Working With Objects
OOram Framework Design Principles
Trygve Reenskaug

OOPSLA '96 Workshop:
Exploration of Framework Design Principles

Summary
This reports highlights the application of frameworks to the creation of a software production facility for Intelligent Networks. We define a framework as an encapsulated solution to a recurring problem. The solution consists of role models designed for reuse through synthesis, accompanied by an ensemble of classes designed for subclassing.

We claim that the essence of object orientation is the representation of interesting phenomena as an equivalent structure of interacting objects. Three useful abstractions are proposed: The common class abstraction resembling Entity-Relation modeling; the type abstraction describing interfaces and possibly other external properties of objects; and the role abstraction describing the structure and activities in patterns of interacting objects.

We endeavor to show that the role abstraction is particularly useful in the context of frameworks, since it not only permits the inheritance of object properties but also the inheritance of system properties such as behavior and constraints.

1 The Problem

Norwegian Telecom Research and Taskon have jointly been studying the industrial creation of Intelligent Network Services (IN). The work builds on the IN technology proposed by the EURESCOM project EUP103 [Nilsen], and on the Taskon OOram object oriented methodology [Ree96].

The construction and deployment of IN services is becoming a very large operation. There will be a large and expanding number of available services, the total system complexity will be staggering, and many organizations employing people in different capacities will be involved in its creation and operation.

A first separation of IN into two distinct domains has been suggested in [Bugge91], this is illustrated in figure 1. In an intelligent network, service functions build on network functions offered by the switching domain. This is a client-server architecture, where the Service Domain is a client of the network encapsulated in the Switching Domain.

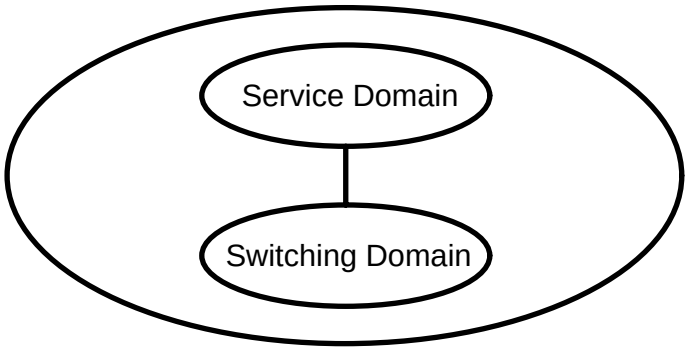

Figure 1. The intelligent network.

We now want to explore how we can construct new IN products; quickly, cheaply, and with high quality. Our focus in this paper will be on reusable frameworks:

> **A Framework is an encapsulated solution to a recurring problem.**
> **It consists of**
> - ***An ensemble of coordinated classes***
> - ***The classes are designed for specialization through subclassing.***

The first step is to assume the complete telecommnucations system can be considered as being a "pure" system of interacting objects as illustrated in figure 2.

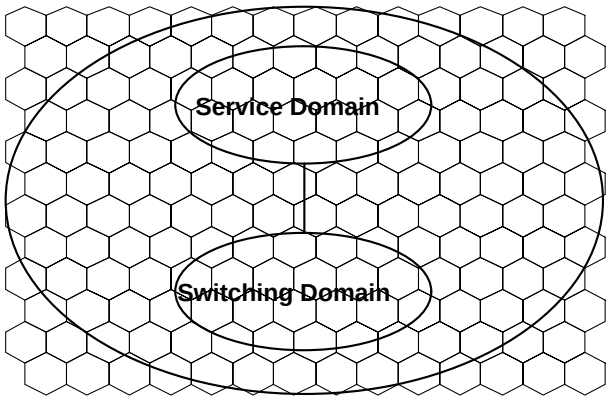

Figure 2. Assume telecommunication system as an object structure.

The second step is to choose a suitable abstraction mechanism to support separation of concern and the construction of reusable frameworks. The three main abstractions that are in use are the *type (interface*), the *class*, and the *role*. The concepts are illustrated through a simple example in figure 3.

