Object-Oriented Modeling with Roles *

Bent Bruun Kristensen
Institute for Electronic Systems, Aalborg University
Fredrik Bajers Vej 7, DK-9220 Aalborg Ø, Denmark
e-mail: bbkristensen@iesd.auc.dk

Abstract

Objects relate to each other in different ways — serving, using, and communicating with each other. From the way in which they treat one another, objects have different perspectives of each other. These perspectives define the role that an object may play towards another. The perspectives are formed as a restricted set of methods of the object, — exactly the methods that are relevant for the relations between the objects. Different roles exist for different purposes, and the roles played by an object may change over time. The role is a powerful modeling concept in object-oriented analysis, design, and programming. A graphical notation is defined to support static and dynamic description of roles. The notation supports generalization and part-whole hierarchies for roles, the extension of methods and active objects with roles, and the integration of roles and locality.

1 Introduction

Objects are often described as isolated entities. We design objects as independent units with one uniform set of capabilities. The object has identity. All information, except the capabilities, are encapsulated in the object. However, objects are not isolated entities, — they are *related* to other objects, they *interact* with other objects, and they play certain *roles* for each other in these relations and interactions. The vital relations and interactions between objects are to some extent in conflict with the idea of an object as an independent, delimited unit. This paper addresses roles of objects as a means of refining the understanding of an object as a monolithic unit. A graphical notation is defined and a comprehensive example is presented in this notation. The theoretical foundation in terms of conceptual abstraction theory for the notation is to a large extent given in [Kristensen & Østerbye 95].

The overall motivation for roles is to allow special perspectives on a phenomenon — modeled by an object. A perspective is used by other objects in the model as a restricted, selective way of knowing — and accessing — the object. The perspective is a set of selected properties of the phenomenon — modeled by a set of methods. Other objects can access the selected set of methods. An important property of such perspectives is that they can change dynamically. This means that the set of methods of an object may have dynamic additive and subtractive properties. The perspective is modeled by a language construct — the role. A role will include a set of methods, but can also include state (in the form of for example instance variables) when it is instantiated in addition to an object.

The power of roles is to give restricted, possibly complementary perspectives on a complex and compound object, and to do this dynamically in order to support dynamicity in the composition. This is important because it corresponds to an essential understanding of how we conceive and conceptually model the world around us. We think and express ourselves in terms of roles. We organize our understanding in terms of different perspectives on phenomena (and the concepts formed mentally to cover these) and the dynamicity of such perspectives. As an example a person has several roles. He or she may be a student at a given time. Later he/she may be an employee — or a student and an employee at the same time. Independently of such roles he/she may become a parent, while a person will always be a child (of his/her parents, even when they have passed away). A person has several roles, that have been chosen in order to fulfill the objective of the modeling. The roles may change, they may exist simultaneously, and there may be other important relations between roles.

The concept of a role is intuitive and important in the modeling of real world phenomena. We need a notation for roles for use in object-oriented modeling, i.e. a graphical notation for use in object-oriented analysis and design, and an abstraction mechanism in object-oriented programming languages. We model concepts/phenomena by means of classes/objects and roles/role instances. The model includes the properties of the concepts/phenomena. Properties are represented and available in the objects/role instances by means of methods.

The object, to which a role is allocated, will be referred to as the *intrinsic* object. The methods of an

*This research was supported in part by the Danish Natural Science Research Council, No. 9400911

intrinsic object are referred to as *intrinsic methods*. The methods of a role are referred to as *extrinsic methods*, and the instantiations of roles are referred to as *role instances*. An intrinsic object *has* or *plays* a role when a role instance of that role is allocated to the object. We refer to an object with all its roles as a *subject*.

Paper Organization and Contributions. In section 2 we introduce modeling with roles and conclude by stating the characteristics of roles. In section 3 we illustrate the simulation of roles by specialization, aggregation and association. We outline the problems in relation to the characteristics of roles with these simulations. In section 4, 5, 6, 7, and 8 we illustrate and motivate the use of roles as abstractions in object-oriented modeling. We define generalization and part-whole hierarchies for roles, associations between roles, roles for methods, and localization of roles. We present a graphical notation for roles to be used in object-oriented analysis and design. The notation supports static and dynamic aspects of the descriptions. In section 9 we discuss experiments in programming language support of roles and associations. In section 10 we discuss related work as well as the results of this paper and some remaining challenges in relation to roles. In the appendix we include the complete example and a summary of our notation.

2 Modeling With Roles

We shall use the "Conference Organizing Problem" [Olle et al. 82] as an illustrating example. We are not trying to solve what this model originally tried to solve. Instead we use it as a well-known context to illustrate our ideas. We illustrate and motivate the use of roles as abstractions in object-oriented analysis and design by a comprehensive example from this problem throughout the paper.

We restrict the model to deal with some "OO" organization only. `OO-Associate` models any person who is related to "OO" and therefore registered in some general register of `OO-Associate`s. Class `OO-associate` is not related to any particular conference, but models only the general information available for persons associated with the "OO" organization, such as for example `Name`, `Address` and `Member-Id`. An actual conference is modeled by the class `Conference`. Related to a conference are an instance of class `Program`, and some instances of the class `OO-Associate`.

For the conference we are not as much interested in `OO-Associate` as in the roles each of these instances are playing in relation to the conference, such as e.g. `Participant`, `Author`, `Reviewer`, `Speaker`, `Panelist`, or `Session-Chair`. We model this situation by introducing roles explicitly in the model, for example the role `Conference-Associate` of `OO-Associate` and then relating `Conference-Associate` to `Conference`. During the preparation of the `Conference` the `Conference-Associate` may play the roles `Participant`, `Author`, `Reviewer`, and — when actually attending the conference — the `Participant` may further play the roles such as `Speaker`, `Panelist`, or `Session-Chair`.

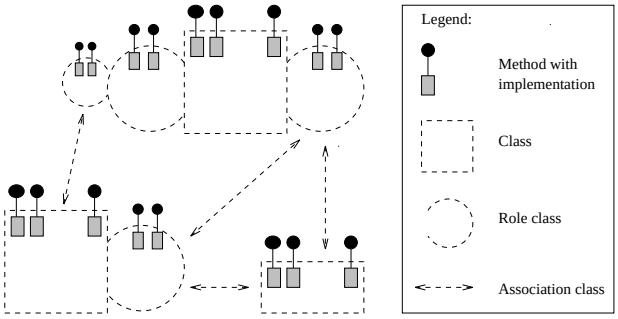


Figure 1: Graphical Notation for Roles

In Figure 1, we illustrate the notation proposed for the (static) description of roles in object-oriented modeling in a schematic diagram. A box — with dotted borders — illustrates a class. The handles on the box illustrate the methods of the class. A semi circle — also dotted — illustrates a role class. The semi circle may be drawn glued onto a class (or another role) or may appear isolated, but linked to a class/role. Also roles may have handles illustrating the methods of the role. A dotted line with arrows illustrates an association class — with some classes (and roles) as domains.

In Figure 2, we illustrate a sequence of snapshots from a specific example involving participants in conferences in "OO". In the illustrations the dotted lines in classes, associations and roles are replaced by solid lines to illustrate instantiations of these, respectively objects, association objects, and role instances. The object `John` is involved in the `ECOOP'95` conference, and he plays the role of a `Conference-Associate`. This is illustrated by the role instance `CA1`, which represents the information of `John` in relation to this conference, as an instance of the role class `Conference-Associate`. Next, `John` is also involved in the `OOPSLA'95` conference. Again he plays the role of a `Conference-Associate`, illustrated by the role instance `CA2`. At the same time he has become a `Reviewer` in relation to `ECOOP'95`. This is illustrated by the role instance `R1` of the role class `Reviewer`,that is a role of the role class `Conference-Associate`. Finally, the `ECOOP'95` is over. From the snapshots we cannot tell whether `John` actually attended the conference. Anyway, he has become a `Conference-Associate` of `TOOLS'95`, illustrated by `CA3`. Also he is actually

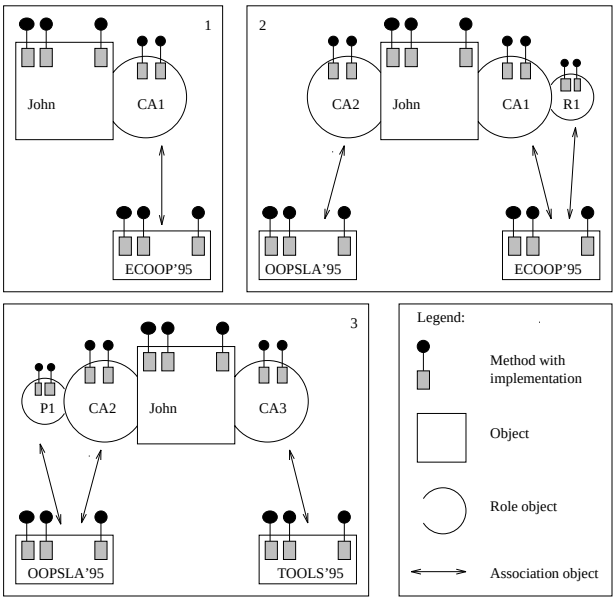


Figure 2: Snapshots of Example with Roles

attending the `OOPSLA'95` conference by becoming a `Participant` in relation to `OOPSLA'95`, illustrated by the role instance `P1` of the role class `Participant`, that is another role of the role class `Conference-Associate`.

