

Roles and Relationships in Object-Oriented Programming, Multiagent Systems and Ontologies

Report on the 2nd Workshop on Roles and Relationships at ECOOP 2007

Guido Boella¹ and Friedrich Steimann²

¹ Università di Torino, Italy
guido@di.unito.it

² Fernuniversität in Hagen, Germany
steimann@FernUni-Hagen.de

Abstract. This report describes the “Roles’07 — Roles and Relationships” workshop held at ECOOP’07 in Berlin on July 30 and 31, 2007. The aims and organization of the workshop are described, and the main contributions of the presentations and invited talks are summarized, so to have a useful survey of current issues in the field. The description of the discussion and conclusions end the paper.

1 Introduction

The notion of *role* is almost ubiquitous in computer science: it occurs in fields such as conceptual modeling, programming languages, software engineering, coordination languages, database systems, multiagent systems, knowledge representation, formal ontology, computational linguistics, and security. Also, it appears to be indispensable outside computer science: fields like sociology, cognitive science, organizational science, and linguistics make heavy use of it. In fact, it seems that like objects and relationships, roles are so fundamental a notion that they should be granted the status of an *ontological primitive*.

The definition of roles inherently depends on the definition of relationships. With the advent of Object Technology, however, relationships have moved out of the focus of attention, giving way to the more restricted concept of attributes or, more technically, references to other objects. A reference is tied to the object holding it and as such is asymmetric — at most the target of the reference can be associated with a role. This is counter to the intuition that every role should have at least one counter-role, namely the one it interacts with. It seems that the natural role of roles in object-oriented designs can only be restored by installing relationships (collaborations, teams, etc.) as first-class programming concepts.

By contrast, the relational nature of roles is already acknowledged in the area of Multiagent Systems, since roles are related to the interaction among agents and to communication protocols. However, even in this area there is no convergence on a single definition of roles yet, and different points of view, such as agent software engineering, specification languages, agent communication, or agent programming languages, make different use of roles.

In computer science, the discussion about roles started in the 1970s with Bachman and Daya [1], and since then, it has kept recurring to the attention of the research community. Interestingly, Bachman developed his Role Data Model as an alternative to the then emerging Relational and Entity-Relationship Data Models [2], fully acknowledging the dependency of roles on relationships. More recently, roles have been used in various areas to handle behavior and interaction, for example, role based access control in security with the RBAC model [3], collaboration roles in UML to describe the interaction among classes [4], channels connecting components in coordination languages [5], the separation of concerns to describe the interaction properties of objects in new contexts in programming languages [6], etc. With the rise of the internet, new communication possibilities and interactive computing created a new demand of research about roles, for example, in organizations in open multiagent systems, in role based programming languages, in using roles for the composition of web services, and in defining roles in standards for interoperability.

Notwithstanding this revival of the research about the notion of role, little agreement seems possible among the proposals in the different fields. This lack of agreement leads to considerable problems with transferring the results from one area to the other, even inside a single area, a consequence which is unacceptable in times in which the sharing of knowledge and standardization alone represent added value in many fields. The likely reasons of these divergences are that many papers on the notion of role fail to have an interdisciplinary character, that much work proposes new definitions of roles to deal with particular practical problems, and that role seems an intuitive notion which can be grasped in its prototypical characters, yet is really an elusive one when details must be clarified. Few proposals, like Steimann [7] or Masolo *et al.* [8], have a more general attitude and try to find a problem independent definition of the role concept and its formalization.

The recognition of the need of a wider agreement on roles lead Guido Boella to organize — together with James Odell, Leendert van der Torre, and Harko Verhagen — the first *Roles* event, titled “Roles, an Interdisciplinary Perspective - Roles’05” [9]. To acknowledge its interdisciplinary character, it was organized as an American Association for Artificial Intelligence (AAAI) Fall Symposium and held on November 3-6, 2005 at Hyatt Crystal City in Arlington, Virginia (<http://normas.di.unito.it/zope/roles05>). The call for papers of Roles’05 produced 30 submissions, of which 22 were presented at the workshop. From the presented papers five were selected for a special issue in *Applied Ontology — An Interdisciplinary Journal of Ontological Analysis and Conceptual Modeling*, representing the different areas involved in the workshop: ontology, programming languages and multiagent systems. Moreover, the article of Friedrich Steimann [2], the invited speaker of the workshop, complemented the other ones by presenting an historical perspective on the subject, analysing the seminal work of Bachman and Daya [1] on the Role Data Model for databases. No other previous event focussed on roles only, even if some other workshops offered the environment for discussing roles, like AOSE’00-’07, CorOrg’05, ’06, NorMas’05, ’07, ’08, VAR’05, COIN’06, ’07. However, they either did not have an interdisciplinary character, or they discussed roles from a specific perspective, for example, NorMas is focused on normative systems.