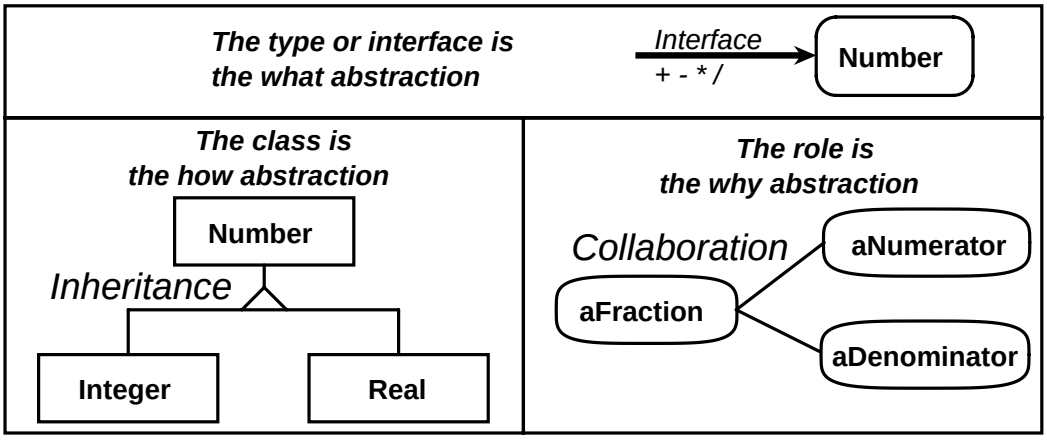

Figure 3. Three abstractions

1. *The Type (or Interface) abstraction* focuses on the external properties of the object, i.e., the set of messages that it can understand. A Type is an implementation-independent description of a set of objects that share specified external properties. Types are commonly organized in a *type hierarchy*, where a *subtype* exhibits all the messages of its *supertype* and adds some of its own. Since the type abstraction hides implementation details, it is useful for describing the component parts of distributed systems. The type supports the idea of reusable interfaces. For example, most current standards of the Object Management Group [OMG] are based on the interface abstraction.

2. *The class abstraction* focuses on the internal implementation of the object. A class is a description of a set of objects sharing a common implementation. In programming, a class denotes a program that controls the properties of a set of objects, called the *instances* of the class. Classes are commonly organized in a *class hierarchy*, where a *subclass* inherits all the code of its *superclass*, and where the subclass may modify the methods of its superclass and also add new methods and object variables. The class supports reusable code and is very useful in programming for sharing concepts and code. It is typically the foundation for reusable class libraries.
3. *The role abstraction* focuses on how patterns of collaborating objects realize certain functionality, called *use cases* or *activities* as follows: *An activity is a task carried out by a set of associated objects in cooperation. In a role model, each role represents a single object involved in one or more activities. Each role only describes those object properties that are of relevance to these activities. [Egil Andersen].* The role model thus preserves object identity and thereby object interaction patterns. Further, since the role is a partial description of a corresponding object, the role model is useful for describing how patterns of objects perform specific tasks. The role abstraction is, therefore, well suited for describing complete phenomena such as the static and dynamic properties of frameworks.

We believe the role abstraction is best suited for reusable frameworks due to its focus on systems of interacting objects. We will therefore concentrate on this abstraction in the following.

2 Role modeling for describing object structures

The common sense of objects is to describe phenomena as structures of interacting objects. *Separation of concern* is a valuable technique for understanding complex systems. If the whole is too complex, we select any *area of concern* and describe the *roles* objects play in that context.

The OOram method is a method for Object Oriented role analysis and modeling [Ree96]. The method was developed by Taskon in response to its own software engineering needs for describing complex object systems, and for its software factory requirements for the rapid and reliable construction of specialized software.

We will in the following discuss five role modeling features: *Role models describe object patterns, Separation of Concern, One model - many Views, Seamless bridge to implementation,* and *Reuse through model inheritance.*

2.1 Role models describe object patterns

As illustrated in figure 4, the OOram role model focuses on patterns of interacting objects: the objects and their interconnection. The following definition is adapted from [Holbæk-Hansen]:

> **A *system* is a part of the real world which we *choose* to regard as a whole, separated from the rest of the world during some period of consideration; a whole that we *choose* to consider as containing a collection of interacting objects, each object characterized by attributes and by *actions* which may involve itself and other objects.**