Characteristics of Roles. The characteristics of roles include the following:

C1: *Visibility:* The visibility of — and access to — an object can be restricted to (the methods of) a role, including those of the object, but excluding the methods of other roles. This may also be seen as the possibility for multiple (disjoint) classification of an object.

When `John` is a `Reviewer` for `ECOOP'95` his `Reports`, in relation to some `Papers` submitted to this conference, will be available to whom it may concern, but not to anybody just because of his relation to the `OOPSLA'95` conference. Furthermore, when `John` is known as a `Reviewer` he is also known as an `OO-Associate` so that `Name`, `Address` etc. are available too.

C2: *Dependency:* The role is dependent on the object, — it cannot exist without an object. The methods of the role can be defined in terms of the methods of the object, but not vice versa.

There is no such thing as a `Reviewer` in relation to the `ECOOP'95` conference without a `Conference-Associate` and `OO-Associate`, — `John` needs to be there and he needs to be related to the `ECOOP'95` conference. `John`'s `Schedule` as a `Reviewer` must include — and thus be defined in terms of — his `Schedule` as (a `Conference-Associate` and) `OO-Associate` in which the data of his usual business is registered. But his `Address` as an `OO-Associate` cannot be defined in terms of his temporary address for example as a `Participant` in the `OOPSLA'95` conference.

C3: *Identity:* An object and its actual roles have one identity, — it is seen and can be manipulated as one entity.

`John` plays the role of a `Conference-Associate` for the `ECOOP'95` and `OOPSLA'95` conferences. Anyone from any of these conferences, who knows about `John`'s role as `Conference-Associate`, should also be able to know about the identity of the object representing `John`. In particular, we should be able to identify that when knowing the two `Conference-Associate` roles of `John` we also know the same `OO-Associate` person (`John`).

C4: *Dynamicity:* A role may be added and removed during the lifetime of an object.

`John` is a `Conference-Associate` of `ECOOP'95` only for a period of time. Also his role as a `Reviewer` for this conference is only a part of the period of time in which he is a `Conference-Associate` of `ECOOP'95`.

C5: *Multiplicity:* Several instances of a role may exist for an object at the same time.

`John` is a `Conference-Associate` to both `ECOOP'95` and `OOPSLA'95` at the same time. He is related to whoever relevant from these conferences through these roles, and the roles represent specific information in relation to these conferences.

C6: *Abstractivity:* Roles can be classified and roles can be organized in generalization and aggregation hierarchies.

The activities making up the reading and evaluation of papers submitted for a conference are captured and classified by `Reviewer`. The role as `Reviewer` may be a general term covering both `Program-Chair` and `Program-Committee-Member`. The role as `Participant` may be composed from `Traveler`, `Hotel-Guest` etc.

A graphical notation for use in object-oriented analysis and design must support roles as characterized by C1 - C6.

3 Role Simulation

Before introducing the role as a concept as such we shall discuss how roles can be simulated by existing concepts, namely by specialization, aggregation, and

association. We shall also discuss the obvious problems with theses approaches in relation to the characteristics C1 - C6. A summary of the problems with these simulations is illustrated in Table 1.

	Specialization	Aggregation	Association
Visibility	–	(+)	(–)
Dependency	+	–	(+)
Identity	+	–	–
Dynamicity	(–)	(+)	+
Multiplicity	–	+	+
Abstractivity	(+)	+	+

Table 1: Summary of Problems with Role Simulation

Using Specialization. We illustrate the simulation of roles by specialization as follows: `Conference-Associate` as a specialization of `OO-Associate`, — `Participant`, `Author`, and `Reviewer` as specializations of `Conference-Associate`, — and finally, `Speaker`, `Panelist`, and `Session-Chair` as specializations of `Participant`.

In the simulation of roles through specialization we have to model all the possible role combinations, usually done by the use of a multiple inheritance [1]. The problems in simulating roles through specialization are:

(1) The visibility is defined to include all the methods of the object in the role as well. The methods as `Speaker` are also visible when a `Participant` is only known as a `Panelist`.

(2) The dynamicity cannot be obtained because the roles become fixed (and dynamic change of classification is not possible). A `Participant` cannot stop being a `Session-Chair`.

(3) The multiplicity cannot be obtained because specialization does not include support of multiple instances of a super class [2]. Several occurrences of `Conference-Associate` cannot exist at the same time.

(4) The abstractivity is partially violated because the class of the object becomes a generalization of any of its roles. `Conference-Associate` is a generalization of `Reviewer`, which is not always appropriate.

[1] If singular objects are supported we need not describe all these combinations as classes, as the combinations can be given when creating the object, — and possibly also if the object changes its classification dynamically.

[2] Repeated inheritance and non-virtual classes as known from Eiffel and C++ are not seen as specialization mechanisms, but examples of using inheritance for other purposes.

An obvious use of specialization is `scientific` and `Staff` as specializations of `OO-Associate`, — and `Student-Volunteer` as a specialization of `Staff`. See the illustration in the appendix.

Using Aggregation. We illustrate the simulation of roles by aggregation as follows: `OO-Associate` as an aggregation of a number of `Conference-Associate`s, — `Conference-Associate` as an aggregation of one instance of `Participant` and possibly a number of `Author`s and `Reviewer`s, — and finally, `Participant` as a possible aggregation of a number of `Speaker`s, `Panelist`s, and `Session-Chair`s.

The problems in simulating roles through aggregation are:

(1) The visibility cannot be obtained directly, because the methods of the role are hidden by being a part object. The methods of `Speaker` are not visible of a `Participant`. In some forms of aggregation some methods of a part object may be "lifted" to become methods of the whole object as well [3].

(2) The dependency is not obtained because the methods of part-objects cannot depend on methods of the whole-object. The methods of `Speaker` — such as `Schedule` — cannot be defined in terms of the methods (for example also `Schedule`) of `Participant`.

(3) The role obtains an identity of its own by being a part object. A `Speaker` object is no longer dependent on the `Participant` object, and can exist independently of this.

(4) The dynamicity is not directly obtainable, because it is still uncertain how part objects can be exchanged dynamically in the whole object [4]. A `Participant` cannot stop being a `Session-Chair` and become a `Speaker` and finally stop attending a conference.

An obvious use of aggregation is `Program` as an aggregation of a number of `Activity`s (which may be specialized to either `Technical-Activity` or `Social-Activity`). The `Technical-Activity` is an aggregation of a number of `Session`s (which may be specialized to either `Presentation`, `Panel-Session` or `Invited-Talk`). The `Presentation` is an aggregation of a number of `Paper-Presentation`s. See the illustration in the appendix.

Using Association. We illustrate `OO-Associate` as an association [Rumbaugh 87] between a number of

[3] Such methods model hereditary properties in the aggregation process for concepts [Kristensen & Østerbye 94].

[4] If the dynamicity is obtained by references to self contained part objects the identity characteristic is not obtained.

Conference-Associates, — Conference-Associate as an association between one instance of Participant and a number of Authors and Reviewers, — and finally Participant as an association between a number of Speakers, Panelists, and Session-Chairs.

