ASSOCIATIVE PROGRAMMING AND MODELING: ABSTRACTIONS OVER COLLABORATION

Bent Bruun Kristensen
Maersk Mc-Kinney Moller Institute, University of Southern Denmark, Denmark

Keywords: Collaboration, abstraction, modeling and programming, association, concurrent and interleaved execution, activity, role.

Abstract: Associations as abstractions over collaborations are motivated and explored. Associations are seen as first class concepts at both modeling and programming levels. Associations are seen as concepts/phenomena and possess properties. Various notations for collaboration in object-oriented programming and modeling are discussed and compared to associations. Concurrent and interleaved execution of objects is described in relation to associations.

1 INTRODUCTION

Description of collaboration between some participants is less supported by existing notations and diagrams. In collaboration the participants engage in the collaboration through specific roles and the actual interaction sequence between the participants follow some rules. Existing notation and diagrams are mainly based on object-centric modeling and programming exemplified by objects, object references and remote method invocation. Examples of non object-centric notations and diagrams include the relation (May et al, 2001). This relation is an example of an abstraction over structural aspects only—interaction aspects are not covered by relations. Associations are seen as an abstraction over both structural and interaction aspects of collaboration. The association supports description at both abstract modeling and concrete programming levels. At the abstract level associations are seen as concepts and phenomena characterized by their properties. At the programming level properties of associations are expressed through language mechanisms.

Associations are inspired from a conceptual model for understanding ambient systems (May et al, 2001). Such systems have a more dynamic situation with respect to collaboration among the entities in the system. In this model we imagine tangible objects existing in habitats and collaborating with other tangible objects—and tangible objects enter and leave habitats. As part of this dynamic picture tangible objects engage in collaboration with other tangible objects—simple or complex collaborations. The notion of associations is a means of capturing planned or spontaneous collaborations between tangible objects—to conceptually understand and prescribe collaboration as abstractions over collaboration. The illustration of an ambient system included as example in this article—"the conference organizing problem"—only includes aspects of collaboration but excludes aspects of user awareness and support through knowledge of time and place, augmentation of reality by additional views, and availability and interaction with software agents and physical robots (Kristensen, 2003).

Our approach is inspired by the evolution from traditional systems (often information systems) towards ambient systems including pervasive (Burkhardt et al, 2001) and ubiquitous (Weiser, 1991) systems. Ambient systems illustrate the change from development of systems towards an understanding where systems are grown through evolution. Associations are a move from object-centric technology towards non-centric technology. For associations we distinguish between an abstract, informal and conceptual modeling level and a concrete, formal and executable programming level.

Bruun Kristensen B. (2006).
ASSOCIATIVE PROGRAMMING AND MODELING: ABSTRACTIONS OVER COLLABORATION.
In *Proceedings of the First International Conference on Software and Data Technologies*, pages 54-62
DOI: 10.5220/0001308600540062

2 OBJECT-ORIENTATION

References support the relations between objects in object-oriented programming languages. In Figure 1 we illustrate the usual notions of class, object, reference and method invocation. Class `C` has method `mc`. Object `Oc` is an object of `C`. Class `D` has method `md` and a reference `Rc` qualified by `C`. Object `Od` is an object of `D` and reference `Rc` has the value `Oc`. Method `md` of `Od` can invoke method `mc` of `Oc` by `Rc.mc(…)`.

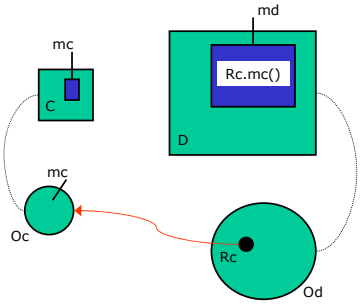

Figure 1: Class, object, reference and method invocation.

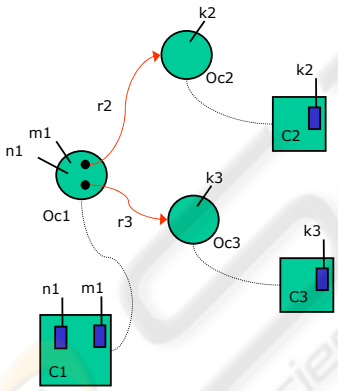

Figure 2: Illustration of object-oriented collaboration.