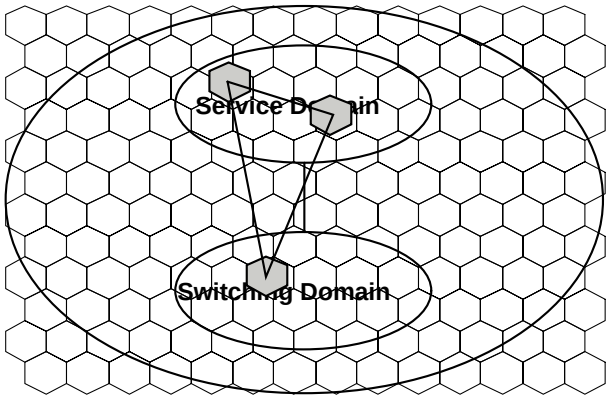

Figure 4. A system as a pattern of objects

Role modeling is based on an important result from systems theory: the value of a system is greater than the sum of the values of its parts. The reason is that the system structure adds value not present in any component. This is illustrated in the trivial example of figure 5; the concept of a fraction is stronger than an arbitrary collection of three numbers. The consequence for practical modeling is that we can capture the synergy caused by a particular pattern of interdependent parts. This is useful for a number of different purposes such as describing the processes in the enterprise; the enterprise in the context of its environment; the overall enterprise information systems; the tasks of individuals and their computer tolls; the enterprise domain services. It is also useful for creating generic descriptions in all these areas, descriptions capture essential properties and that can be specialized for a variety of purposes.

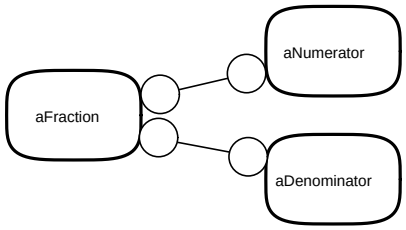

Figure 5. Trivial role model

2.2 Separation of Concern

The above definition of *system* permits different choices. We study different subjects by choosing different areas of concern and thus studying different aspects of our objectified understanding of the world. Several systems can and do coexist as illustrated in figure 6. One possible subject is the part service employed by a caller when he wants to establish a telephone conversation (POTS - Plain Old Telephone Service). Another is the part service that responds to the call on behalf of the called party. A third is a general mechanism for invoking any specified service. A fourth is the establishment of an actual communication path, marked "Switching Domain" in figure 1. The interdependencies between the subjects is expressed as objects playing several roles as illustrated in the figure. The subjects are described as role models, their interdependencies are managed by objects integrating their behaviors when playing multiple roles.

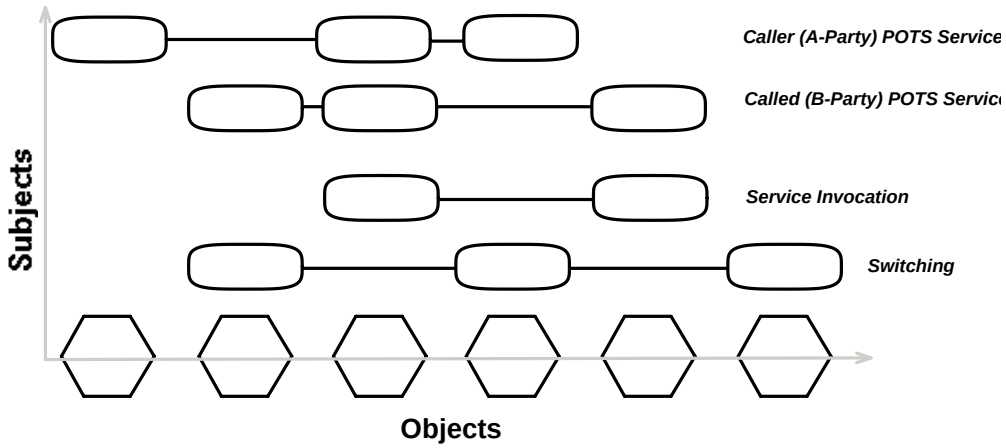

Figure 6. Subjects and objects

The consequence for practical modeling is that we are freed from the tyranny of the single model. Viewing the world and our information systems through tinted "object" glasses, we can describe any phenomenon we might be interested in without ever having to formulate the single, global model. And this is just as well, because we are becoming parts of a truly global network that we do not control and that we can never hope to fully understand.

2.3 One model - many Views

A role model describes the static and dynamic properties of a system. Different *views* highlight different aspects of the model. For example, a framework could describe and implement a general system for the invocation of services. Each user would have his own instance of the *Invocation* role, this object holds the user's set of available services and knows how to select an appropriate service object in response to a request from some *Client* object. The *Invocation* object selects the appropriate *Service* object and initializes it in an appropriate manner before giving a *Service* reference to the *Client*. This framework is illustrated below.