The problems in simulating roles through association are:

> (1) The visibility is not directly supported, and we need to be able to forward access to the associated objects from the association object (some times supported by a delegation mechanism). A method of Reviewer is not visible from outside given a Conference-Association link (object). The Conference-Association link must make such methods visible for example by forwarding to Reviewer.
>
> (2) The identity characteristic is not obtainable, because the roles and the object are self-contained objects. Reviewer, Participant, and Speaker are all individual objects, only related by associations.
>
> (3) Additional undesirable dependency is obtained because the role is visible from the object, and not only vice versa. The methods of Reviewer are visible from Conference-Associate, — which is not desirable.

An obvious use of association is Paper-Submission as an association between a Paper, its Author roles, its Reviewer roles, and Paper-Presentation. (Paper models the general information available for papers submitted for the conference). See the illustration in the appendix.

4 The Role Concept

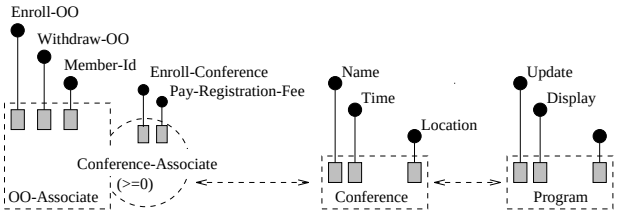


Figure 3: The Role Conference-Associate

In Figure 3, we illustrate Conference-Associate as a role of class OO-Associate in relation to Conference. We consider Conference-Associate to be a role of OO-Associate because a given conference is related to a specific role of an OO-Associate only and not the entire OO-Associate. An OO-Associate object may have any number (the notation (>=0) states the multiplicity) of Conference-Associate roles, because it may be involved in a number of conferences at the same time and this number may be changing over time. For OO-Associate we have methods such as Enroll-OO, Withdraw-OO, Member-Id etc., and for Conference-Associate methods such as Enroll-Conference, Pay-Registration-Fee, etc. In the method Enroll-Conference (of Conference-Associate) the method Member-Id (of OO-Associate) may be utilized in its definition, but not vice versa. Conference may have methods such as Location, Time, Name, etc. and Program may have methods such as Update, Display, etc.

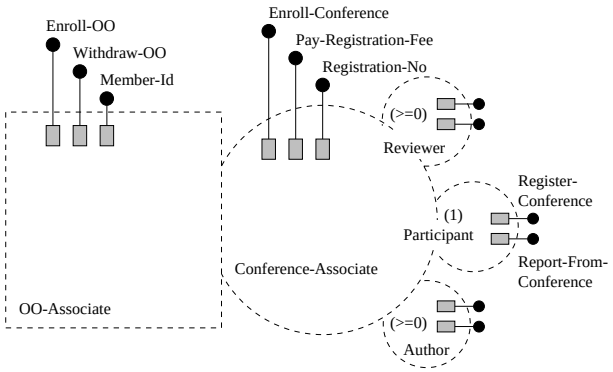


Figure 4: Roles of Conference-Associate

Roles of Roles. When a Conference-Associate is involved in the preparation and/or holding of a given conference this may include several roles, for example as a Participant, an Author, and a Reviewer. These roles then become roles of Conference-Associate — roles of a role — as illustrated in Figure 4. The Participant role may have methods such as Register-Conference and Report-From-Conference. The method Register-Conference may utilize the methods Registration-No and Member-Id in its definition, but not vice versa.

We consider Participant, Author, and Reviewer to be roles of Conference-Associate because the various relations to a given conference are related to these specific roles only (and not to Conference-Associate as a whole): A Participant is related to the Conference; an Author is related to the Paper, and a Reviewer is related to a Paper (one for each paper authored and for each paper reviewed). For a given conference there can be one Participant role only but several Author and Reviewer roles, and the number of these may vary in time.

Alternatively, we could model Participant, Author, and Reviewer as roles of OO-Associate directly and then omit Conference-Associate. However, by the chosen approach we manage to organize all the roles for a given conference together as a role with its "role

roles". As regards the alternative we would have a problem with specifying that for a given conference there is at most one `Participant` role related.

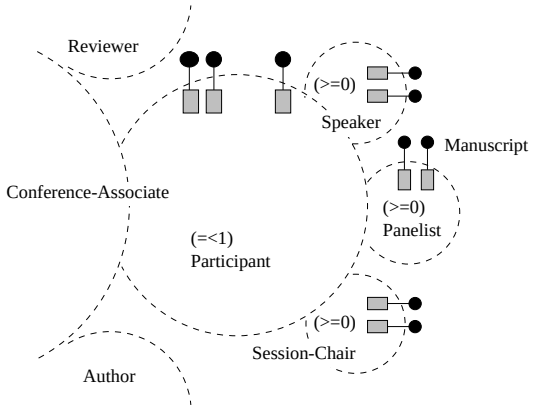


Figure 5: Roles of `Participant`

The actual participation in the holding of a given conference may either be as an "ordinary" participant or it may include several special roles, such as a `Speaker`, `Panelist`, and `Session-Chair`. We model these roles as roles of `Participant` (itself a role of another role) as illustrated in In Figure 5. The role `Panelist` may have a method `Manuscript` to model his/her opening presentation for the panel discussion.

We consider `Speaker`, `Panelist`, and `Session-Chair` to be roles of (the role) `Participant` because the various relations to a given conference are related to these specific roles only (and not to the `Participant` as a whole): A `Speaker` is related to the `Session`, a `Panelist` is related to the `Panel-Session`, and a `Session-Chair` is related to a `Session` (one role for each session to which a participant is related). For a given conference and a participant there may be several `Speaker`, `Panelist`, and `Session-Chair` roles, and the number of these may vary in time.

5 Role Abstraction

The abstraction processes *classification*, *specialization*, and *aggregation* are available for roles. The specialization and aggregation processes introduce relations between the methods of the roles [5]. The relations define the availability of the methods.

The availability of the methods of subjects depends on the classification of the subject as illustrated in Figure 6. The class `C` has the method `y` and the role `R` with method `x`. An instance of `C` with a role instance of `R` may be classified as either `C` or `R`. When classified as `C` the method `y`, but not the method `x`, is available.

[5] Based on the theory of *properties* of the *concepts* and *phenomena* in conceptual modeling [Kristensen & Østerbye 94].

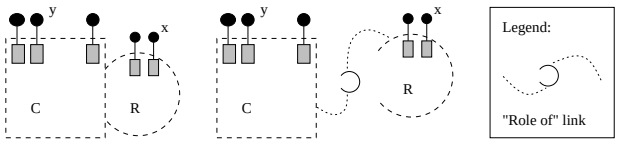


Figure 6: Availability of Methods

When classified as `R` both the methods `y` and `x` are available. An alternative notation for relating a role and its class/role by means of a "role of class/role link" is also illustrated in Figure 6 [6].

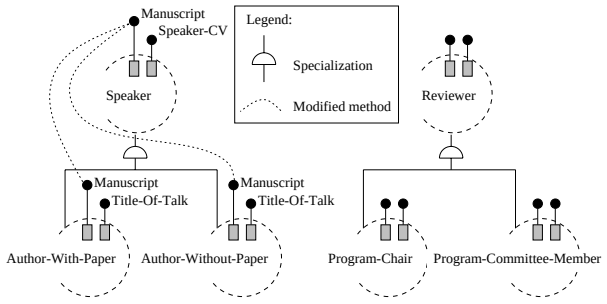


Figure 7: Specializations of `Speaker` and `Reviewer`

Specialization of Roles. A `Speaker` may either have submitted an accepted paper or be an invited speaker without any paper. We model this by specializations of the role `Speaker`, namely `Author-With-Paper` and `Author-Without-Paper`. Specializations of roles remain to be roles. This means that `Author-With-Paper` and `Author-Without-Paper` are roles for `Participant`, — and not classes. A `Reviewer` may either be the `Program-Chairman` or an ordinary member of the program-committee. We model this as specializations of the role `Reviewer` as `Program-Chairman` and `Program-Committee-Member`. In Figure 7, we illustrate the specializations of `Speaker` and `Reviewer`. For `Speaker` we have included the methods `Speaker-CV` to model the information to be given when the `Speaker` is announced by the `Session-Chair`. `Speaker-CV` is an *inherited* method for `Author-With-Paper` and `Author-Without-Paper`. The `Manuscript` of `Speaker` is a *modified* method. We imagine this to be some outline that is refined differently in

[6] By introducing the notation for a link to relate a role to its class/role we have the possibility that the role is a role for several classes/roles — instead of just one of these, as it has been the assumption until now. An argument against this possibility is that this may just be bad design, because the role is a role for some super class of the classes (same possibility for roles). The notation may still be valuable because the super class is then only implicitly given (must at least have a subset of the methods that the classes have in common, — the subset that is used from the methods of the role).

the two role specializations. We model `Title-Of-Talk` as an *added* method. For `Author-With-Paper` this method will probably be implemented by means of the `Paper` associated somehow with this kind of `Speaker`, and for `Author-Without-Paper` it will be defined directly.