In a typical object-oriented collaboration as illustrated in Figure 2, method `m1` of class `C1` contains the following example—(this collaboration example is also used in subsections 2.2 (Figure 3) and 2.3 (Figure 4)):

```
x1 = r2.k2(…);
x2 = r2.k2(…);
y = n1(x1, x2);
r3.k3(y);
```

We observe the following characteristics of this schematic example from object-oriented programming:

- The reference is statically bound to the class (and any object of the class) whereas the value of the reference varies dynamically
- The reference is qualified by a class, which determines which types of objects may be referenced by the reference
- The reference is used for different purposes (invocations of different methods from different methods)
- The use of a reference for a given purpose is separated from the reference and distributed over several method bodies

2.1 Object-Oriented Delegation

The characteristics of collaboration through object-oriented delegation include that collaboration is explicitly described and implemented in a method in Figure 3 as `a1` of `A1`. Collaboration is initiated by `Oc1.m1(…)` by invoking `a1` of object `oA1`. This approach includes some typical problems:

- `Oc1` and `n1` are not necessarily known to `oA1` for the invocation `n1(x1, x2)` ("self" problem)
- `A1` may be parameterized by references `r2` and `r3` and with some *reverse* reference for `r`

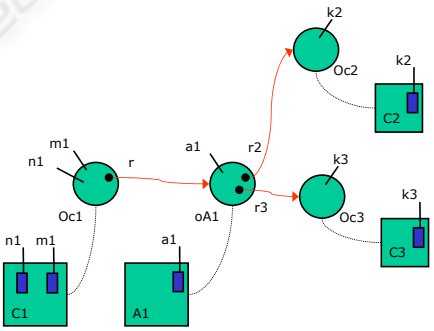

Figure 3: Illustration of object-oriented delegation.

2.2 Control Object/Method

The characteristics of collaboration through a control method/object include that collaboration is explicitly described and implemented in a method/object (objectification of collaboration but here only exemplified by methodification as `a1` of `A1` in Figure 4). Collaboration is initiated by `Oc1.m1(…)` by invoking `a1` of object `oA1`. This solution includes the following problems:

- The effect through `y` on `Oc1` must be a side-effect through invocation of `r1.n1`
- `A1` may be parameterized by references `r1`, `r2` and `r3`

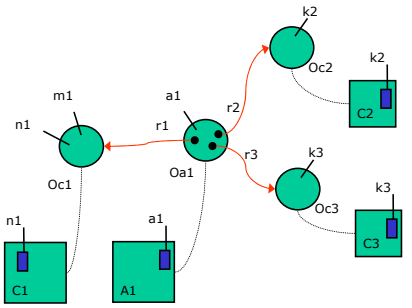

Figure 4: Illustration of object-oriented control method.

3 ASSOCIATIONS

Associations represent an alternative to object-centric modeling and programming. Our associations support not only structural relationship, but also collaboration between objects. An association is described as an abstraction, it may be instantiated, and it has identity. Dynamic changing associations are supported—descriptions may be added to executing systems and objects of these may associate participating objects of the executing system.

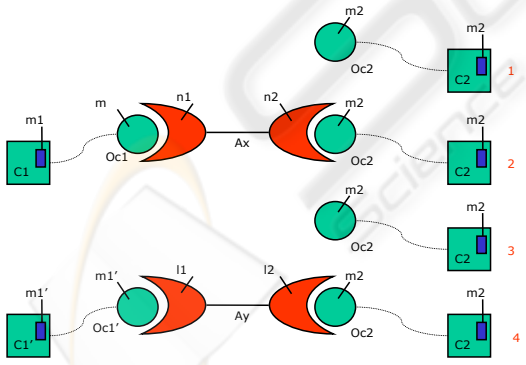

Figure 5: Snapshots of associations.

Figure 5 illustrates dynamic creation and deletion of objects of associations through four snapshots. In (2) an object `Ax` with roles with properties `n1` and `n2` is created. Object `Oc2` of class `C2` function as one participant in the association. In snapshot (1) no associations exist for `Oc2`. In (2) `Oc2` is associated by means of `Ax` with object `Oc1` of class `C1`. In (3) the association `Ax` no longer exists. In (4) `Oc2` is associated by means of an object `Ay` with roles with properties `l1` and `l2` with object `Oc1'` of class `C1'`.