A *Collaboration View* highlights the system's role collaboration structure, showing the roles and the message communication paths between them. An example is shown in figure 7. The role is shown as a superellipse, and the communication path as a line. The small circle is a port for sending messages, associated with the port are the messages that the role may send to its collaborator. A double circle indicates a *many* relation. In figure 7, the *Invocation* can handle many clients; the *Client* role represents one archetypical client.

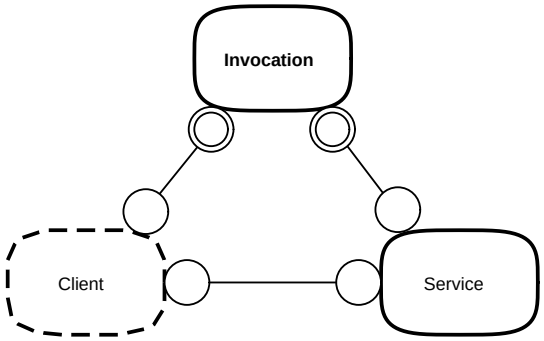

Figure 7. Basic Invocation Framework

A *Scenario View* shows a particular trace of message interactions for a system activity. An example is shown in figure 8. The roles are shown horizontally along the top of the diagram; time increases in the down direction. Message interactions are shown as horizontal arrows from the sender role to the receiver role. The first interaction is shown at the top, it shows an environment role sending a stimulus message to a system role.

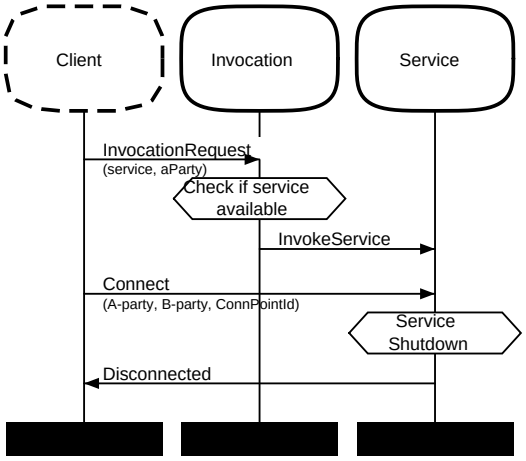

Figure 8. Basic Invocation Framework

A *Method View* shows how a role handles a received message. Internal operations are usually shown in pseudocode, while messages sent are shown explicitly. Figure 9 shows how the A POTS service establishes connection with the corresponding B service.

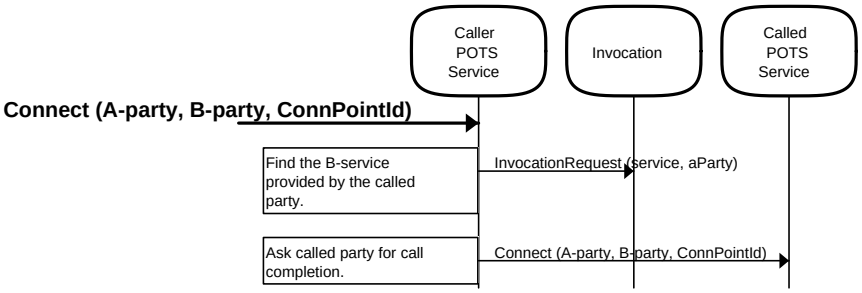

Figure 9. Sample Method: Establish communication with called party.

2.4 Reuse through model inheritance (synthesis)

Separation of concern is a powerful device enabling us to *divide and conquer*. The converse operation is equally powerful. *Role model synthesis* enables us to compose *derived systems* from one or more *base systems*, preserving their object patterns and activities in a controlled manner.

Figure 10 indicated how this is achieved. We specify how objects play several roles from different role models and also how a role in one model acts as environment role in another model.