Like its predecessor Roles'05, Roles'07 aimed at gathering researchers from different disciplines to foster interchange of knowledge and ideas concerning roles and relationships, trying to converge on ontologically founded proposals which can be applied to programming as well as agent languages. Roles'07, organized in the context of the ECOOP'07 conference, attracted 10 submissions (of which 8 got accepted), 10 presentations, and about 30 participants. Invited talks were held by one of the fathers of the notion of roles in modelling languages, Trygve Reenskaug, and by one of today's most active researchers in the study of relationships, James Noble. The proceedings of the workshop are available at <http://iv.tu-berlin.de/TechnBerichte/2007/2007-09.pdf>, the workshop website is <http://normas.di.unito.it/zope/roles07>.

The scope of Roles'07 was outlined by the following list topics:

- Roles as first-class constructs in programming, modelling, ontologies, and multi-agent systems.
- Relationships as first-class programming constructs.
- Applications that would profit from roles and relationships.
- Patterns dealing with the realization of relationships and roles.
- Roles in foundational ontologies and applicative ontologies.
- Roles in models (e.g. UML) and domain-specific languages.
- Roles in multiagent systems design, specification and programming.
- Experience reports with role-oriented approaches.
- Existing and new programming constructs related to roles.
- Literature surveys on roles.
- Reports on roles from other disciplines, like sociology, organizational theory, linguistics, etc.

2 Organizers

Guido Boella (guido@di.unito.it)

Guido Boella received the PhD degree in Computer Science at Università di Torino in 2000. He is currently professor at Dipartimento di Informatica of this university. His research interests include programming languages, multiagent systems, in particular, normative systems, institutions and roles using qualitative decision theory. He organized the workshops on normative multiagent systems (NorMas'05, '07, '08), on coordination and organization (CoOrg'05, '06), the AAI Fall Symposium on roles Roles'05 and the COIN@ECAI'06 workshop. He developed the programming language powerJava, an extension of Java with roles.

Steffen Göbel (steffen.goebel@sap.com)

Steffen Göbel works as a senior researcher in the Software Engineering & Architecture program at SAP Research CEC Dresden. He received his Diploma and his doctoral degree in computer science from Technische Universität Dresden. His main research interests are model-driven development, software product lines and component-based software engineering.

Friedrich Steimann (steimann@acm.org)

Friedrich Steimann is currently a full professor at the Fernuniversität in Hagen, Germany. He heads the department of Programming Systems and conducts research on

object-oriented programming concepts, software modelling, and development tools. He is a leading researcher on roles in object orientation. He has a past in computational linguistics and medical informatics.

Steffen Zschaler (sz9@inf.tu-dresden.de)

Steffen Zschaler graduated from Technische Universität Dresden in 2002, where he is now working as a research assistant. His main research interests are model-driven development, non-functional properties and component-based software engineering. He is currently involved in the Modelplex research project.

3 Contributions of the Workshop

The three different sessions in which we can ideally organize the presentations are: roles and programming languages, roles and relationships, and roles and ontologies. For each one of them, we illustrate the main issues faced by the presentations.

3.1 Roles and Programming Languages

BabyUML - Roles and Classes in Object Oriented Programming [10]. The value of a system is greater than the sum of its parts; the system organization giving the added value. Trygve Reenskaug's talk shows how a system description can be split into state and behavior parts. Despite being one of the inspirators of UML, Trygve Reenskaug is still unsatisfied by the modelling language. So it is proposing an alternative project named BabyUML. The project goal is to create a programming environment with explicit specification of system state and behavior.

Trygve Reenskaug starts from a quotation of Steven Pinker's "How the Mind Works" [11]: he claims that the meaning of a system comes from the meaning of its parts and from the way they are combined. Objects encapsulate state and behavior. Object state is represented in the object's instance variables. Object behavior is specified in the object's methods. The execution of a method is triggered by the object receiving a message. The binding of message to method is dynamic and depends on the implementation of the receiving object. But the value of a system is greater than the sum of its parts. The properties of a system are similar to the properties of an unattached object. System state is the accumulated state of its objects and their associations. System behavior is triggered by messages that the system receives from its environment and is accomplished through an organized process of message interaction between its objects.

Current mainstream programming languages are class oriented; they specify sets of objects with common properties. Class based languages work well in simple cases; but they are less than ideal in complex cases where the system as a whole tends to be hidden among the details of the classes and methods. In this article, Trygve Reenskaug remedies this deficiency by showing how class centred programming can be augmented by role centred programming where system behavior is specified explicitly in terms of collaborations and roles.

The role is a slippery concept. Roles cannot be defined by their shape or their constitution, only by what they do in the context of a system. The essence of object orientation is that in a system objects collaborate to achieve a common goal. Roles are references to

participating objects; each role represents the contribution these objects make towards a system goal. The concept of role conforms to the common usage of the word:

- A role represents a functionality.
- This functionality can be utilized in its collaboration with other roles.
- A role is bound to one or more objects selected from a universe of objects. These selected objects are called the players of the role.
- A role delegates the performance of its functionality to its players.
- A role specifies requirements for its players. An important property is the set of messages that its players must understand.
- The required properties can be implemented by several classes so the players need not be instances of the same class. Different objects can thus perform the same role in different ways.