In UML models the main concepts are captured through class diagrams supplied with relation/association classes as fundamental model structures. In addition these models include sequence and collaboration diagrams, where the interaction of objects is modeled in terms of method invocations. This description is separated from classes and associations, and neither sequence nor collaboration diagrams are conceptualized as abstractions over collaboration. Our notion of association is an abstraction over interaction and collaboration and the actual method invocations between objects are modeled as integrated elements of the association. In addition roles played by participating objects in an association are also modeled as extensions of the objects to participate in the association. Traditionally abstractions over certain aspects of an object—as for example the collaboration of the object with other objects or the objects' role towards other objects—are objectifications of such aspects. In this sense our notion of association is an integrated objectification of collaboration aspects and role aspects.

The association is seen as an abstraction during conceptual modelling (Madsen et al, 1993). In conceptual modeling different forms of abstraction in terms of concepts and phenomena are illustrated in (Kristensen, 2003): Classification (and exemplification) where a concept classifies a number of phenomena (which themselves exemplify the concept). Specialization (and generalization) where a more general concept generalizes a more specific concept (which itself specializes the general concept). Aggregation (and decomposition) where a whole concept describes the aggregated phenomenon of several part concepts (which themselves can be decomposed from the whole concept).

3.1 Example: `paper_review`

As an illustrating example, we examine the association of reviewing papers a conference—referred to as a `paper_review`. This association requires a certain degree of collaboration between those who are involved in it. For instance, an `author` will submit a `paper` for review, while the `chairman` will submit papers to `reviewers` who must report back. A directive describes how the association should be carried out. With the

paper_review, the directive might be carried out in distinct portions:

```
1.  prepare_paper_review
        author_submits_paper_to_chairman
        chairman_submits_papers_to_reviewers
        reviewers_submit_reports_to_chairman
2.  paper_selection
3.  chairman_informs_authors
```

The association paper_review is only one type of review that can take place. For example, a periodical_review is the review of a submitted article that takes place for a periodical; it is somewhat similar but involves an editor rather than a chairman and its selection process is different. Both paper_review and periodical_review are specialized types of review. The directive that specifies how paper_review should be carried out may also be seen as a specialization of a more general review directive:

```
1.  prepare_review
2.  carry_out_review
3.  complete_review
```

These portions correspond to (1), (2) and (3) above (which are more specialized). The participants of these associations may also be similarly classified. For instance, all review associations involve a coordinator and an author. Thus, in a paper_review, we can refine a coordinator to be a chairman—in a periodical_review, we can refine a coordinator to become an editor.

These different types of review associations may have similar methods. For example, producing a status_report (produce a listing of the current status of the ongoing reviewing process) is something that each review association must do—a paper_review will produce a specialized type of status_report, as will a periodical_review. Finally, such associations are constituted from smaller sets of associations. For example, within the paper_review association, there is a paper_selection association to choose acceptable papers.

3.2 Execution

In addition to action directives, associations include roles to be played by participants in the collaboration. Roles are abstractions in associations. Roles may specify additional methods or may extend existing methods of an object (Kristensen, 2003). In Figure 6 R1, R2 and R3 are roles of association class X. Ax illustrates an object of class X. Ax associates objects Oc1, Oc2 and Oc3 (each playing a role R1, R2 and R3) of classes C1, C2 and C3 respectively. The method k1 of R1 illustrates an additional method—alternatively k1 may be described as an extension of the existing method n1 of C1 (similar for methods k2 and k3).

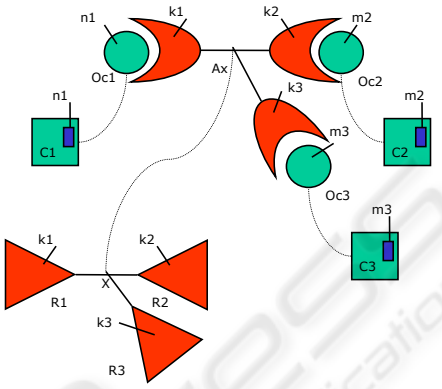

Figure 6: Illustration of association.