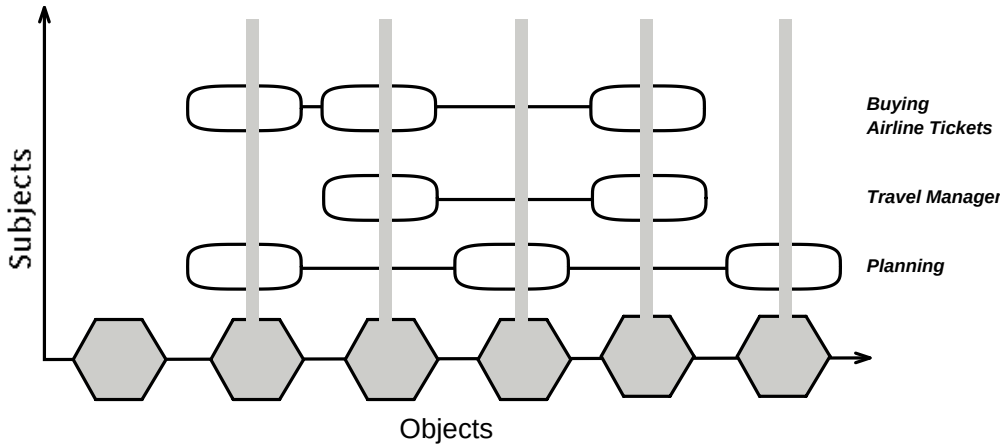

Figure 10. This 'hat stand' synthesis illustration is due to Philip Dellaferra.

There are three forms of synthesis that are particularly useful for reusable components. These forms are illustrated below.

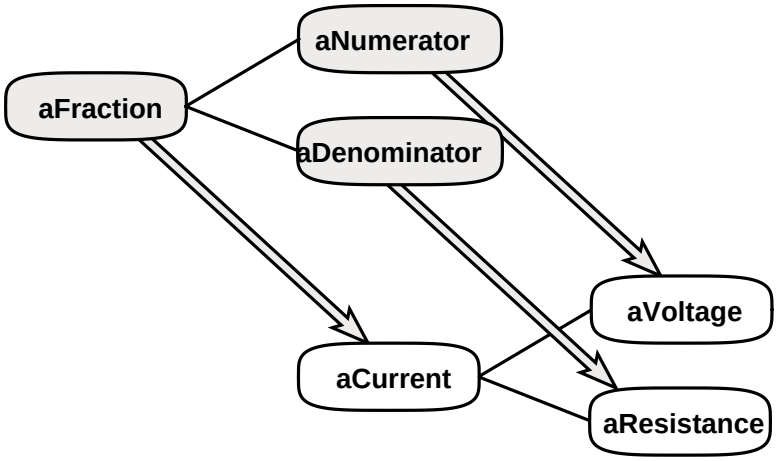

Figure 11. Specializing a general concept with synthesis.

Specialization -- generalization. A base model representing a general concept can be specialized in the derived model. This is illustrated by a somewhat artificial example in figure 11, where the general concept of a fraction is specialized into a model of Ohm's law in electrical engineering. All static and dynamic properties of fractions are preserved. In addition, the derived model can include features that are specific for electrical engineering. (Notice that the figure illustrates a single synthesis operation, the three open arrows show one set of corresponding role mappings.

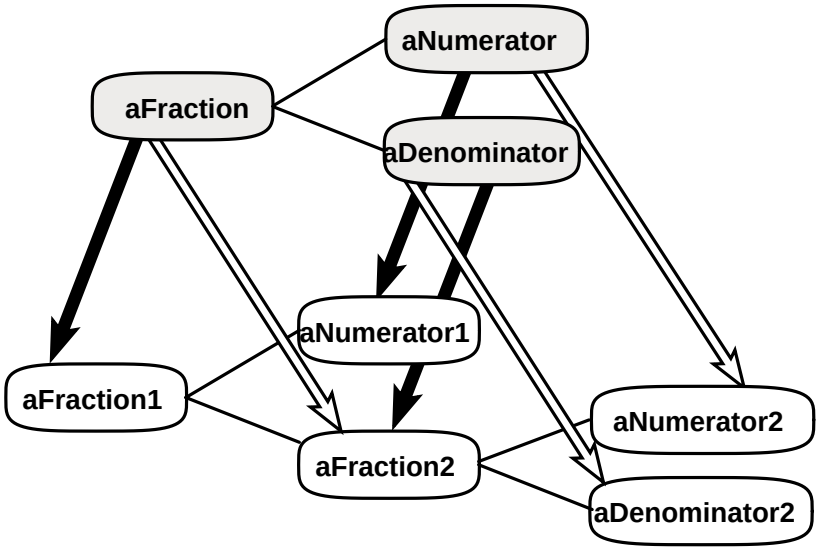

Figure 12. Composition on same level of abstraction.