A collaboration is a structure of interconnected roles that enables the system to perform one or more tasks. First, a collaboration describes a structure of collaborating roles, each performing a specialized function, which collectively accomplish some desired functionality. Second, a collaboration structure constitutes a graph where the nodes are roles and the edges are the message interaction paths.

One of the crucial points in this work is the link between *role* and *objects*. It is annotated by *select from*; this signifies that objects are dynamically selected from a set of relevant objects to play the roles. Many different selection mechanisms can be used. These methods dynamically select the appropriate player objects. In principle, the methods should perform the selection on each call to ensure up-to-date mapping.

Roles for Robots - Roles and Self-Reconfigurable Robots [12]. A self-reconfigurable robot is a robotic device that can change its own shape. Self-reconfigurable robots are commonly built from multiple identical modules that can manipulate each other to change the shape of the robot. Programming a modular, self-reconfigurable robot is however a complicated task: the robot is essentially a real-time, distributed embedded system, where control and communication paths often are tightly coupled to the current physical configuration of the robot, control is distributed across the modules that constitute the robot constraints on the physical size and power consumption of each module limits the available processing power of each module.

The issue of providing a high-level programming platform for developing controllers remains largely unexplored. To facilitate the task of programming modular, self-reconfigurable robots, Nicolai Dvinge, Ulrik P. Schultz, and David Christensen developed a declarative, role-based language RAPL, for the ATRON modular, self-reconfigurable robot, that allows the programmer to associate roles and behavior to structural elements in a modular robot.

Roles provide the abstraction necessary to focus on the behavior of a specific module in a given context. The conceptual view of a role is that it defines the module structure and the active and reactive behavior of each module in a robot. There is a one-to-one mapping between a role and a module, but modules can change their roles (and thus their behavior) as a reaction to messages from other modules or internal events.

An ATRON robot as a whole is implicitly assigned a role using the object oriented concept of a whole-part structure. Behavior for the robot is declared for each individual role. The functionality of the whole and the role that it can play is thus created in

coordination between the individual modules, corresponding to how the control of a modular robot necessarily must be implemented in practice.

A Metamodel for Roles: Introducing Sessions [13]. Role is a widespread concept, it is used in many areas like multiagent systems, databases, programming languages, organizations, security and OO modeling. Unfortunately, it seems that the literature is not actually able to give a uniform definition of roles, there exists several approaches that model roles in many different (and opposite) ways. Valerio Genovese's aim is to build a formal framework through which describe different definitions appeared in the literature or implemented in computer systems by means of different configurations of parameters. In particular, he gives a new role's foundation introducing sessions, which are a formal instrument to talk about role's states and he shows how sessions may be useful to model relationships. In the presented work it has been shown how the great majority of proposed role's accounts can be unified within a general infrastructure which is based on 4 basic notions: Role, Player, Context, Session (where is kept the state of a specific interaction), and where the role notion is pivotal in the interaction of two entities: one that offers the role (context) and the other one that plays it (player). Exploiting the session's formalization, it has been stressed how roles can be employed in specifying relationships in a rich and complete way.

Short presentations. Besides the presentations of papers we had two short presentations. The first one is by Stephan Herrmann, about the definition of a metamodel for the ObjectTeams/Java language he developed [14]. He starts from an intuition of Friedrich Steimann, who in [2] proposes as a criterium for evaluating role models the possibility to use the role model as a metamodel of itself (see Section 5.1).

During the development of ObjectTeams/Java the author developed several metamodels as a way to facilitate implementation in Java. While doing so he fell into the trap of a naive metamodel which contains three fundamental classes: Team (the context of a role), Role and Base (the player of a role). While it looks natural at first, this approach fails to support some combinations, where Teams play roles themselves or Teams contain other Teams. These combinations require a model, where Teams, Roles and Bases are not disjoint sets, but actually each intersection is populated with legal elements, too. Since overlapping classes are problematic in object-oriented design, the author chooses to use Object Teams as the language for the metamodel. The central idea behind this model is to leave the metaclass Class untouched, i.e., to refrain from sub-classing Class to produce Team, Role and Base. Instead the model identifies additional properties that can be attached to a Class if it appears in a certain context. Classes appearing in a Collaboration may play either the role of the Collaboration's Team, or they may play a role of a Role within the Collaboration. This models the fact that Teams and Roles only occur in the context of a Collaboration, where each Collaboration has exactly one defining Team and any number of interacting Roles.

The second presentation is by Roberto Grenna, showing his work with Luisa Leonardelli. This work is based on the intuition of Steimann [7] that design patterns are inherently based on roles, and on the implementation of a role based programming language, powerJava [15]. The target was realizing the implementation in powerJava for some design patterns: the State and the Mediator. The primitives offered by powerJava allows the programmer to use roles in the implementation of the patterns. In particular,

the advantages are that the number of parameters decreases, since roles use the principle of Java inner classes, the number of written classes is less than in the correspondent Java code, and when a role is defined, it can be played by each class implementing the requirements needed for that role. However, it is difficult to give a precise measure of the advantages of using roles in design patterns besides the improvement of the conceptual clearness.