The directive of association X is executed by the respective owners of the actions among the participants of X. The notation R::k(…) means that the object playing role R executes its method k(…). Hence the collaboration is explicitly described through the directive of X e.g.:

```
x1 = R2::k2(…);
x2 = R2::k2(…);
y = R1::n1(x1, x2);
R3::k3(y);
```

An association is a description of a central abstract unit. The notation R::… is different from remote denotation, because "…" is situated in the context of R and interpreted in this context. The execution of its contribution from a directive is done by the participating object. In *sequential execution*, description and execution of sequencing are in terms of one execution thread only—a method in one object invokes a method of another object and one thread executes the entire invocation sequence. In *multi-sequential execution*, sequencing is described as several execution threads (one for each object) but is executed by one thread only. The thread switches (at language defined points) between the executions of the objects—this *interleaved* execution of the sequencing of objects means that only one object is executed at a given time.

4 CONCURRENCY

In general objects execute concurrently. Communication and synchronization constructs describe the interplay between such active objects. Objects have an individual action part—on instantiation, an object will immediately execute its action part and is inherently active. The description of the action part may involve the activation of methods in the object itself and (activation requests of) methods for other objects. Because the objects are active, the interaction between objects is usually coordinated by means of various forms of ego-centric language mechanisms for synchronization of the execution of the life cycle of the object and method requests of/from other objects. As an example, when one object attempts a method request of another object, then the first objects must wait until the other object explicitly accepts this invocation. When the invocation is accepted the objects are synchronized and the invocation can take place.

4.1 Interleaved Execution

In associations the collaboration (including communication and synchronization between the participating objects) is described in directives of associations only. Active objects are executed in parallel (and shared data resources are typically active objects to ensure exclusive access). The association directive itself supports various ways of describing the sequencing of the collaboration including sequential, repeated, parallel, interleaved, any order executions etc. The individual action part of an object only describes its individual life cycle, i.e. no form for interaction with other objects is included. The execution of the total life cycle of an object described through several contributions in directives (of current collaborations) is an interleaved execution of its contributing parts and also interleaved execution with its individual action part. Interleaved execution for one such object of several different parts means that (at language defined points in the parts) execution will switch between the parts.

Figure 7 illustrates the mechanisms introduced. Object `Ax` is of association class `X`. Object `Oc1` is a participant of class `C1` and `R1` denotes the role played by `Oc1`. The construct `R1::k1(…)` denotes a contribution to the directive of `Ax` from role `R1`. The object `Oc1` executes its individual action part (exemplified by "… `n1(…)` …") interleaved with the various contributions from role `R1` of directive `Ax` (and contributions from similar directives of associations in which `Oc1` currently plays roles).

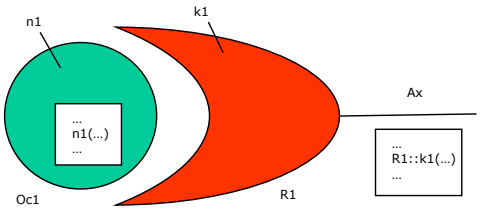

Figure 7: Interleaved execution of active objects.

4.2 Example: `paper_review`

In the `paper_review` example, a `reviewer` may be actively performing other actions than those in connection with the `paper_review` such as `researching` and `teaching`. A `teaching` association between `teacher`, `student` and `administration` is specialized into `course_teaching` and `supervision`. Association `course_teaching` includes an iterative sequence of `remind_students` actions from `teacher` to `student`s. Figure 8 illustrates how a `person` (with own individual action part illustrated by "… `exercise(…)` …") participates in both `a_paper_review` and `a_course_teaching` associations. The contribution to the person in `a_paper_review` includes `submit_report`. The contribution from `a_course_teaching` includes `remind_student`. An active person object executes `exercise`, `submit_report` and `remind_student` interleaved.

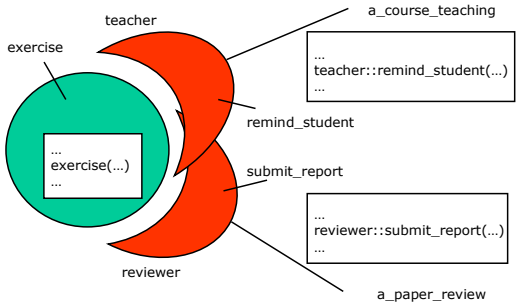

Figure 8: Illustration of `reviewer` and `teacher` roles.