Classes with roles may be specialized as usual. A role for a class is also a role for any subclass of the class, but in general not vice versa. Roles may also be specialized to roles, *sub-roles*. A sub-role is a role for a class if any of its super-roles is a role for that class.

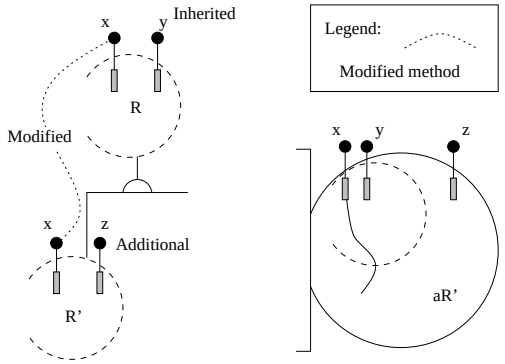


Figure 8: Specialization: Relations between Methods

The relations between the methods of a role and a specialization of the role are illustrated in Figure 8. The role `R` is specialized to `R'`, where the relations among the methods are: `y` is inherited, `z` is added, and `x` is modified. The methods `x`, `y`, and `z` are all available when classified with `R'`, whereas only `x` and `y` are available for `R`. For both `R` and `R'` the method `x` is a modified combination of the contributions for this method from `R` and `R'`.

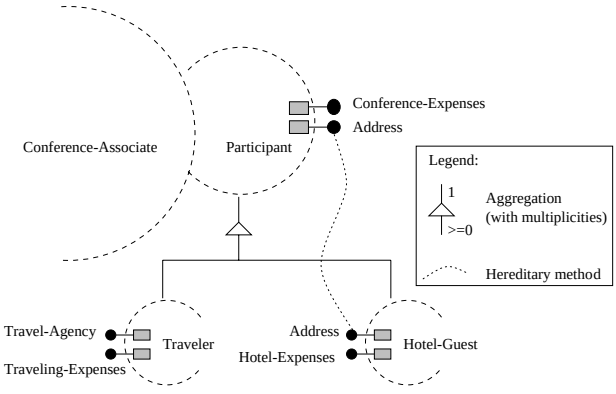


Figure 9: Aggregations of `Participant`

Aggregation of Roles. The `Participant` role of `Conference-Associate` may be aggregated from other roles of `Conference-Associate`, for instance part-roles like `Traveler`, `Hotel-Guest` and `Tourist`. As roles of `Conference-Associate` they all supply various additional methods, which may be applied to form the methods of the whole-role `Participant`. In Figure 9, we illustrate the aggregation of the role `Participant` from `Traveler` and `Hotel-Guest`. The method `Conference-Expenses` of `Participant` is an *emerging* method that includes all expenses from the part-roles such as `Traveling-Expenses` and `Hotel-Expenses`. The method `Travel-Agency` of `Traveler` is a *hidden* method not available for `Participant`, whereas the method `Address` of `Hotel-Guest` is an *hereditary* method and as such directly available also for `Participant`.

Classes with roles may be aggregated as usual from other classes possibly with roles. In general there are no special relations between the roles of part- and whole-objects. Roles may also be aggregated from other roles, *part-roles*. The whole-role and part-roles must be roles for the same class and the role instances must be allocated for the same object of this class [7]. One of the differences between a role of a role and a part-role (of a whole-role) is that the methods of a part-role can be utilized in the whole-role, whereas the roles of a role can utilize the methods of the role (and in both cases not vice versa).

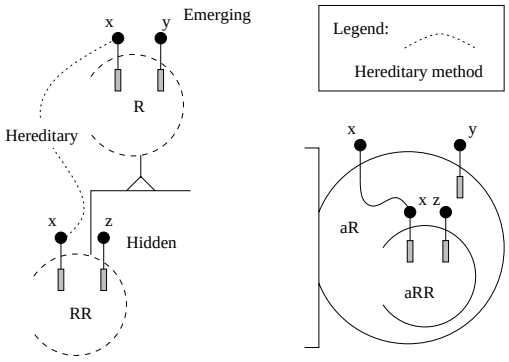


Figure 10: Aggregation: Relations between Methods

The relations between the methods of a whole-role and the methods of a part-role are illustrated in Figure 10. The role `R` is aggregated from (among others) the role `RR`, where the relations among the methods are: `y` is emerging, `z` is hidden, and `x` is hereditary. The methods `x` and `y` are available when classified with `R`, whereas `x` and `z` are available for `RR`. For `R` and `RR` the `x` is the same method (same definition).

[7] An aggregated part-role is a part of the whole-role, but may have hereditary methods, that are also methods of the whole-role. A part-role plays a "dual role". It is an addition to something and by this it gives a specific perspective on this something. At the same time it is a building block of the whole-role, so that Its methods may be used to form emerging methods for the whole-role.

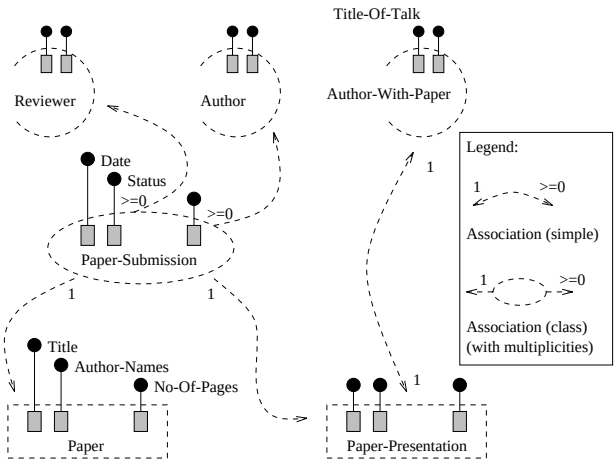


Figure 11: The Association `Paper-Submission` etc.

Associations To Roles. We model `Paper-Submission` as an association between `Paper`, `Author`, `Reviewer`, and `Paper-Presentation`. In Figure 11, we illustrate `Paper-Submission` with methods `Date` and `Status`, and `Paper` with methods `Title`, `Author-Names`, and `No-Of-Pages`. Another association is between `Author-With-Paper` and `Paper-Presentation`. From the method `Title-Of-Talk` of `Author-With-Paper`, through `Paper-Presentation`, through `Paper-Submission`, the `Title` of `Paper` is available. It is outside the scope of this paper to discuss association classes/objects and the accessibility and interaction supported by these [8].

The figure in the appendix illustrates the interplay of roles, associations, aggregation and specialization in a realistic modeling situation. It further illustrates the practical correlation between associations and roles. An association instance will relate a specific role instance of `Speaker` and a `Session` object. The `Session` object holds information that is specific to the individual session. The role instance will contain information about the particular `Speaker`, — exactly the information about the `Participant` relevant from the `Speaker` perspective (and in case the `Participant` is a `Speaker` in several `Sessions` exactly the information relevant for this particular `Session`).

6 Behavior

The `OO-Associate` may have a method, `Phone`, modeling his phone number at work. If someone, who knows a person only as `OO-Associate`, (or only as `Conference-Associate` because `Phone` is available from there also) invokes the `Phone` he/she will get exactly this number. As a `Participant` the method `Phone` may be defined to give for example the phone number of the hotel at which the `Participant` stays while attending the conference. If someone, who knows a `OO-Associate` (or a `Conference-Associate`) only as `Participant`, invokes the `Phone` he/she will get the phone number of the hotel.