3.2 Roles and Relationships

Member interposition: Defining Classes [16]. The collaborations between objects are the key to understand large object oriented programs. Software systems do not accomplish their tasks with a single object in isolation, but only by employing a collection of objects. Conceptual modeling languages, such as the Unified Modeling Language (UML) and the Entity-Relationship (ER) model, allow explicit representation of object collaborations through associations and relationships, respectively.

Today's languages allow the description of objects through the programming language abstraction of a class, yet they lack a peer abstraction for object collaborations. Programmers must resort to the use of references to indicate collaborations and thereby often hide the intent and, at the same time, further complicate any analysis of a program since references are a powerful, all encompassing programming construct. To use Rumbaugh [17]'s words, "class-based object oriented implementations of object collaborations hide the semantic information of collaborations but expose their implementation details".

As classes allow the description of a collection of individual objects, relationships in relationship-based language proposed by Stephanie Balzer and Thomas Gross allow the description of a collection of groups of interacting objects. The description involves the declaration of attributes and methods. Relationships indicate the classes of which the interacting objects are instances to delimit the scope of the collaboration. Relationship-based languages also allow the declaration of multiplicities (consistency constraints).

Roles emerge from relationships. The participants of a relationship declaration can be named to indicate the conceptual role the particular class plays in the relationship. Some properties of objects only apply when the object is fulfilling a particular role.

Member interposition is a mechanism proposed by the authors to accommodate relationship-dependent properties of objects, without resorting to inheritance and role classes. For example, consider the relationship `Assist` between a collaborating student and a course. If the attribute `instructionLanguage` of the role `teachingAssistant` is interposed, then each student uses the same language in all the courses he assists. If, instead, the attribute is not interposed, the attribute describes the language used by the student in each course it teaches, and it can differ from course to course.

Whereas an interposed member describes a class that plays a particular role in a relationship, a non-interposed member describes the collaboration that exists between the participants of a relationship. The authors also refer to interposed members as participant-level members and to non interposed members as relationship-level members. Like non-interposed members, interposed members are part of the interface

of their defining relationships (and not part of the interface of the classes they are interposed into).

Relationships define roles, objects offer them [18]. Matteo Baldoni, Guido Boella and Leendert van der Torre show how to use the powerJava language (an extension of Java with roles) to model relationships with roles. The powerJava approach is based on the definition of roles as affordances [19]. Inspired by research in cognitive science, this view sees the properties (attributes and operations) of an object as something not independent from whom is interacting with it. In this way, an object affords different ways of interaction to different kinds of objects. The idea behind roles as affordances is that the interaction with an object does not happen directly with it by accessing its public attributes and invoking its public operations. Rather, the interaction with an object happens via a role: to invoke an operation, it is necessary first to be the player of a role offered by the object the operation belongs to. The roles which can be played depend on the properties of the role player (the requirements), as it follows from the definition of affordance. The language extension implements roles as inner classes, so to associate with them attributes and methods, which share the same namespace of the outer class and of other roles: thus, roles are instances having a different identity respect to the players that play them.

Baldoni *et al.* [20] shows how the relationship as attribute pattern to model relationships in OO can be extended with roles, thus endowing the relationship with a state and a behavior, albeit distributed in the two roles composing the relationship. In [18] the authors show how the relationship object pattern can be extended with roles as well. The authors start from an intuition of Steimann [7] who proposes to model roles as classifiers related to relationships, but such that these classifiers are not allowed to have instances. In Java terminology, roles should be modelled as abstract classes, where some behavior is specified, but not all the behavior, since some abstract methods must be implemented in the classes extending them. These abstract classes representing roles should be then extended by other classes in order to be instantiated. However, given that in a language like Java multiple inheritance is not allowed, this solution is not viable, and roles can be identified with interfaces only.

In [18], the authors overcome the problem of the lack of multiple inheritance, by allowing objects participating to the relationship to offer roles which inherit from abstract roles related to the relationship, rather than imposing that objects extend the roles themselves. The advantage of this solution is a tighter coupling between the relationship and its participants, since the roles belong both to the namespace of the relationship and to the namespace of the player, thus having access to the private state of both of them.

3.3 Roles and Ontologies

Role Representation Model Using OWL and SWRL [21]. Roles are very important in ontology engineering. Although the ontology language OWL is a standard, consideration about roles is not enough. This fact can cause to decrease semantic interoperability of ontologies because of conceptual gaps between OWL and developers who need roles. In this paper Kouji Kozaki, Eiichi Sunagawa, Yoshinobu Kitamura, Riichiro Mizoguchi merge an ontological analysis of roles with practical considerations about designing tools for building ontologies, applying it to the OWL language. The most novel

element of their model is the notion of role holder, an abstraction of a composition of a role-playing entity with an instance of a role concept. In turn, the role concept instance can exist only in presence of an instance of the context the role is associated with.