4.3 Specialization of Directive

Collaborations may have general directives prepared for further specialization in directives of sub-collaborations —for example the directive of the

more general association `teaching` may include possibilities for both `course_teaching` and `supervision`. The general directive of `teaching` has the form:

```
1. planning
2. …
3. inner: content
4. …
5. examination
```

This directive illustrates two types of specialization of collaborations: In each of 1) and 5) we illustrate virtual (part) collaborations. This means that for example `planning` is a virtual abstraction with some preliminary description (a directive similar to the `teaching` directive) and may be extended in specializations of `teaching` cf. virtual classes (Madsen et al, 1993). In `course_teaching` (as a specialization of `teaching`) `planning` may be specialized to `course_session_planning`. In 3) we illustrate an explicit `inner` construct (named as `inner: content`). Several such `inner` constructions may be specified in the directive (Kristensen, 1993), (Kristensen et al, 1996). In `course_teaching` the `inner:content` construct is specialized into a sequence of `lecturing` activities (for concrete courses `lecturing` is specified further with respect to number and content). The specialized directive is the original general directive with these two types of specializations included. The execution of a participating active object is still an interleaved execution of its individual action part and its contributions from all such specialized directives. The directive of `course_teaching`, specialized from the directive of `teaching`, includes of the sequence `course_session_planning`, a sequence of `lecturing`, and `curriculum_examination` of the form:

```
1. course_session_planning
2. …
3. … lecturing …
4. …
5. curriculum_examination
```

5 PROGRAMMING

We introduce association classes and objects with roles and directives in schematic programming language form. We include the `paper_review` example as an illustration. Finally we conclude by defining interleaved execution schematically. In general the notation … indicates various less important or repeated parts left out of the descriptions.

5.1 Association Classes

Association classes `Xj` include roles `Rj` (with method `nj`) and local associations `Yj` and a directive (with various characteristic ingredients to be explained later). Association object `Axj` is instantiated:

```
association Xj {
    role virtual Rj for Ci {
       method nj (…)   …
    }
    …
    association virtual Yj {…}
    …
    directive {
       … Yj … inner:Ij … Rj::nj(…) …
    }
}

object Axj of Xj
```

Object `Oci` of class `Ci` enters role `Rj` of `Axj`. A role is qualified by a class, `Ci`, meaning that only objects of this class or its subclasses, `CCi`, may enter that role. Also role `Rj` may invoke methods of `Ci` and `CCi`. The action part of `Ci` illustrates various characteristic ingredients: mi(…) is an invocation of a Ci method, whereas the description `inner:Ii` is replaced by its refined description denoted by `{…}` in `CCj`:

```
class Ci {
    method mi {…}
    …
    action_part {… mi(…) … inner:Ii …}
}

object Oci of Ci

Oci enters Axj as Rj

class CCi extends Ci {… Ii:{…} …}
```

Associations `Yj` and roles `Rj` may be specified as virtual in order to be specialized further in specializations like `XXj` of the enclosing association `Xj` (for roles also the classifying class `Ci` may be specialized as e.g. `CCi`). The directive of `XXj` is the directive of `Xj` where for each `Ij` the description `inner:Ij` is replaced by its refined description denoted by `{…}` in `XXj`:

```
association XXj {
    role Rjk for CCi {…}   …
    association Yj {…}   …
    directive {
       … Ij:{…} …
```

```
      }
   }
```

5.2 Example: paper_review

The paper_review is presented in the language style below. The ordinary concurrent object person executes method exercise repeatedly in its action part:

```
class person {
   method exercise {…}   …
   action_part {… exercise(…) …}
}
```

Association paper_review is a sub-association of review and specializes the directive by prepare_review, carry_out_review and complete_review. Also roles reviewer and coordinator are specialized:

```
association review {
   role virtual reviewer for person {…}
   role virtual coordinator for person {
      method submit (…)   …
   }
   role author {…}
   association virtual prepare_review {…}
   association virtual carry_out_review {…}
   association virtual complete_review {…}
   …
   directive {
      prepare_review
      carry_out_review
      complete_review
   }
}

association paper_review extends review {
   role reviewer {
      method submit_report {…}   …
   }
   role chairman extends coordinator {…}
   association prepare_review {
     author::submit (paper, chairman)
     … chairman::submit (paper, reviewer) …
     reviewer::submit_report(…)
     }
   association paper_selection
        extends carry_out_review {…}
   association extends complete_review {
      … chairman::inform_author …
   }
   directive {…}
}
```