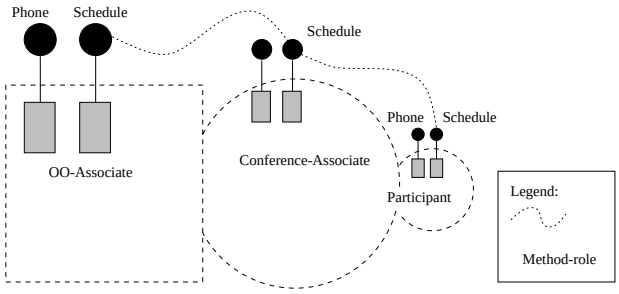


Figure 12: Method of Roles: `Phone` and `Schedule`

Method Roles. The class `OO-Associate` may have a method `Schedule` which will give the time schedule. When describing a role `Conference-Associate` for `OO-Associate` we may describe an addition to the method `Schedule`. This addition to `Schedule` is a role for that method (for simplicity we assume that it will have the same name but in general it may be any name). An invocation of the `Schedule` method of the `Conference-Associate` will give a combination of the time schedules registered for `Conference-Associate` and `OO-Associate`. This kind of role for a method is named a *method-role* [9]. The `Schedule` method of `Conference-Associate` may itself have a method-role described in `Participant`. In Figure 12, we link [10] the method `Schedule` of `OO-Associate` to its method-roles in `Conference-Associate` and `Participant`.

The relations between the methods of an intrinsic object and the methods of a role are *intrinsic*, *extrinsic*, and *method-role* as illustrated in Figure 13. The class/role `CR` has the role `R`, where the relations among the methods are: `y` is intrinsic, `z` is extrinsic, and `x'` is a method-role for `x`. The methods `x` and `y` are available when classified with `CR`, whereas `x'` and `z` are available for `R`. For `R` the `x'` is the combination of the contributions for this method from `CR` and `R`, whereas for `CR` the `x` is the contribution from `CR` only. The names `x` and `x'` can be identical or different.

[8] Here associations relate classes and/or roles. However, when associations are seen as classes/objects we may introduce roles even for associations themselves. This possibility is not discussed further in the paper.

[9] Various possibilities for the combination of methods are known from the inheritance mechanisms in existing object-oriented programming languages. Combinations of method-roles are briefly discussed in [Kristensen & Østerbye 95].

[10] An alternative notation where the handle of a method-role is glued onto the handle of its method is given in [Kristensen & Østerbye 95].

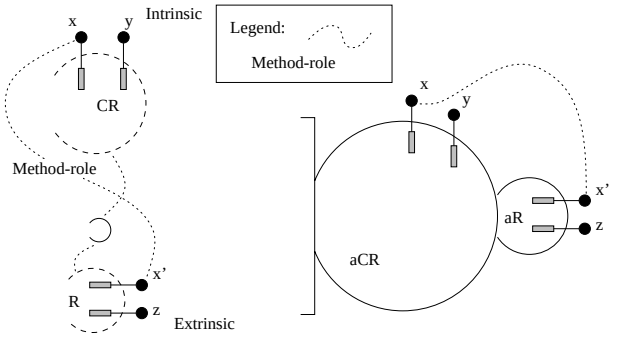


Figure 13: Roles: Relations between Methods

Subject Abstraction. A subject is an intrinsic object with its role instances. By abstracting on an intrinsic object and role instances we obtain a subject description. As an example we may abstract a `Ever-Reviewer` from class `OO-Associate` with the role `Conference-Associate` with the role `Reviewer`. Dependent on the multiplicities this subject may give instances that are for ever functioning as a `Reviewer` for a `Conference`. Consequently, we distinguish between subject (descriptions) and instances of these, subject instances.

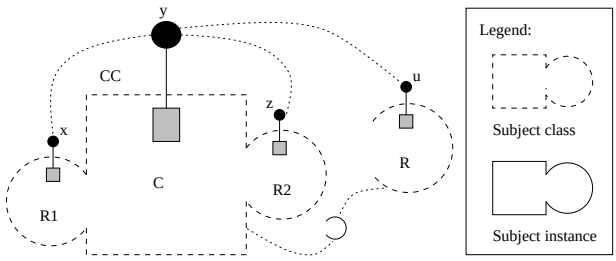


Figure 14: Subject Abstraction and Method Roles

Figure 14 is an illustration of subjects as abstractions. From the class `C` and the roles `R1` and `R2` we abstract the subject `CC`. A subject is illustrated graphically with no border between the box and the semi circle. An instance of `CC` will have an intrinsic object of `C`, one role instance of `R1`, and one of `R2`. This subject instance of `CC` does not limit itself to having this form but may take other additional role instances. Furthermore, subject abstraction is also possible for roles of roles.

When we abstract a subject we require that there are no naming conflicts among the methods of its constituents (`C`, `R1`, and `R2`). If a name conflict exists we may resolve the conflict by classifying the subject as one of its constituent class or roles (as respectively `C`, `R1`, or `R2`) and choose the method from this constituent in this way. Method roles — with identical or different names — do not cause name conflicts in subject abstraction. When classified by the subject a method-role is the combination of the method of the intrinsic object and all the method-roles in the subject. In Figure 14 the method `y`, classified by the subject abstraction `CC`, is the combination of `y` and `x` and `z` (all from the subject `CC`), but not method `u` because role `R` is not included in subject `CC`.

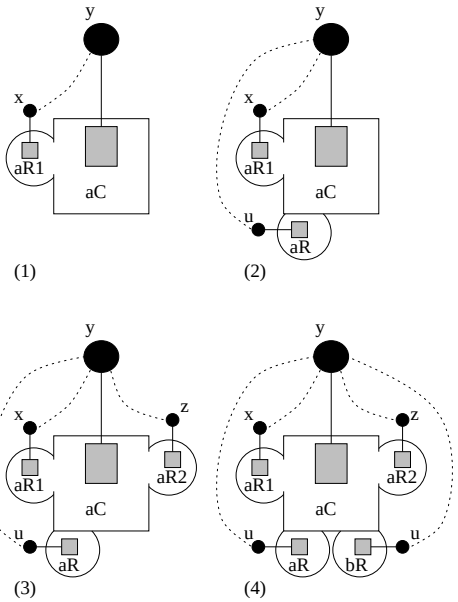


Figure 15: Subjects & Dynamic Role Allocation

Figure 15 illustrates a dynamic development of an object and its role instances given the definitions from Figure 14. The object `aC` takes the role instances `aR1`, `aR`, `aR2`, and `bR`, in this order, corresponding to the parts (1), (2), (3), and (4) of the figure. In (1) and (2) the method `y` for `CC` and the method `x` for `R1` both means `y` combined with `x`. In (3) and (4) the method `x` for `R1` is `y` combined with `x`. In (3) and (4) the method `y` for `CC` is `y` combined with `x` and `z`. In (2), (3), and (4) the method `u` for `R` is `y` combined with `u` (the `u` method of `aR`).

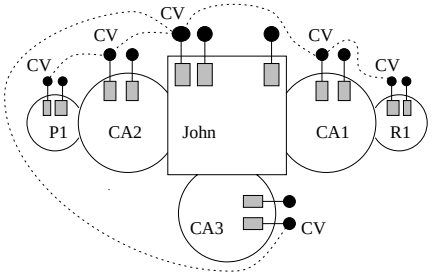


Figure 16: `SUBJECT` Classification: The Method `CV`

When no explicit subject abstraction is specified for a current combination of intrinsic objects and role instances we may obtain an implicit classification of the complete subject as it appears, by classifying `aC` as `SUBJECT`. The method `y` with this classification is `y` combined with `x`, `z`, and both the `u`'s. Assume that the method `CV` is intended to display a complete update of personal and professional data for an `OO-Associate` with all his/her roles. We assume that all earlier accumulated information is available from the `CV` of `OO-Associate`, whereas all the new information is available in the `CV` method-roles of the roles of `OO-Associate`, as illustrated in Figure 16. No matter how an `OO-Associate` instance appears with various roles, whenever we classify this as `SUBJECT`, the specified combination of the accumulated information and all new information is obtained by the `CV` method.