Roles (in the sense of role concept instances) can exist without a player, due to their relation with a context. Conversely, some contexts exist only in presence of the roles which compose the context. For example, a marriage exists only as far as both the roles of wife and husband exist. As Loebe [22] does, the authors provide a classification of roles basing on the type of context they are related to. Besides relational roles, processual roles and social roles, Hozo distinguishes action related roles, attribute roles and composite roles. Composite roles allow to model a role like teacher, which includes both the role of staff member of a school and of agent of a teaching activity, and, thus, they depend on multiple contexts at a time. Details can be found in [23].

The authors represent the Hozo role model in OWL. E.g., `hozo:BasicConcept` class, `hozo:RoleConcept` class and `hozo:RoleHolder` class express basic concepts, role concepts and role-holders respectively. `hozo:playedBy` property represents a relation between classes of role concept and classes of potential player. Besides concept definitions the authors give rules which are applied into classes and properties. They are implemented as SWRL rules. Rules are not only applied to instance models for inference, but also to imply the policies on using the classes and properties underlying Hozo.

The distinction between role concepts and role-holders is realized via the category `hozo:RoleHolder`. Role-Holders are described with the property `hozo:inheritFrom`: it is used for representing that a role-holder inherits definitions both from the role concept and its player. But the property does not imply inheritance of identity, and in that respect `hozo:inheritFrom` differs from `rdfs:subClassOf`.

Towards a Definition of Roles for Software Engineering and Programming Languages [24]. Frank Loebe describes an analytic, ontology-oriented view on roles, starting from the plurality of views and definitions that role has in literature. A major goal of the work is the provision of a role definition which maximizes the coverage of applications of the term “role”. To the extent possible this should be independent from specific application areas, spanning from conceptual modeling to software engineering to linguistics, etc. This leads to a very general, yet weak, analytical definition for the notion of role:

“Definition 1. A role is an entity which is dependent on two other entities, referred to as the player of the role and the context of the role.”

The author classifies three kinds of roles according to the different kinds of contexts they belong to. Roles are parts of contexts (in some sense of the term) and the contexts emerge from the existence of the roles, in a mutual existential dependence. Three kinds of contexts are considered in the classification of role, which determine different playing relations:

- Relational role: corresponds to the way in which an argument participates in the context of some relationship; e.g., two as a factor of four refers to a relationship. Relational roles are special properties, and the “plays” relationship between entities and relational roles is thus subsumed by the “has-property” relation.
- Processual role: corresponds to the manner in which a player behaves in the context of some process (i.e., it participates in the process): e.g., John as the mover of some

pen is categorized as a processual role. Processual roles are parts of processes, and, therefore, processes themselves.

- Social role: corresponds to the involvement of a social object within the context of some society; e.g., a student in a university. Social roles are often defined with their own properties, relations and processes in which they (may) participate (confirming the view of roles as “patterns of behavior”). This means that social roles are considered as objects.

Theories of programming and software engineering, in different shapes like object-, agent-, or aspect-orientation, form a major area of using and applying roles in computer science. In this context it does not appear reasonable to argue for the direct adoption of Definition 1 above, which would be too weak while the broadness of coverage is not necessary. Most occurrences of roles in this area seem to require properties for roles and involvement in complex systems, closely resembling social roles from above (a slight generalization to non-social objects may be required). For a common use of “role” in this area, Loebe proposes the following adaptation of Definition 1 as a working hypothesis:

“Definition 2. A role R is an entity which mediates between a context C , comprehended as a system or a society of interrelated entities E_1, E_2, \dots , and exactly one of these entities, E_i . R depends on both, E_i and C , and it exhibits specific properties and behavior.”

Note that this definition exhibits some similarity to the UML 2.0 definition of role [4, p.575]: “Role: A constituent element of a structured classifier that represents the appearance of an instance (or possibly, a set of instances) within the context defined by the structured classifier.”

4 Invited Talks

4.1 The OOram Software Engineering Method

For this workshop Trygve Reenskaug, one of the fathers of object orientation and ideator of the model-view-controller pattern for GUI software design in 1979, while visiting Xerox Parc, resurrected the software for modelling with the OOram Software Engineering Method. OOram is a precursor of the UML modelling language, but it is still impressive the simplicity it offers to model a system in terms of roles, only later passing to the design of classes.

On his webpage (<http://folk.uio.no/trygver/>) Trygve Reenskaug says:

“Roles are about objects and how they interact to achieve some purpose. For thirty years I have tried to get them into the main stream, but haven’t succeeded. I believe the reason is that our programming languages are class oriented rather than object oriented. So why model in terms of objects when you cannot code them? And why model at all when you cannot keep model and code synchronized?”

