Subjectivity in Object-Oriented Systems Workshop Summary

William Harrison, Harold Ossher, and Hafedh Mili

IBM Thomas J Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598

Abstract

Subjectivity in object-oriented systems is a relatively new research area. At this, the second workshop in this area, discussion surrounded applications, principles, and implementation strategies. The discussions are summarized here.

1. Introduction

The 1995 Workshop on Subjectivity was structured to cover a series of topics, about which the participants could explore their concerns, results, speculations, conclusions, etc.. The topic areas were: Application Requirements, Principles, Subjectivity Support in the Real World, Connections among the Subjective Views of an Object, and User Interaction with Subjective Objects. Each topic area was introduced with a few presentations and proceeded to general discussion. Prior to the beginning of the series of topics, Bill Harrison gave a presentation with the goal of separating various concepts involved, with a view toward establishing a common terminology. Distinguishing between the definitional information and the instantiated information, the "meta"-level contains the concept of ***class***, the defining information about an object. In non-subjective models, a class's definition is unique and that single concept suffices. However, subjectivity requires the introduction of a concept to define the collection of related classes that form a particular subjective viewpoint. This is what he called a ***subject***. The concept of ***object*** exists at the instantiation level — it is an instance of a class. In like manner, subjective models require a term for theinstantiation of a subject. He calls this a ***subject activation***. The fact that the class definition for an object is not complete within any one subject means that the state and methods for an object may be distributed across many subject activations. It is useful therefore to have a term for the state and methods defined for an object by one single subject activation. This has been called a ***quantum*** by Rainer Kossman of BNR. It is important to note that all quanta of an object have the same identity, and that identity is fundamental to the exploitation of subjective views. The concept of ***interface*** does not need to be extended to deal with subjective points of view, but the fact that an object may support many different interfaces raises the distinction between a reference to an object which specifies only its identity and a reference to an object which identifies an interface that constrains the operations that may be used with that reference. However, since the term ***capability*** has already been employed for this latter concept, no new term needs to be coined.

2. Topic I: Application Requirements

The Application Requirements topic covered experience with building systems in which subjectivity has been needed. Two speakers introduced the discussion, Don Batory, from the University of Texas and François Charoy, from CRIN.

Software Components with Subjective Interfaces

Don Batory discussed the role of subjectivity in building domain-specific software system generators. In the ***GenVoca*** system components customize or mutate to enlarge the interfaces they export when they are

instantiated. Software System Generators are at the intersection of software reuse, domain modeling, programming language compilers, software architectures, transformation systems, parameterized programming, and object-oriented software. GenVoca makes use of subjectivity because its objects and abstractions do not just have a single interface. Instead, there is a family of interfaces. The appropriate interfaces are application dependent (subjective), but the available components may not express them all. GenVoca weakens the need to find exact-match software with exact-match views..

GenVoca components are composable because they export and import the same interface. This allows them to be "piped" together as building-blocks. But no single interface will suffice for all domains. In fact, GenVoca components really export domain-specific instead of cast-in-concrete interfaces. So the interfaces for a component change with the addition or removal of other components. For example, consider a size-of layer on top of containers. It exports a read_size, which becomes part of the interfaces of all other components as well, so that they can be composed together. Components are composed by wrappering: for each class, there is an operation that says for each op you want to wrap, here is a an operation that knows how to wrap.

Discussion: A question arose about how to show that with respect to persistence, one component preserves the behavior of the other. The answer was that there is meta-data exchanged up and down between the wrappering layers.

There was also some discussion about what to do when two components provided behavior for a function in a form that did not expect to "wrap" around other function. Discussion elaborated the fact that id encounter that, but not too often.

Additional discussion investigated the question of what state and meta-information are needed about wrapper classes. For example, what if one component wants to employ "wait " and "signal" operations in its wrapper only if the wrapped operation doesn't only use a "read"

Dimensions of Subjectivity in Software Engineering Frameworks

View management is an old issue, but traditional solutions have relied on a known global schema at the start of the project. François Charoy discussed the COO system, which is based on P-ROOT, extended with a workspace service, a transaction service, a lock service and a constraints service. In COO, a component has some object-types and some interfaces. A workspace, with various perspectives, is used to instantiate a component.

