside the pattern or architecture to enter the framework, to take advantage of the high-level facilities and services, they must play the role of one or more role class objects defined in the pattern or architecture. The following example illustrates the concept.

4. Elements of Role-playing. Our approach of using role-playing is based on the following points: The domain analysis starts with a separation of business domain analysis and application task analysis. High-level patterns for solving application-specific problem tasks (such as security authorization, data query, etc.) can be built up within an organization over time. In order to facilitate mapping between business and application task domains, roles should be treated as first-class objects, and the class-to-role associations as first-class relationship objects, in the object analysis & modeling method. RoleClasses are defined as bona fide classes, along the same lines as [8], [9] and [10]. Our role-playing approach differs from how roles are used in other methods, such as Use/Case [11], OMT [12], CRC [13] and UML [14]. As special class objects, roles are the basic building blocks of pattern instantiations and architectures used to realize applications in frameworks. The services and facilities of the framework handle the implementation details of mapping the conceptual role classes into underlying program classes. The creation of application "worlds" from collections of tightly-coupled role-playing objects is analogous to the notion of *contexts* in UML[14]. However, they are operationalized in a similar notion as the framework *plug-in* [5].



Business Object Classes from Potentially Many Domains involved in RolePlaying

Figure 3. Roleplaying Example for Security Authorization Framework.

5. Role-playing Defined by Example. In Figure 3, we have a portion of a security authorization architecture framework modeled in UML, based in a cursory definition provided in [15]. It is our goal to construct an Architecture class representing this model, where it is likely that several of the underlying structural and behavioral phenomena could

Davis & Bonnell. Role-playing. OOPSLA-97 Workshop on Object Technology, Architectures and Domain Analysis Submission Page 6

be supported directly out of the Framework via the InstantiatedPatterns that are known to us. In adapting the Framework and its defined Architecture, we have eight classes of Business Objects, to be defined in the Framework, that will role-play in the architecture. The Person class, for instance, likely has attributes and several interfaces that will be used to adapt the pre-existing architecture and patterns to the current business application domain (the definition of the domain is not explicitly modeled in the figure).

For actual implementation of this example, given a CORBA-compliant framework, the role relationships binding the classes of the various application domain models and the role classes in the task architecture model would be instantiated via the CORBA Relationship Service [16] [17]. A number of the details of this process are left out of this discussion for brevity. There are several ways to implement role-playing in the underlying object technology. In the case of a CORBA-complaint framework, role class objects and interfaces can be accessed via framework services, via multiple inheritance, or mixins (depending on what services are actually available). If the available services were provided by COM [18], role class objects would be accessed from the business objects by creating container classes and exposing the required interfaces.

6. Summary. This position paper has presented one means of examining the interaction of domain analysis, design patterns, software architecture and application frameworks. The enabling mechanism is role-playing, involving use of an OOA/OOD method, a process for creating implementations of modeled objects, a perspective of treating business objects differently from application task objects, and a framework to provide an extended set of services. Our initial investigation indicates this approach is likely to assist in bridging the gap between design levels, thus providing for a more coherent process to create high-level O-O applications.

6. References

- [1] R.R. Schaller, "Moore's Law: Past, Present, and Future", *IEEE Spectrum*, June 1997, pp. 53-59.
- [2] J.O. Coplien, "Idioms and Patterns as Architectural Literature", *IEEE Software*, Vol. 14, No. 1, January-February 1997, pp. 36-42.
- [3] R.T. Monroe, A. Kompanek, R. Melton, and D. Garlan, "Architectural Styles, Design Patterns, and Objects", *IEEE Software*, Vol. 14, No. 1, January-February 1997, pp. 43-52.
- [4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison Wesley Publishing Company, Inc. 1995.
- [5] W. Pree. *Design Patterns for Object-Oriented Software Development*. Addison Wesley Publishing Company, Inc. 1995.
- [6] J.P. Davis, "High-level Design-for-Synthesis: the Next Step", *Electronic Design Automation & Test Conference Proceedings*, Seoul, Korea, August 1995, EDA&T, pp. 378-391.
- [7] A. Tanenbaum, *Structured Computer Organization*. Prentice Hall, Inc. 1976.
- [8] J.P. Davis, and R.D. Bonnell, "A Framework for Constructing Visual Knowledge Specifications in Acquiring Organizational Knowledge", *Knowledge Acquisition*, Vol. 1, No. 3, 1991, pp. 79-115.
- [9] R. Wieringa, W. de Jonge, and P. Spruit, "Roles and Dynamic Subclasses: A Modal Logic Approach", In M. Tokoro and R. Pareschi (eds.), *Object-Oriented Programming, 8th European Conference (ECOOP'94)*, Lecture Notes in Computer Science, nr. 821, Springer-Verlag, 1994, pp. 32-59.

- [10] T. Reenskaug, and W. Reenskaug, *Working With Objects: The OOram Software Engineering Method*. Prentice Hall. 1995.
- [11] I. Jacobson, M. Christerson, P. Jonsson, and G. övergaard. *Object-Oriented Software Engineering A Use Case Driven Approach*. Addison Wesley Publishing Company, Inc. 1992.
- [12] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, Inc. 1991.
- [13] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software*. Prentice Hall PTR. 1990.
- [14] *Unified Modeling Language Proposal to the OMG's Analysis and Design Task Force - Part 1 - UML Definition*. Version 1.0, 13 January 1997. Copyright 1997 Rational Software Corporation.
- [15] B.L. Lampson, "Authentication in Distributed Systems", in S. Mullender (Ed.), *Distributed Systems*. Addison Wesley. 1993. Pp. 543-580.
- [16] R. Orfali, D. Harkey, and J. Edwards. *The Essential Distributed Objects Survival Guide*. John Wiley & Sons, Inc. 1996.
- [17] *OMG CORBA Common Object Services Specification - Volume 2 (COSS2)*, December 1994. Object Management Group.
- [18] A. Denning. *OLE Controls Inside and Out*. Microsoft Press. 1995.