So the discussion after his invited talk could not but lead to the question: what are the reasons of the difficulty of introducing the notion of role in the object oriented community? Under the light of his long experience in proposing roles as a fundamental

concept, Trygve Reenskaug's answer is the cognitive difficulty which people experiences in conceptualizing both the notions of class and role. Witness the fallacy, made also in prominent books about OO, of saying that classes send or receive messages. Only objects send and receive messages and, if a more abstract notion is needed to precise a model, we should better say that roles send and receive messages. Instead, we keep teaching to students in programming language courses to think only in terms of classes and objects as the only elements when designing a program, thus perpetuating the difficulty of using roles.

He considers, however, a positive evolution in OO the recent debate about the notion of trait, which seems to be able to decompose a class into the different behaviors required by the specific roles that its instances can play.

Finally, Trygve Reenskaug gives us the explanation about the genesis of collaboration roles in UML. He was not directly involved in the standardization of this part, and he does not approve the idea of roles as classifiers in UML 1.0. He considers UML 2.0, where roles are properties through which the interaction between two objects takes place, an improvement with respect to UML 1.0.

4.2 Where Are the Relationships?

James Noble starts from a quotation about the programming language Smalltalk: "a program ... a community of communicating objects". Thus, the question of his talk becomes: "But where are the relationships?".

The need of introducing the notion of relationship as a first class citizen in Object Oriented programming, in the same way as this notion is used in OO modelling, has been argued by several authors, at least since Rumbaugh [17]: he claims that relationship are complementary to, and as important as, objects themselves. For this reason, they should be available in programming languages too, and not only in modelling languages, like UML or ER, either as primitives, or represented by means of suitable patterns. Noble [25] proposed two main alternatives for modelling relationships by mean of patterns: the relationship as attribute pattern, reducing the relationship to references, and the relationship object pattern, introducing a new class to represent the relationship. Each of these two solutions have pros and cons, as discussed by Noble [25].

However, we would like to have relationships as a real programming primitive. Many languages offer such primitives: Two-way Pointers, JavaFX, ReJ Relationships, RJ Relationships.

These solutions do not capture all the characteristics we would like to have in a relationship primitive; first, it should be abstract: its implementation is encapsulated. Second, participants and clients should be decoupled from implementation, so to have polymorphism and relationship interfaces. Third, to have reusability, relationships should not depend on participants and multiple instantiation of relationships and participants should be possible. Fourth, there should be compositionality, so to build new relationships from old ones. Finally, separation of concerns should hold: participants should be reusable without the relationship and reusable without each other.

These properties are satisfied by the Relationship Aspect Library proposed by Noble [26]. This proposal, however, does not satisfy some other requirements. For example, Balzer *et al.* [27] allows to add invariants to relationships, JQL and RAL

allow to access the relationships of databases. In summary, the hypothesis is: $\text{Relationship} = \text{Tuples} + \text{Roles} + \text{Extents}$. Where, tuples are associations between participating objects, per-tuple data and behavior are classes, per-participant data and behavior are roles and collections of tuples are the extents.

This raises the question: what comes first? relationships or roles? Noble's hypothesis is that roles are relationship monopolies. A role provides behavior - perhaps with some state. But it makes sense only in the context of a relationship - where the object is playing that role. Thus, support for relationships should precede the support for roles.

5 Discussion

5.1 The Metarole Challenge

In his presentation titled "Towards a Role Manifesto", Friedrich Steimann made the point that if the attendees agreed on his position that the role concept deserves the status of an ontological primitive, roles should be found in the metamodel of any modelling formalism offering roles. According to Steimann's argumentation, this ruled out the role-as-adjunct-instance representation of roles, since this capture, like the Role Object Pattern, resorts to roles itself (namely the Role role and the Subject role), starting an infinite regress. In fact, so Steimann argued, any metamodel introducing the notion of roles (which a metamodel should do if role is to be regarded an ontological primitive) while at the same time resorting to the concept of role should use the very modelling construct it defines, or it raises doubts of the so defined role construct being the acclaimed primitive. Because roles are tied to relationships and metamodels without relationships seem unthinkable, any metamodel will also use roles, and the role concept the metamodel defines had better be (very similar to) the one it uses. Otherwise, the question why the one it uses is unsuitable for the modelling language it defines must be answered. Steimann conceded that this still leaves room for using the role-as-adjunct-instance approach to representing roles on the model level, but made clear that this capture cannot be the ontological primitive being sought for.

5.2 Group Discussion

Following Steimann's presentation, a discussion was started based on the following questions:

- Roles and state: Does each instance of role playing come with its own state? Or is this state part of the relationship? Does playing a role alone contribute to the state of an object?
- Role and relationship: Can roles be viewed independently from relationships?
- Composability of collaborations: Can a system be lumped together as a set of collaborations? If not, what else needs to be provided?
- Role interference: How can we deal with the fact that an object participating in several collaborations can change its state in unpredictable manners (as seen from each collaboration)?
- Why don't current programming languages come with a role construct?