Subjectivity can be seen in several dimensions:

Dimension	Characterization / Examples
Value	in different subjects, objects ma have different values. for versioning or history managem
Datatype definition	different subjects may have different schemas, like Relation DB Views, OODB Virtual Classes or PCTE SDS's
Interface	different subjects may exercise different interfaces
Operational	in different subjects, an object may have different behaviors.
Active	events in one subject activation may produce a reaction in other subjects
Semantic	different subjects may employ different transaction or integrity constraints
Concurrency Control:	different kinds of concurrency control may be employed in different subjects
Presentation	
Access Control	
Process	

P-RooT has a limited kind of subjective support. There is no universal "public" interface for an object. Access depends on subject activation using it. P-Root employs a dynamic composition mechanism, but does not make composition explicit. There is therefore no explicit management of relationships between perspectives. They

are implicit in the bodies of operations and somewhat hard to track.

Discussion: There was some discussion on the enforcement of constraints and the use of triggers. Triggers may be cross-subject to check semantic constraints, e.g. quality control subject: to be delivered product must have zero defects, or a product management subject: product must be delivered next week. There is a need to define precedence and who wins in logical conflicts. A similar problem arises with concurrency control: what is precedence of lock types etc. in global lock table.

3. Topic II: Principles

The Principles topic addressed the relationships among the concepts of "subjects", "subjective views of objects", "objects", "roles", etc.. Two speakers introduced the discussion, Bent Bruun Kristensen, from Aalborg University and Pablo Victory, from JP Morgan.

Subjectivity and Roles

Bent Kristensen discussed conceptual modeling approaches. In his work, ***roles*** are a specialization of concepts, and he wants to use them directly in problem analysis. His intention is to use roles, etc., directly as part of a modeling style and then later discuss the mapping of the ideas to different programming languages or support systems.

Consider this example use of roles. John is associated with ECOOP as a reviewer and with OOPSLA as a participant. Roles may appear and disappear during the lifetime of an object. If we take the term "role" to be refer to a meta-level (specification) concept, roles can have their own hierarchies (for specialization). Roles can also be aggregated, e.g. "participant" is made up of being a "traveler" and being a "hotel guest ". It is possible to associate objects and, similarly, to associate a role of one object with a role of another.

A property of an object role may be intrinsic or extrinsic. When you add a role to an object, a relationship can be defined relating a property of the role to an existing property of the object.

A role may add to an object by adding operations, but may also add behavior to an existing operation. In this terminology, a "subject" is the intrinsic object (actually concept) with some of its roles. The subject specifies which of the actual roles do you include when you actually access the thing. (The term subject here is employed not as a meta-level concept, but at the level of instantiations, unlike the term "subject" described in the Introduction. In addition, it refers to a single object's information and not to the characterization of many classes of objects.)

Discussion: There was some discussion of the "life-cycle" problem, in which an object changes the roles it takes on during its life. For each role, this can be modeled as subject which always exists, but in which the object is retyped, e.g. one subject may reflect the states: single, married, widowed, divorced, etc., while another reflects renter, home-owner, dependent. The idea of relationships among roles is closely related to the idea of defining correspondence rules in subject composition.

Real-World Object Behavior

Pablo Victory described various ***forces*** that affect an object's behavior in the real world: ***sender*** force, ***context*** force, and ***state*** force. There is a precedence among them. The sender of a message may affect how an object responds. For example, the way you respond to your spouse's greeting may be different from the way you respond to your dog's. Context also affects the behavior — greetings in public may be different from greetings in private. State also affects behavior. This effect is often defined in an object's code by case-like statements. But it would be preferable if case-like structures were not hidden in the code itself.

Pablo argued strongly that we need to focus on new tools and methods to enable the coding of these facets of an object separately, and that we need new methodologies to enable developers to clearly separate their concerns.

Discussion: Dispatching on the basis of context seems closely related to classical multi-method dispatch, and its associated rules. There was discussion of the question of

the precedence of the "forces". Can an inference engine be used to sort out logical rule structures that relate these?