Composition on same level of abstraction. Base models describing related phenomena may be synthesized together into a composite derived model. For example, a bank account deposit base model may be combined with a withdrawal base model into a derived bank account model. Another example is shown in figure 12. The simple fraction model is synthesized twice to yield a kind of composite fraction.

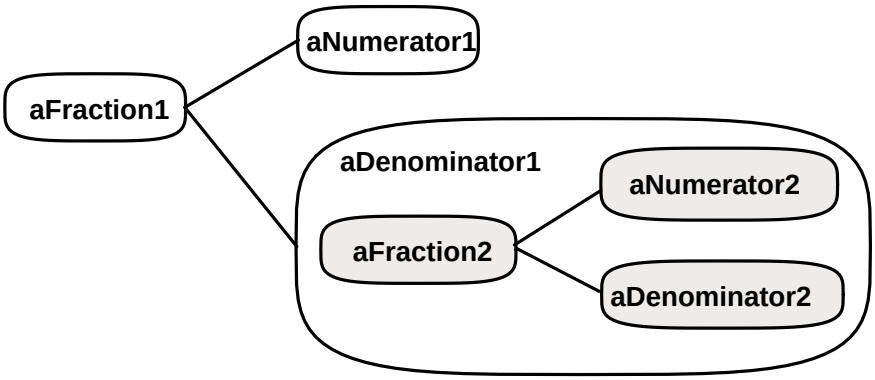

Figure 13. Aggregation

Aggregation. The object encapsulation property makes the internal construction of an object invisible from outside the object. A simple role can therefore be replaced by a role encapsulating a separate role structure. This is illustrated in figure 13, where the Numerator role of the simple fraction model is replaced by a role encapsulating another instance of the fraction model. The net effect is the same as in the previous example; but the internal construction of the Fraction1 denominator is now invisible to the outer Fraction model.

2.5 Seamless bridge to implementation

The object programmer has to think in terms of collaborating objects and write code in terms of classes, i.e., isolated objects. This conceptual clash is bridged through role modeling. A role is a partial specification of an object and thus of a class. A port is an abstraction of a variable that can be mapped on to some kind of program variable. Role model synthesis maps to the inheritance in a coordinated ensemble of classes.

While the role, type, and class abstractions serve different purposes, they are related because through the common concept of objects. There is a many-to-many relationship between role, type, and class as illustrated in figure 14.

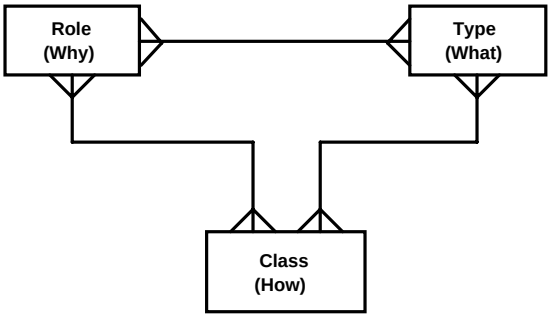

Figure 14. Semantic model of Role - Type - Class

3 Frameworks for Intelligent Networks

A particularly powerful mechanism is the OOram Framework. This is a reusable component that solves some general problem with a pattern of interacting objects. The pattern is described in a base model and it is implemented as an ensemble of base classes (superclasses). The framework is used by synthesizing the base model into the product design model, and also by programming the implementation classes of the product as derived classes (subclasses) of the reusable component's classes.

An OOram Framework is an encapsulated solution to a recurring problem.

It consists of

- ***Role models describing the static and dynamic properties of the solution***
- ***The role models are designed for specialization through synthesis.***
- ***A corresponding ensemble of coordinated classes***
- ***The classes are designed for specialization through subclassing.***

Figure 15 illustrates how the *Invocation framework* described above is applied to let the Caller find and access an appropriate *Caller POTS Service* object. Notice that this figure illustrates one synthesis operation; all base model roles are mapped onto corresponding derived model roles in order to preserve the overall behavior of the base model.