To address these and other questions, the participants divided into four groups according to their different perspectives:

1. **Ontology group** (Kouji Kozaki, Frank Loebe, and Riichiro Mizoguchi): How are roles defined in ontologies? Which are their properties? Are roles an ontological primitive? In the ontology community there is an increasing consensus towards the idea that roles are entities (also called *qua entities*) which depend not only on the player of the role but also on a context. In fact, discussants agreed on the idea that the relationship of role to context is more prominent than that to role player and that roles can exist in some cases even without their players.
2. **Multiagent systems group** (Guido Boella and Thibaud Brocard): How are roles used in multiagent systems? Which is the relation between roles and organizations? Which are the properties attributed to roles in such field? The main debate in the multiagent community is whether to use roles as first class entities at runtime or only as a modelling concept. Recent works are leaning towards the first position. In this field, the concept of role is strictly related to the notion of organization which provides the context of the role. Roles are associated not only with a state and behavior but also with obligations, permissions, and institutional powers.
3. **Roles and relationships group** (Stefanie Balzer, Valerio Genovese, and Ulrik Schultz): Which is the relation between roles and relationships? Which of these notions come first? Can roles be reduced to relationships or vice versa? Roles are a promising abstraction when we have to understand the behavior of large systems. Roles are dependent on collaborations, and they are at the core of the answer to the question: how to compose collaborations safely? Roles belong to collaborations and players can be seen as a constellation of the roles they play. The interaction inside the collaboration happens via the roles played by objects. Roles, since they depend on the collaboration, can exist even detached from their players.
4. **Roles in modelling and programming languages group** (Uwe Assmann and Trygve Reenskaug): Which is the role of roles in modelling and programming systems? The result of the discussion goes against the position of the invited talk of James Noble: roles have priority over relationships. To summarize, roles can be seen from two different and opposite points of view, depending on how they are related with relationships. In the first view, roles are simply labels of their players in the relationship. Whereas, in the second view, roles identify a state of the particular interaction they describe, and they have an identity. In this approach, it is possible to extend the interaction capabilities of the players adding attributes and behavior, or also unifying the state of the interaction within a single entity (the relationship). Of course, with this approach some problems arise as concerns the coherence of the states of the two roles. The problem of roles and collaborations should be faced by studying respectively, the structure of the systems, its invariants and then its behavior, like proposed in the metamodel of roles of Genovese [13].

Friedrich Steimann as the moderator tried to reach a common agreement among the different groups, but the result was that the different communities are still on diverging positions. Moreover, the moderator had to accept that the discussants all insisted on roles having their identity separated from that of the player, that in fact roles can even exist without a player.

6 Conclusions

While relationship is a widely accepted notion, indeed one whose definition is (except perhaps for small variations) clear and generally agreed upon, role is still not. With this workshop, we tried to make the unequivocalness of the relationship concept carry over to the role concept, by putting the inherent relatedness of the two into focus. To a certain extent, this approach has failed: it seems that the traditional differences of the role conceptions created by the different disciplines are stronger than the commonalities suggested by the fact that the very concept of role is meaningless without that of counter role, and thus without relationship. To a certain extent, however, it has also succeeded: it showed that all application scenarios in which roles are identified and used are sufficiently similar to agree on its reason for existence and its general purpose (but not its representation!), and that there is a common vocabulary in which the differences can be formulated.

Acknowledgments

We first of all would like to thank the two invited speakers Trygve Reenskaug and James Noble for their stimulating talks, and all the participants of this workshop. Moreover, we would like to thank the organization of ECOOP'07 for the support offered to the invited speakers. We extend our thanks to all those who have participated in the organization of this workshop, in particular to the program committee, composed by: Uwe Assmann, Technische Universitaet Dresden; Colin Atkinson, Universitaet Mannheim; Matteo Baldoni, Università di Torino; Giancarlo Guizzardi, LOA-CNR Trento; Stephan Hermann, Technische Universitaet Berlin; Pierre Kelsen, University of Luxembourg; Claudio Masolo, LOA-CNR Trento; James Odell, Intelligent Automation, inc. Rockville MD; Andrea Omicini, DEIS Università di Bologna; Kasper Østerbye, IT University of Copenhagen; James Noble, Victoria University of Wellington; Daniel Oberle, SAP Research; Elke Pulvermueller, University of Luxembourg; Dirk Riehle, SAP Research, SAP Labs, LLC - Palo Alto, CA; Trygve Reenskaug, University of Oslo; Leendert van der Torre, University of Luxembourg; Harko Verhagen, DSV, KTH/SU.