There was general agreement that a multimethod-like dispatch is a good thing to do, but there were also concerns about the comprehensibility of the resulting system. Decisions about what needs to happen may be too complex to understand. Exchange of metadata is important in mediating these decisions. In "roles" world the sender is assuming an identity as a role. Variation in behavior happens by way of the role interaction. .

Ira Foreman related these interactions to Holt's work on "Rattle" at MCC. The space of objects is divided into roles and there are synchronous interactions among roles, fitting into a petri-net model of what the call looks like. The petri-net alternatives are somewhat like the multiple-dispatch criteria.

4. Topic III: Packaging, Persistence and Polylingualism

The Packaging topic addressed the implications of the fact that with subjectivity, the implementation of an object can exist in fragments, each in a different programming language or paradigm for life-cycle management, for sharing and passing information among the fragments, for naming, and for persistence. Two speakers introduced the discussion, Bill Harrison, from IBM Research, Dirck Riehle, from the Union Bank of Switzerland, and Jack Wileden, from the University of Massachusetts.

Packaging Rules for Subjects

Bill Harrison discussed how rules describing the physical packaging of software within a subject can addressed so that the functional aspacts of an implementation can be separated from the packaging aspects.

A compositor may be responsible for the assignment of the executable code for a subject to processes and to machines. Actual assignments may be made for many reasons: performance, security, integrity, or resource availability. To address these varied requirements, ***packaging rules*** may be given to the compositor indicating how code from various input subjects is to be packaged into processes. The processes may be assigned to specific nodes and various start-up policies can be set in place.

Most object-oriented language implementations assign all of the memory required to represent an object's state in some contiguous space. This implementation is not adequate for many uses of subjects. The storage associated with different subjects' views of the objects may have different constraints with respect to: lifetime, locality to code, security, and transaction behavior Packaging rules can be specified to the compositor indicating how the information needed for different state elements is to be stored. Each different fragment, which we have called a ***facet***, can be assigned to a different part of memory to satisfy the constraints.

The Tools and Materials Metaphor

Dirck Riehle focused on using the tools and materials methodology, at the design level. This approach makes employs the concepts of tools (e.g., lister, formeditor, ...), aspects (e.g., listable, formeditable, ...), and materials: (e.g., folder, contract form, ...).

The materials are arranged in a classification hierarchy. For example, paper has the subclasses: bank information, document, and registration media. Registration Media has the subclasses: Calculation Sheet, and Form With Signature.

Aspects describe the interfaces that are to be supported by classes. For example, Registration Media must support Form Editable, Paper must support Printable and Storable, and Document, Contract, and form must support Host Transmittable.

Adapter technology is then used to describe how the materials various aspects are supported by the adapters. For example, Form Interest Adapter is used to support Form Editable on Registration Media.

Discussion: It was felt that many of the problems arose from lack of subjectivity support. For example, FormEditable support acts as a wrapper to InterestRate Information. There remains the problem of how the

FormEditable interface support determines which concrete class to use for each real object's concrete class to which it is applied. This can be done with factories in which there is a global factory which knows that the FormEditable interface is implemented for InterestRate Information by a class FormInterestAdapter. One of the advantages of the subject-composition models is that it does not need such global factories, but uses the class correspondences instead to make this determination.

A SPIN on Subjectivity

Jack Wileden addressed the need for Convergent Computing Systems — those whose components which come from different computational paradigms or are written in different programming languages. One example of a convergent system is a Persistent Object system, because it combines programming languages and database/file systems.

Convergence manifests the incompatibilities in presumptions about name management, persistence, and interoperability.

Name management determines the meaning of names in a convergent system. Existing name management mechanisms, such as the Unix™ directory structure, search paths, or environment variables and programming language scope rules, both based on the union/override model, are distinctly inadequate.

Interoperability and polylingual access to objects is needed so that developers can have maximum freedom to define object types. Whether the objects are shared or unshared should have minimal impact on the developers of these objects.