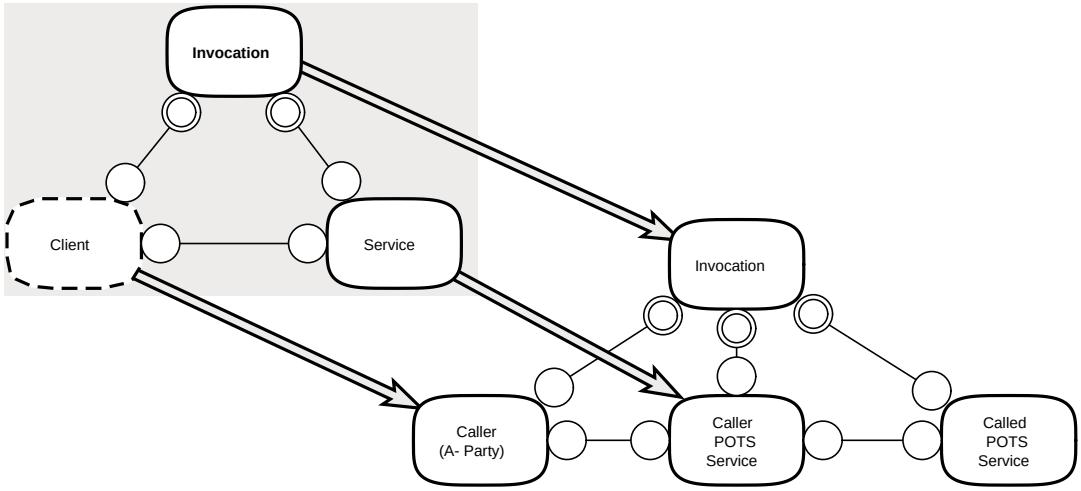

Figure 15. Synthesize POTS from reusable base models

The *Caller POTS Service* object can now reapply the invocation service to find the appropriate *Called POTS Service* belonging the the called user. The *Called POTS Service* now plays the role of the *Client*, and the *Called POTS Service* plays the *Service* role. This is illustrated in figure 16.

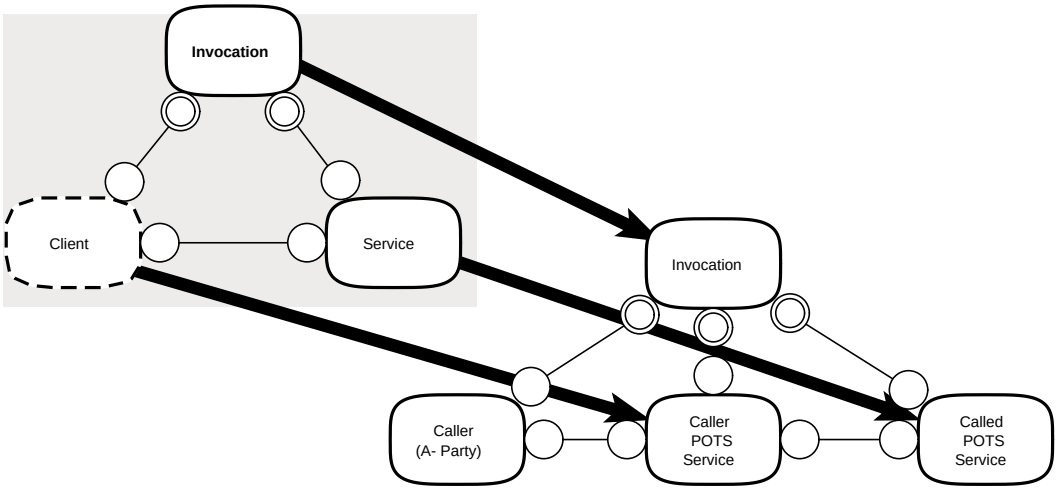

Figure 16. Repeat to synthesize B-side

Figure 17 shows a corresponding scenario. The two applications of the Invocation framework can readily be appreciated by comparing this scenario with figure 8.

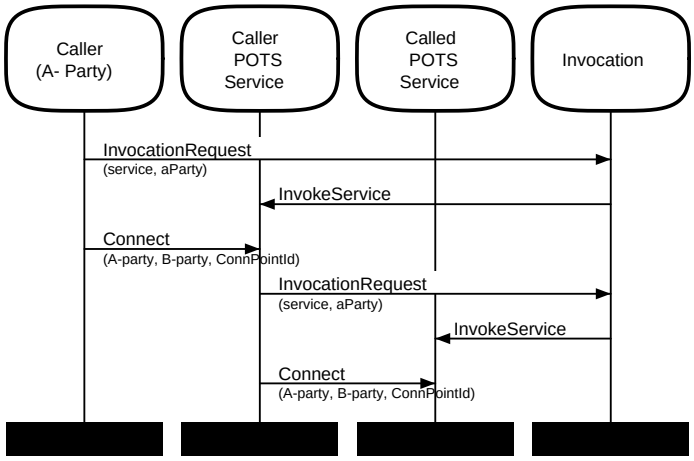

Figure 17. Sample Scenario: Open POTS Telephone Service

If we later were to develop a *Call Forward* service product, this product can be simply constructed from existing frameworks. In this example, we first construct a *Call Forward* object as a new variant of the *Called POTS Service*. As illustrated in figure 18, this new variant plays the Invocation framework a third time in order to find the new termination point. This new C-party could terminate the search by providing a *Called POTS service* as shown. Alternatively, it could implement another forwarding step by providing another *Call Forward service*.