References

1. Bachman, C., Daya, M.: The role concept in data models. In: *Procs. of VLDB 1977*, pp. 464–476 (1977)
2. Steimann, F.: The role data model revisited. *Applied Ontology* 2(2), 89–103 (2007)
3. Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role-based access control models. *IEEE Computer* 2, 38–47 (1996)
4. Rumbaugh, J., Jacobson, I., Booch, G.: *Unified Modeling Language Reference Manual*. 2nd edn., Pearson Higher Education (2004)
5. Arbab, F.: Abstract behavior types: A foundation model for components and their composition. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) *FMCO 2002*. LNCS, vol. 2852, pp. 33–70. Springer, Heidelberg (2003)
6. Kendall, E.A.: Role model designs and implementations with aspect-oriented programming. In: *Proceedings of OOPSLA 1999*, pp. 353–369. ACM Press, New York (1999)

7. Steimann, F.: On the representation of roles in object-oriented and conceptual modelling. *Data and Knowledge Engineering* 35, 83–848 (2000)
8. Masolo, C., Vieu, L., Bottazzi, E., Catenacci, C., Ferrario, R., Gangemi, A., Guarino, N.: Social roles and their descriptions. In: *KR 2004. Procs. of Conference on the Principles of Knowledge Representation and Reasoning*, pp. 267–277. AAAI Press, Menlo Park (2004)
9. Boella, G., Odell, J., van der Torre, L., Verhagen, H. (eds.): *AAAI 2005 Fall Symposium on Roles, an interdisciplinary perspective (Roles 2005)*, Arlington, VA, 03/11/05-06/11/05. Volume FS-05-08 of AAAI Technical Report. AAAI, Menlo Park (2005)
10. Reengskaug, T.: Roles and classes in object oriented programming. In: *Roles 2007. Proceedings of the 2nd Workshop on Roles and Relationship in Object Oriented Programming, Multiagent Systems, and Ontologies (2007)*
11. Pinker, S.: *How the Mind Works*. Norton, New York (1997)
12. Dvinge, N., Schultz, U.P., Christensen, D.: Roles and self-reconfigurable robots. In: *Roles 2007. Proceedings of the 2nd Workshop on Roles and Relationship in Object Oriented Programming, Multiagent Systems, and Ontologies (2007)*
13. Genovese, V.: A meta-model for roles: Introducing sessions. In: *Roles 2007. Proceedings of the 2nd Workshop on Roles and Relationship in Object Oriented Programming, Multiagent Systems, and Ontologies (2007)*
14. Herrmann, S.: A precise model for contextual roles: The programming language Object-Teams/Java. *Applied Ontology* 2(2), 181–207 (2007)
15. Baldoni, M., Boella, G., van der Torre, L.: Interaction between Objects in powerJava. *Journal of Object Technology* 6, 7–12 (2007)
16. Balzer, S., Gross, T.R.: Member interposition: Defining classes. In: *Roles 2007. Proceedings of the 2nd Workshop on Roles and Relationship in Object Oriented Programming, Multiagent Systems, and Ontologies (2007)*
17. Rumbaugh, J.: Relations as semantic constructs in an object-oriented language. In: *Procs. of the OOPSLA-87: Conference on Object-Oriented Programming Systems, Languages and Applications*, Orlando, FL, pp. 466–481 (1987)
18. Baldoni, M., Boella, G., van der Torre, L.: Relationships define roles, objects offer them. In: *Roles 2007. Proceedings of the 2nd Workshop on Roles and Relationship in Object Oriented Programming, Multiagent Systems, and Ontologies (2007)*
19. Gibson, J.: *The Ecological Approach to Visual Perception*. Lawrence Erlbaum Associates, New Jersey (1979)
20. Baldoni, M., Boella, G., van der Torre, L.: Relationships meet their roles in object oriented programming. In: *Arbab, F., Sirjani, M. (eds.) FSEN 2007. LNCS, vol. 4767*. Springer, Heidelberg (2007)
21. Kozaki, K., Sunagawa, E., Kitamura, Y., Mizoguchi, R.: Role representation model using owl and swrl. In: *Roles 2007. Proceedings of the 2nd Workshop on Roles and Relationship in Object Oriented Programming, Multiagent Systems, and Ontologies (2007)*
22. Loebe, F.: Abstract vs. social roles - towards a general theoretical account of roles. *Applied Ontology* 2(2), 127–158 (2007)
23. Mizoguchi, R., Sunagawa, E., Kozaki, K., Kitamura, Y.: A model of roles in ontology development tool: Hozo. *Applied Ontology* 2(2), 159–179 (2007)
24. Loebe, F.: Towards a definition of roles for software engineering and programming languages. In: *Roles 2007. Proceedings of the 2nd Workshop on Roles and Relationship in Object Oriented Programming, Multiagent Systems, and Ontologies (2007)*
25. Noble, J.: Basic relationship patterns. In: *Procs. of EuroPLOP (1997)*
26. Pearce, D., Noble, J.: Relationship aspects. In: *Procs. of AOSD*, pp. 75–86 (2006)
27. Balzer, S., Gross, T.R., Eugster, P.: A relational model of object collaborations and its use in reasoning about relationships. In: *Ernst, E. (ed.) ECOOP 2007. LNCS, vol. 4609*, pp. 323–346. Springer, Heidelberg (2007)