7 Locality

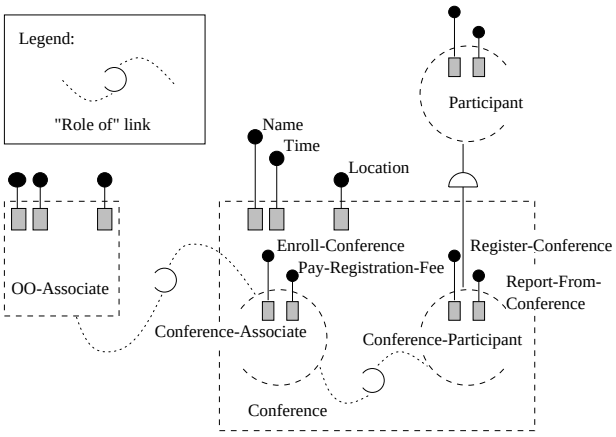


Figure 17: Localization of `Conference-Participant`

The role `Conference-Associate` is meaningful only in the context of a `Conference`. We may therefore apply the locality concept [Madsen & Møller-Pedersen 92] to support this relation and define `Conference-Associate` local to `Conference`. Furthermore, the `Participant` role is a general class, whereas a specialization of this, `Conference-Participant`, is meaningful only in the context of a `Conference`. Therefore `Participant` is described independently of `Conference` but specialized to `Conference-Participant` locally to `Conference` so that refined and added methods may utilize the direct access to the methods of `Conference`. The availability of methods is supported by the concept of locality: The methods of the enclosing class directly available. In Figure 17, we illustrate the `Conference-Associate` and `Conference-Participant` roles as defined locally to `Conference`. Graphically this is indicated by drawing the local figure inside the enclosing figure. The methods `Enroll-Conference` of `Conference-Associate` and `Report-From-Conference` of `Conference-Participant` may then use the methods of `Conference`, such as `Name` and `Time` directly. The dependability among objects is supported by the concept of locality: An object of the enclosing class will always exist for instances of the local classes/roles. The role `Conference-Associate` is dependent on and can always assume the existence of the instance of `Conference` in relation to which it has been instantiated.

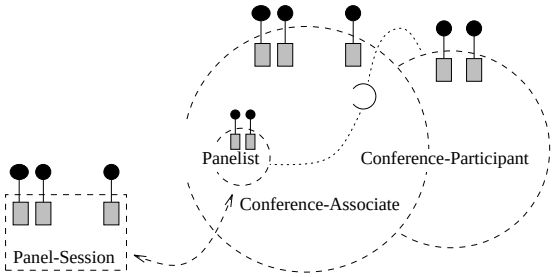


Figure 18: Role Local to Role

Roles may be meaningful only in the context of another role. Therefore, the locality of roles means that roles can not only be local to classes but also to other roles. The meaning of locality in terms of availability and dependability is unchanged. We see no natural use of roles local to roles in the conference example. However to illustrate the possibility we illustrate in Figure 18 that the role `Panelist` of `Conference-Participant` can be defined local to `Conference-Associate`. By this organization `Panelist` will have direct access to the methods of `Conference-Associate` and `Conference` (but this could also be achieved just by defining `Panelist` local to `Conference`).

8 The Dynamics of Roles

In order to describe the dynamic organization of roles we need additional graphical notation. The notation is separate from the notation for the static aspects of roles, and may be seen as an extension of usual regular expressions [11]. A regular expression may define legal sequences of atomic entities. In our case the atomic entities are role names and an occurrence of a role name indicates that an instance of this role will exist for some period of time. Because several role instances may exist at the same point in time we need to introduce the possibility of overlapping role names in the legal sequences.

[11] As it is the case for usual regular expressions our notation also allows for ambiguous descriptions, in the sense that some legal sequence of role occurrences may be interpreted in several ways according to a given description in the notation.

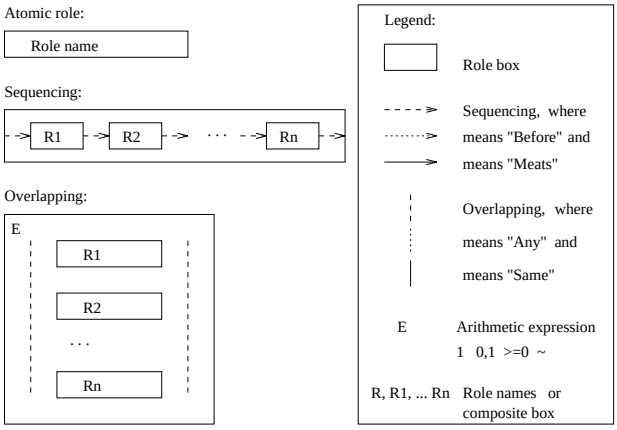


Figure 19: Basic Notation for the Dynamics of Roles

In Figure 19, we show the basic notation for the description of the dynamics of roles. A diagram defines the dynamic behavior of the roles for some class (or another role). Only such roles can be included in the diagram. The notation is interpreted as follows:

- The *atomic* box has a role name in it and means that an instance of this role will exist. The result of such a box is this role instance. An instance of any specialized role of the role name may replace the instance of the role. An atomic box without a role name means "no role".

- The *sequencing* box means that the role instances (as defined according to the boxes `R1 ... Rn` in the sequencing box) will exist in the given sequence. The arrow between two boxes indicates whether there will be a gap between the results of the boxes or these will meet. The result is the given sequence.

- The *overlapping* box means that the role instances (as defined according to the boxes `R1 ... Rn` in the overlapping box) will exist in some overlapping form. The lines before and after the boxes indicates whether the results of the boxes may start and/or end at the same or different point in time. The result is the overlapping form. The expression `E` may put additional limitations on the overlapping form.

- The *iteration* box and the *duplication* box define special cases of sequencing and overlapping with the same box, `R`. The expression `E` describes limitations on the number of `R`'s, and the arrow and the line has the same meaning as in the sequencing and overlapping boxes, respectively.

- Any composite box may be given a name. This name may also be used in an atomic box. The meaning of such an atomic box is the meaning of the composite box. Recursion in the named composite boxes is not allowed.

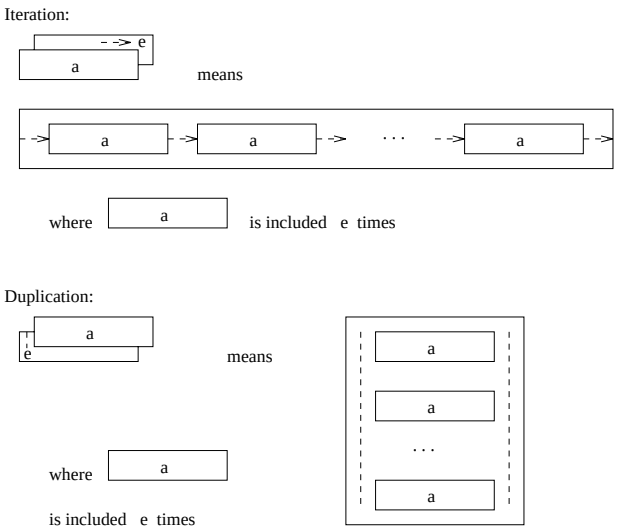


Figure 20: Notation: Iteration and Duplication Boxes

In Figure 20, we illustrate how the notation for iteration and duplication can be defined informally by sequencing and overlapping boxes, respectively.

The notation for the dynamics of roles extends easily to roles of roles. The dynamics of the roles of a role is simply described separately. When a role stops to exist so will all its roles. The possible localization of a role will not influence the description of the dynamics of these.

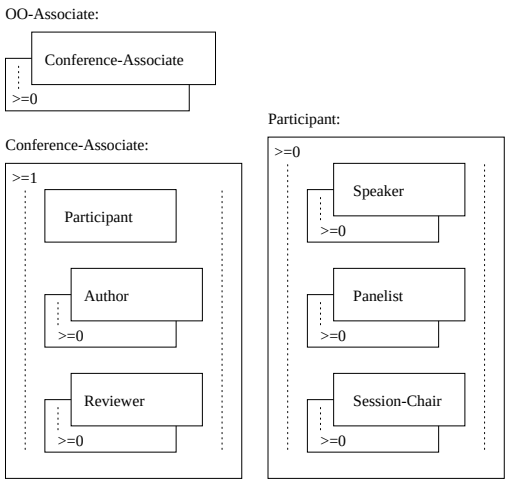


Figure 21: The Dynamics in the Example

In Figure 21, we illustrate the dynamics of the roles of `OO-Associate`, `Conference-Associate`, and `Participant`. An `OO-Associate` may have any number of overlapping `Conference-Associate` roles al-

located, and these may start and end in any order. A `Conference-Associate` may have at most one `Participant` role, any number of `Author` roles, and any number of `Reviewer` roles allocated. These are overlapping and may start and end in any order. A `Participant` may have any number of `Speaker`, `Panelist`, and `Session-Chair` roles allocated, — overlapping, and starting and ending in any order.