It is a distinct challenge to construct a multi-lingual persistent object store. In such a store, persistent quanta of same or compatible types can be created for the same. Transparent access to these multi-lingual object from programs in any language is crucially important. PolySpin is an extension to the SPIN architecture to supports polylingual access

5. Topic IV: Connections among the Subjective Views of an Object

The Connections topic addressed issues of how data and control are shared across the subjective views of an object, with most of the discussion concerning data sharing. One speaker introduced the discussion, Michael Werner, from the Wentworth Institute of Technology.

Why Use Subject-Oriented Programming in Databases

Michael Werner stated the goal of facilitating the development and evolution of suites of cooperating applications.

In his view, different subjects cooperate with each other in order to create a shared database. Sharing implies that objects will be created or modified by one subject and accessed (read) by another. Although each individual subject may perceive the database schema in its own way, sharing necessitates that a common conceptual schema underlie the individual subjective schemas.

Although subjectivity in general goes well beyond views, even this restriction of subjective approaches to views yields important gains in the database area. The restricted subjective views correspond to subschemas, existing at several levels: use case schema, role schema, actor schema, and conceptual schema.

An actor is an entity that is completely external to the system, for example: a professor, student, department head. A single actor might play various roles. The professor is at times a teacher of her classes, at other times an advisor. A graduate student may play the usual student role of taking classes and earning credits towards a degree, but may also play the role of teaching assistant.

Each role may provide for a number of functions. For example, a professor in her teaching role might use the registration system to obtain lists of students in her classes and enter grades, in her advising role she might obtain transcripts of her advisees, enter course substitution approvals, etc.

Composition consists largely of conforming, followed by combining. Conforming is necessary when the views to be composed have different subjective perceptions of the system. These subjective views must be mapped to a common view. Combining consists primarily of merging conformed views.

6. Topic V: User Interaction with Subjective Objects

The User Interaction topic addressed the fact that both object builders and object users have a conception of an object, and the question of how subjects may affect this breakdown into builders and users. One speaker introduced the discussion: Jay Fenton of Electronic Communities.

Dividing Objective from Subjective Knowledge

Jay described his mission as one to create Global Cyberspace. Subjectivity is a critical concept to reach this goal. Global Cyberspace different points of view (avatars in different places), different times (network latency), different roles (a bone is a dogs way of fertilizing a lawn), and different paradigms.

Most people are familiar with the model-view-controller idea, and its variation in which only two elements: (M and VC) are used. The Model is intended to represent the objective world (of trees and stuff) with particular laws of nature, as in the alternate reality kit Randy Smith.

But subjectivity arises from the existence of many perceptors (brains) each of which has its own Model. Here, there is a "relatively objective world" with "identities" that have "attached properties" to which "analytical rules", "objective rules", and the like apply. We need to understand how to characterize the phenomenological boundary between the external world and the internal worlds.

7. General Discussion

Harold Ossher raised the question: "What make a system 'subjective': multiple interfaces per objects, additional dispatch criteria, view synthesis, 'something different' based on context of use, different appearances of an object", noting that there is a breakdown of the monolithic class structures. Donald Cowan observed that one use of subjectivity is to be able to reference arbitrary collections of objects and attribute appropriate behavior to those collections through added, context sensitive interfaces. Ian Simmonds pointed out that certain things are subjectively explicit if you can directly name and pull out those parts of the system – that it's a modularity question. Hafedh Mili commented that subjectivity allows people to develop applications separately and to put them together. Ian Holland noted that the exact meaning of "subject" varies according to which design method is being used. Dirck Riehl objects to the use of the term "subjectivity" since we are transferring meanings about subjectivity from our other experiences, perhaps somewhat inappropriately. Don observed that we're seeing a number of implementations of the same concept. He can see the foam but feels that it is not distilling into a firm set of concepts. Hafedh Mili agreed with Ian that subjectivity can be an excuse for sloppy modeling, but should not encourage it. Hafedh suggested that preplanning the use of different names for the same thing when appearing in different contexts can be a good idea. Don noted that we need to raise the level of abstraction in programming. Bill pointed out that, after all, subjects are to be a higher-level packaging construct. Don said that aiming at the issue of composition is critical. Ian Simmonds noted that composing from smaller elements is more powerful than decomposing from larger ones. One often can't take apart for reuse those things that are already glued together too strongly.