Association course_teaching is a specialized association of teaching—and refines the directive of teaching. Also the roles teacher, student and administration may be specialized—for example for teacher by adding method remind_student:

```
association teaching {
   association virtual planning {…}
   association virtual examination {…}
   role teacher for person {…}
   role student for person {…}
   role administration for organization {…}
   directive {
      planning
      …
      inner: content
      …
      examination
   }
}

association course_teaching extends teaching
{
   association course_session_planning
       extends planning {…}
   association curriculum_examination
       extends examination {…}
   association lecturing {
      … teacher::remind_student(…) …
   }
   role teacher {
     method remind_student {…}
      …
   }
   directive {
      content: {… lecturing …}
   }
}
```

Objects of course_review and paper_review are instantiated. Object John then enters the association objects a_course_teaching and a_paper_review:

```
object a_course_teaching of course_teaching
object a_ paper_review of paper_review
object John of person

John enters a_course_teaching as teacher
John enters a_paper_review as reviewer
```

5.3 Interleaved Execution

Action parts of active objects are executed concurrently with directives of associations, but each action part and directive is executed sequentially:

```
class Ci {
   …
   action_part {… mi(…) …}
}

association Xj {
   role virtual Rj for Ci {…}
   …
   directive {… Rj::nj(…) …}
}
```

A given `Oci` of `Ci` is engaged as role `Rj` of `Xj` in a collection of association objects `Axj` of `Ax`, where

1) the next individual action for `Oci` is `mi(…)`
2) for the collection of `Ax`'s the next action to be executed for `Axj` with `Oci` in role `Rj` is `Rj::nj(…)`

Interleaved execution of `Oci` means, that exactly one out of `mi` and the collection of `nj`'s is selected randomly and executed. For an object of a specialized class engaging in specialized associations the specialized action part and specialized directives are used.

6 RELATED APPROACHES

Notions similar to associations are available in object-oriented modeling whereas in object-oriented programming associations are implemented by means of references. Our associations support both modeling and programming (Kristensen, 2003). We include the association as a first class concept in our modeling and programming notation. In classical object-centric modeling and programming the fundamental problem is that "no object is an island" (Beck et al, 1989). In object-oriented systems an object supports *encapsulation*; the object is *self-contained*; focus is on *structure* instead of *function* and focus is on *methods* instead of *processes*. These characteristics are seen as appreciated properties of object-oriented systems, but are also essential problems because they emphasize an object-centric point of view.

Relations from (Rumbaugh, 1987) are introduced as non object-centric abstractions. In an illustrative example a relation `Employment` with property `Salary` is defined between classes `Person` and `Company`. Objects of class `Person` play the role of `Employee` and objects of class `Company` play the role of `Employer`. The relation `Employment` captures an abstraction, the properties of which we do not place at neither `Person` nor `Company`—the relation is *between* these and therefore in conflict with the intentions of the object-centric approach.

6.1 Language/Notation

Various approaches to notation for non object-centric modeling and programming include: *Relations* (Rumbaugh, 1987) and the corresponding associations in OMT (Rumbaugh et al, 1991) and UML (Booch et al, 1998) are object-external abstractions but these relations/associations only cover structural aspects, not collaboration. *Sequence* and *collaboration diagrams* in UML support the description of object interaction by means of method invocation, but not as abstractions and not integrated with relations/associations of objects. *Complex associations* (Kristensen, 1994) are object-external abstractions and support only complex structural relationships between complex, structured objects. *Subject-oriented* programming (Harrison et al, 1993) and *subjective behavior* (Kristensen 2001) support different views on objects respectively from an external and internal perspective, but not relationships between objects. *Activities* (Kristensen et al, 1996), (Kristensen, 1993) are abstractions over collaborations of objects, but include no support of roleification of objects participating in the collaboration. *Roles* (Kristensen et al, 1996), (Kristensen 1995) are abstractions over roleification of objects for various relationships of objects, but no explicit collaboration is included.

6.2 Collaboration Approaches

Design patterns (Gamma et al, 1994) capture experience of object oriented design and programming. In this approach language constructs for collaborating objects are typically simulated by patterns of objects including for example `DECORATOR`, `OBSERVER`, and `MEDIATOR`. *Activity-based computing* (ABC) (Bardram, 2005) supports mobility and cooperation in human work activities. ABC is a framework supporting a computing infrastructure to describe how to keep track of collaborative activities. The system offers a distributed, real time joint repository for activities including states, participants, communication and information. *Model Driven Architecture* (Zhao, 2005) is supported by the notion of roles (as a modeling paradigm) by viewing object interactions from the dimensions roles, responsibilities and collaborators. The approach yields a semantically rich model, and also a simple, elegant design that is flexible and adaptable.