9 Experiments

Two series of experiments in programming language support of role and association classes have been conducted [Tinggård & Worm 95] and [Jensen & Jørgensen 95]. The objective was to gain more experience with the design of abstraction mechanisms of this kind, to consider efficient implementation techniques, and to be able to use the language mechanisms and the implementation for reasonably realistic test cases. The experience from the test cases is that role and association classes are straightforward to use and appear to give well-organized descriptions.

In [Jensen & Jørgensen 95] an existing object-oriented programming language, objective C, is extended with constructs for the support of roles. The extension includes (in some cases simplified variants of) dynamic creation, deletion, and transfer of role instances, roles of roles, specialization and aggregation of roles, and method roles. The experiment is organized as a series of minor experiments, where a following experiment extends — and builds on the implementation of — the previous one. Roles are simulated by classes and objects of the existing language. The extensions are implemented by a mixture of a preprocessing of some of the additional syntactical constructs and the addition of a special superclass for role classes.

In [Tinggård & Worm 95] a series of theoretical and practical experiments are conducted. An overall experiment investigates the conceptual abstraction processes, (classification, specialization and aggregation etc.) and their relation to concepts such as the association and the role. A pair of two steps experiments investigate basic and complex variants of associations and roles, respectively. In each case the nature of these concepts is analyzed and important observations of these are described. The resulting clarification of these concepts forms an excellent general foundation for the design of specific language constructs for these concepts for an existing or new programming language. Each experiment includes a practical implementation (a simulation by classes and objects in Smalltalk) of a selected part of the concepts.

10 Summary

Results. The main results are summarized as:

- A motivation of the use of roles as abstractions in object-oriented modeling, and the problems with simulating this abstraction by means of specialization, aggregation, and association.
- A graphical notation for roles to be used in object-oriented analysis and design, which supports the description of both static and dynamic aspects.
- The presentation of roles as specialized abstractions that can be supported by means of generalization hierarchies, part-whole hierarchies, association organizations, and that generalizes to roles for roles, roles for methods, and the notion of localization.
- A comprehensive example, inspired from the "Conference Organizing Problem", that illustrates the expressiveness of the role concept.

Related Work.
Objects with Roles. In [Pernici 90] a model is described for object behavior based on the concept of role. The model is similar to the basics in our model in the sense that an object can play different roles during its execution to receive and send different messages at different stages of evolution, etc. The focus is on the modeling of the object life-cycle with sequences of *abstract role-states*. State transitions are governed by *rules*. Object behavior is described separately in each role, and different behavior in different roles is related by specifying rules and constraints to govern concurrent behavior. The rules are essential in the model. They control the transitions between role-states and define which messages can be received and sent by the object in a role-state. They ensure integrity and they also control role creation and termination. Inherited roles are briefly described similar to inherited methods, and there is no further exploration of the notion of role, including roles of roles, role abstraction, method roles, etc.

Subjects. The *subjects* [Harrison & Ossher 93] model that different agents might view the same object from different perspectives. The agents do not only have a filtered view of an object, but some of the methods of the object may be there only because of the given perspectives of an agent. Usually, the criteria for deciding which properties of real-world phenomena to include in a model are based on a single perspective only.

Aspects. The *aspect* [Richardson & Schwarz 91] is like a role in the sense that it is dynamically allocatable and serves as dynamic classification of an object. The aspect is presented as a programming language

construct, and analysis or design issues are not discussed. The aspect does not "inherit" the methods of the intrinsic object, so that methods of the intrinsic object must explicitly be exported to the aspect. The use of aspects seems to be an improved kind of aggregation, where new parts can be added dynamically.

OORASS. OORASS [Reenskaug et al. 92] is an object-oriented methodology with emphasis on *role analysis and synthesis*. The method focuses on the creation of an organized structure of collaborating objects and its representation. The *role* concept represents the responsibilities of the object within the organized structure of collaborating objects. In order to exploit generally useful objects it is necessary to distinguish between an objects' capabilities and its position in the collaboration structure. This distinction is important because objects of the same type may be reused in many different positions. The role specifies which capabilities are needed, and may as such appear in different positions. Role modeling consists of two subparts: Analysis for breaking the total problem into subproblems and for creating a model for each subproblem, — and synthesis for combining smaller models into larger ones. Roles appear in role diagrams, where the interaction is represented by contracts (groups of acceptable messages for roles). The roles model the responsibilities needed at a given position in the collaboration structure. The roles are then combined by various operations, — join, merge, and hide, to form the objects. A role is considered an aspect of an object, in the sense that it contains the references and action sequences of an object participating in a certain responsibility. In OORASS the objects possess their (role-)capabilities inherently through the synthesis process of roles.

MON. MON [Maughan & Durnota 94] is a graphical notation including roles. Roles are classes and they are related to the role-playing class by a role relationship. Roles inherit from their role-playing class. The role relationship can have assertions (preconditions, postconditions, invariants) annotated. The graphical notation allows a few simple constraints on the dynamic allocation of roles to objects: Roles are mutually exclusive, or that they *can* be taken simultaneously.

MOSES. MOSES [Renouf & Henderson-Sellers 94] is an object-oriented methodology that is extended to incorporate *object roles*. The need for dynamic (not addressed by most object-oriented programming languages) and multiple (where the support by means of multiple inheritance is a solution only if there is no need for disjoint classification of objects) classification is discussed. Object roles are claimed to be a successful solution to the complexity of application domains found in business (and other) environments. Object roles are classes, and there is a special graphical notation for specifying that such a class is used as a role for some other class. The use of object roles is included in the methodology as an additional sub-activity. The proposal appears as an incomplete model that only supports partly inheritance among object roles and partly object roles for object roles themselves.

Design Patterns. We have introduced the role as a language mechanism to be used to create abstractions of varying definition. The role as a language mechanism is distinct from design patterns, which is an example of software architecture. A design pattern [Gamma et al. 94] can be seen as an organizational idiom, comprising an abstract code component. A pattern represents a general abstract solution to similar recurring problems. Design patterns like *decorator* and *state* (from [Gamma et al. 94]) simulate some of the properties of roles introduced in this paper. Because an abstraction mechanism — like the one we introduce for roles — is a part of a language it has fundamental influence on how we form our initial understanding of the world around us, and how we express this understanding. In general abstraction mechanisms may support the abstraction processes specialization and aggregation, and the role abstraction mechanism is no exception in this respect. Abstraction mechanisms also support the direct and improved description of design components, such as for example design patterns. Design patterns may be more advanced and comprehensive, and still abstract, reusable design components, but they are specific components and not general language mechanisms.

Challenges.

Naming of Roles. The duplication of the same role in subject abstraction must be supported by a naming facility for the role instances. The names are used both for role instance allocation and for resolving a classification, that gives several role instances as the result. A preliminary proposal is given in [Kristensen & Østerbye 95].

Roles of Active Objects. Active objects model phenomena, that are inherently active. The interaction between active objects may be coordinated by means of various forms of language constructs. A construct may support the synchronization of the execution of the life cycle of the object and method activation requests from other objects. Active objects can also take on roles. This models the situation when a phenomenon basically is active doing its life cycle, but once in a while combines this cycle with other activities, — described as roles also with life cycles. The life cycles of the intrinsic object and its role instances may either be executed concurrently or interleaved. A solution to the dynamicity of roles must also be provided.

Dynamics: Aggregation and Specialization. In aggregation the part-roles and the whole-role are

described independently in the diagrams, but in general these are dependent on each other. In static aggregation the whole-role can exist only if the part-roles exist. In dynamic aggregation the situation may be more complicated. In specialization the description of the dynamics of the roles for a class/role may be refined for a specialization of the class/role. Both more potential roles may exist for the specialized class/roles, and additional dynamics may be allowed. A simple, but insufficient solution is to give an alternative description for the specialized class/role, but to require that any legal sequence of the former description is also "sub"-sequence of some legal sequence of the alternative description.