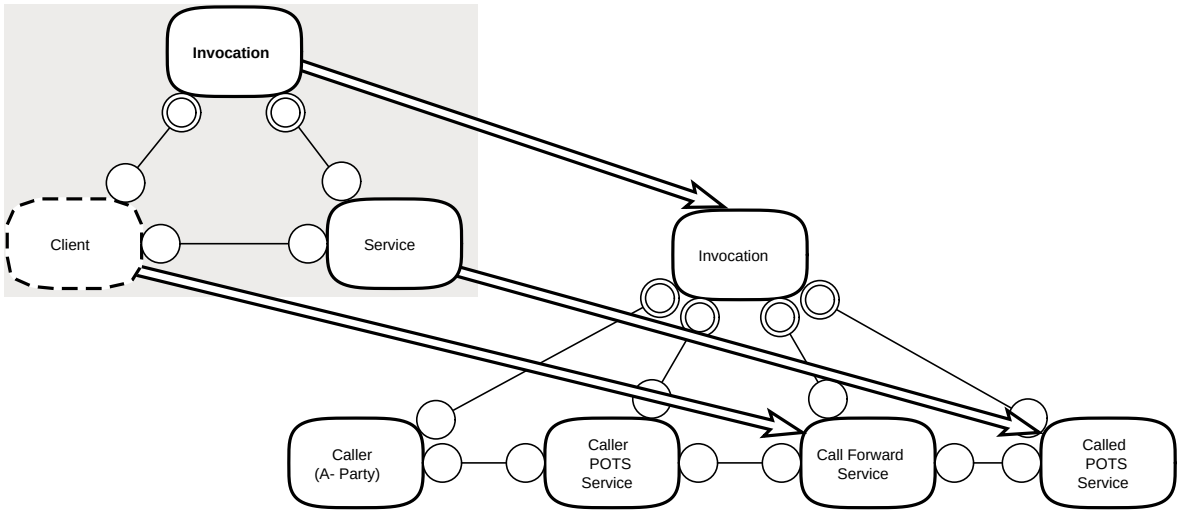

Figure 18. Construct Call Forward by replacing B-Service and repeat synthesis a third time

4 OOram: The Common Sense of Objects for frameworks

1. *Role models describe object patterns.* This enables the framework precisely to describe the static and dynamic properties of a cluster of interacting objects.
2. *Separation of Concern.* This permits the identification and description of solutions to a series of generic problems.
3. *One model - many Views.* This supports the highlighting of different aspects of the framework. We have shown examples of collaboration view, scenario view, and method view. All showing different aspects of the same model.

4. *Seamless bridge to implementation.* Supports the implementation of the framework in a way that prepares the ground for constructing the product classes by subclassing the framework classes.
5. *Reuse through model inheritance.* This is the crux of the method. It permits the controlled reuse of framework structure and interaction patterns.

5 References

The author: trygve@taskon.no

Taskon: http://www.sn.no/taskon/

[Nilsen] R. Nilsen, J. Simons, P. Dellaferra: *Object oriented IN service provision.* TINA Workshop. L'aquila, Italy, 1993.

[Ree 96] Reenskaug, Wold, Lehne: *Working With Objects.* Manning/Prentice Hall 1996. ISBN 0-13-452930-8

[Bugge91] Bugge B. et al. *Methods and tools for service creation in an intelligent network, initial document.* Kjeller, Norwegian Telecom Research, 1991 (Research Doc. No. 34/91)

[Ree 93] Trygve Reenskaug: *The Industrial Creation of Intelligent Network Services.* TINA Workshop. L'aquila, Italy, 1993.

[Holbæk-Hansen] Holbæk-Hansen, et.al.: *System Description and the DELTA language.* Norwegian Computing Center publication no. 523. Second printing. Oslo, 1977.