7 CONCLUSION

Associative programming and modeling is characterized by:

- Object-oriented programming contains object-centric descriptions, and collaboration is implicitly described only and distributed among methods of participating objects. In object-oriented methodologies alternatives exist typically only for analysis and design, but not for implementation
- Associations support associative modeling and programming through abstractions over collaboration. Associations support objectification of integrated collaboration aspects and role aspects. Classification, specialization and aggregation are available
- In associations directives (sequencing rules for interactions among the participating objects) are central, partial descriptions related to the participating objects. The objects execute their own contributions to the collaboration in their context. An active object participating in various associations execute contributions from the directives interleaved

Challenges include:

- Notation at the modeling and programming levels for creation and deletion of associations
- Entry and exit of objects in associations
- Similarities between inheritance of directives and inheritance anomaly (Matsuoka et al, 1993)

ACKNOWLEDGEMENTS

This research was supported in part by the A. P. Møller and Chastine Mc-Kinney Møller Foundation. We thank Palle Nowack, Steffen Jensen and Beata Hargesheimer for collaboration and contributions.

REFERENCES

Bardram. J.E., 2005. *Activity-Based Computing: Support for Mobility and Collaboration in Ubiquitous Computing*. Personal and Ubiquitous Computing, 9(5).

Beck, K., Cunningham, W., 1989. *A Laboratory For Teaching Object-Oriented Thinking*. Proceedings of the Object-Oriented Systems, Languages and Applications Conference.

Booch, G., Rumbaugh, J., Jacobson, I., 1998. *The Unified Modeling Language User Guide*. Addison Wesley.

Burkhardt, J., Henn, H., Hepper, S., Schaeck, T., Rindtorff, K., 2001. *Pervasive Computing: Technology and Architecture of Mobile Internet Applications*. Addison Wesley.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley.

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

Kristensen, B.B., 1993. *Transverse Activities: Abstractions in Object-Oriented Programming*. Proceedings of International Symposium on Object Technologies for Advanced Software.

Kristensen, B.B., 1994. *Complex Associations: Abstractions in Object-Oriented Modeling*. Proceedings of Conference on Object-Oriented Programming Systems, Languages, and Applications.

Kristensen, B.B., 1995. *Object-Oriented Modeling with Roles*. Proceedings of the 2nd International Conference on Object-Oriented Information Systems.

Kristensen, B.B., May, D.C-M., 1996. *Activities: Abstractions for Collective Behavior*. Proceedings of the European Conference on Object-Oriented Programming.

Kristensen, B.B., Østerbye, K., 1996. *Roles: Conceptual Abstraction Theory & Practical Language Issues*. Special Issue of Theory and Practice of Object Systems on Subjectivity in Object-Oriented Systems.

Kristensen, B.B., 2001. *Subjective Behavior*. International Journal of Computer Systems Science and Engineering, Volume 16, Number 1.

Kristensen, B.B., 2003. *Associations: Abstractions over Collaboration*. Proceedings of the 2003 IEEE International Conference on Systems, Man & Cybernetics.

Madsen, O.L., Møller-Pedersen, B., Nygaard, K., 1993. *Object Oriented Programming in the Beta Programming Language*. Addison Wesley.

Matsuoka, S., Yonezawa, A., 1993. *Analysis of Inheritance Anomaly in Object-Oriented Concurrent Languages*. In G. Agha, P. Wegner, and A. Yonezawa, editors, Research Directions in Object-Based Concurrency. MIT Press.

May, D.C-M., B.B.Kristensen, B.B., P.Nowack, P., 2001. *TangO: Modeling In Style*. Proceedings of the Second International Conference on Generative Systems in the Electronic Arts.

Rumbaugh, J., 1987. *Relations as Semantic Constructs in an Object-Oriented Language*. Proceedings of the Object-Oriented Systems, Languages and Applications Conference.

Rumbaugh, J., Blaha, J.M., Premerlani, W., Eddy, F., Lorensen, 1991. *Object-Oriented Modeling and Design*. Prentice Hall.

Weiser, M., 1991. *The Computer for the 21st Century*. www.ubiq.com/hypertext/weiser/SciAmDraft3.html.

Zhao, L., 2005. *Designing Application Domain Models with Roles*. Lecture Notes in Computer Science, Volume 3599.