Dynamicity in Aggregation. In principle the part-objects of the whole-object can be added and deleted dynamically. In practice this is usually only possible by a simulation by means of references to part-objects. Can the dynamic nature of roles compensate for the lack of dynamicity in aggregation? 1) Roles cannot support dynamic part-objects directly because of differences in the availability of methods and classification. 2) Whole-roles are aggregated by part-roles, — added and removed dynamically. Some kind of dynamicity in aggregation is achieved — but only as aggregation of roles by roles. 3) The part-objects in whole-objects can have roles, that are added and removed dynamically. Some kind of dynamicity in aggregation is achieved — but only as roles of aggregated part-objects.

Acknowledgments. We thank Kasper Østerbye for our inspiring joint work on the conceptual abstraction theory for roles as well as the actual design of specific language mechanisms. We also thank Morten Jensen, Michael Dorph Jørgensen, Carsten Tinggård and Torben Worm for fruitful discussions and new insight obtained in relation to the experimental projects.

References

[Gamma et al. 94] E.Gamma, R.Helm, R.Johnson, J.Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, 1994.

[Harrison & Ossher 93] W.Harrison, H.Ossher: Subject-Oriented Programming (A Critique of Pure Objects). Proceedings of the Object-Oriented Programming Systems, Languages and Applications Conference, 1993.

[Jensen & Jørgensen 95] M.Jensen, M.D.Jørgensen: Roles – A Dynamic Alternative. Thesis (in Danish), Aalborg University, 1995.

[Kristensen & Østerbye 94] B.B.Kristensen, K.Østerbye: Conceptual Modeling and Programming Languages. Sigplan Notices, 29 (9), 1994.

[Kristensen & Østerbye 95] B.B.Kristensen, K.Østerbye: Roles: Conceptual Abstraction Theory & Practical Language Issues. Department of Computer Science, Aalborg University, 1995.

[Madsen & Møller-Pedersen 92] O.L.Madsen, B.Møller-Pedersen: Part Objects and Their Location. Proceedings of International Conference on Technology of Object-Oriented Languages and Systems (TOOLS EUROPE'92), 1992.

[Maughan & Durnota 94] G.Maughan, B.Durnota: MON: An Object Relationship Model Incorporating Roles, Classification, Publicity and Assertions. Proceedings of OOIS'94, International Conference on Object Oriented Information Systems, 1994.

[Olle et al. 82] T.W.Olle, A.A.Verrijn-Stuart, H.G.Sol, Eds.: Information System Design Methodologies: A Comparative Review. North-Holland, 1982.

[Pernici 90] B.Pernici: Objects with Roles. Proceedings ACM-IEEE Conference of Office Information Systems (COIS), 1990.

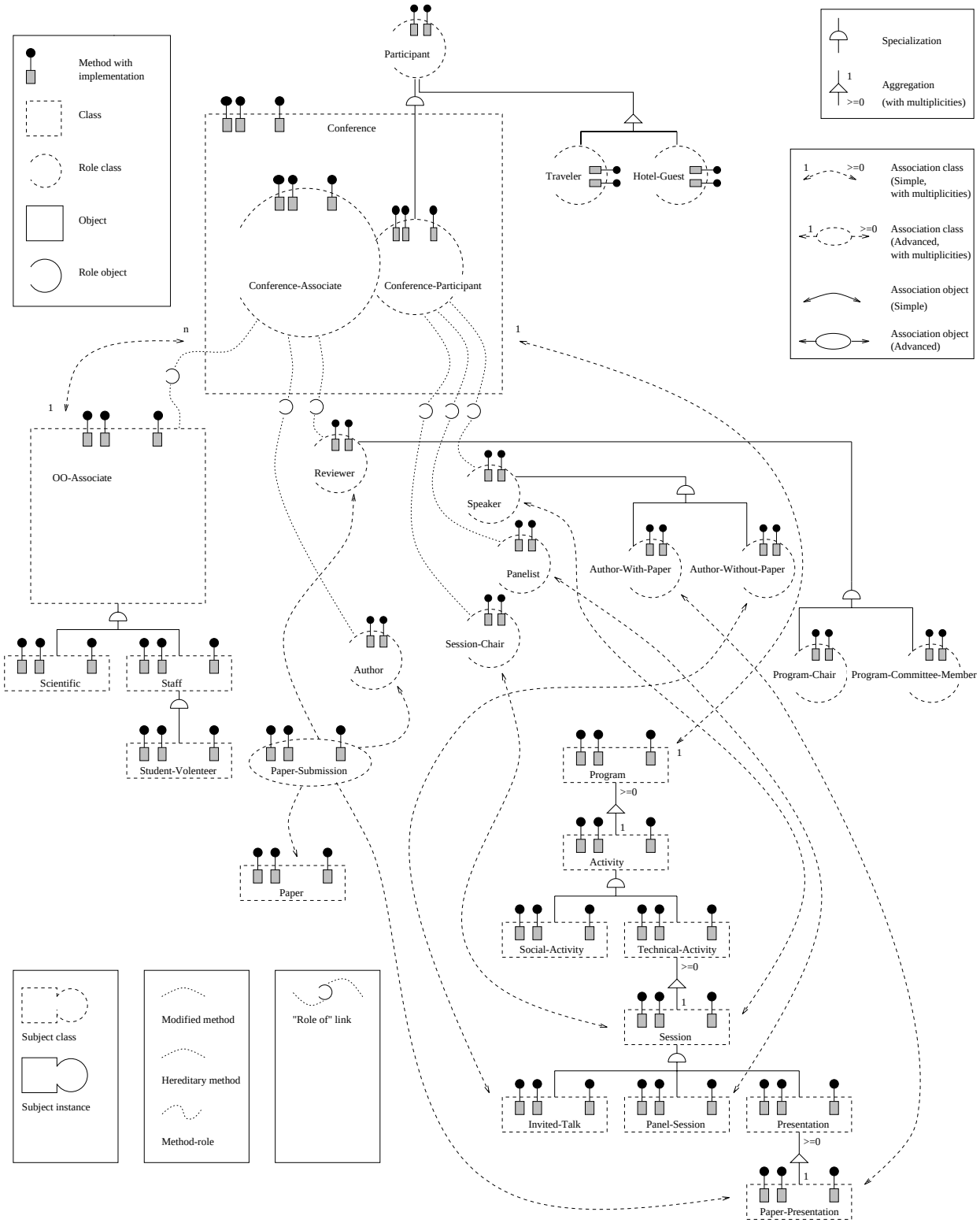
[Renouf & Henderson-Sellers 94] D.W.Renouf, B.Henderson-Sellers: Incorporating Roles into MOSES. Proceedings of International Conference on Technology of Object-Oriented Languages and Systems (TOOLS PACIFIC'94), 1994.

[Reenskaug et al. 92] T.Reenskaug, E.P.Andersen, A.J.Berre, A.Hurlen, A.Landmark, O.A.Lehne, E.Nordhagen, E.Ness-Ulseth, G.Oftedal, A.L.Skar, P.Stenslet: OORASS: Seamless Support for the Creation and Maintenance of Object Oriented Systems. Journal of Object-Oriented Programming, 1992.

[Richardson & Schwarz 91] J.Richardson, P.Schwarz: Aspects: Extending Objects to Support Multiple, Independent Roles. Proceedings of the 1991 ACM SIGMOD International Conference on the Management of Data, 1991.

[Rumbaugh 87] J.Rumbaugh: Relations as Semantic Constructs in an Object-Oriented Language. Proceedings of the Object-Oriented Programming Systems, Languages and Applications Conference, 1987.

[Tinggård & Worm 95] C.Tinggård, T.Worm: Modeling and Programming Languages. Thesis (in Danish), Aalborg University, 1995.


Method with implementation
Class
Role class
Object
Role object
Specialization
Aggregation (with multiplicities)
Association class (Simple, with multiplicities)
Association class (Advanced, with multiplicities)
Association object (Simple)
Association object (Advanced)
Participant
Conference
Traveler
Hotel-Guest
Conference-Associate
Conference-Participant
OO-Associate
Reviewer
Speaker
Panelist
Author-With-Paper
Author-Without-Paper
Session-Chair
Author
Scientific
Staff
Student-Volenteer
Paper-Submission
Paper
Program-Chair
Program-Committee-Member
Program
Activity
Social-Activity
Technical-Activity
Session
Invited-Talk
Panel-Session
Presentation
Paper-Presentation
Subject class
Subject instance
Modified method
Hereditary method
Method-role
"Role of" link