

Using Actors as a computational model for OOram

H. Vestli

Telenor Research and Development,

P O Box 83, 2007 Kjeller, Norway

Telephone: + 47 63 84 87 88. Fax: + 47 63 81 00 76

E-mail: hakon.vestli@fou.telenor.no

Abstract

OOram is a well-established object-oriented software methodology, but it lacks a precise computational model. In this paper we study the use of Actors - the actor model of computation - as a computational model for OOram. In particular, we study role model synthesis, introducing a new type of synthesis. This could lead to an understanding of OOram in general, and of role model synthesis in particular, that is both more powerful and more precise, making OOram more suitable for analysis and design of object-based distributed systems.

Keywords

Actors, OOram, computational model, concurrency, distribution, visual language.

1 INTRODUCTION

Our area of interest is telecommunication applications, thus we need methods for software production that allow modelling of distributed and concurrent programs. OOram - *Object-oriented role analysis and modelling* - is a contemporary method for analysing and constructing object-oriented systems (Reenskaug, Wold and Lehne, 1995; Reenskaug et al., 1992; Andersen and Reenskaug, 1992). OOram has been the subject of study in Telenor for some time. The problem with OOram is in our view the lack of a precise computational model.

Actors (Agha, 1986; Agha et al., 1994) is a formal basis for concurrent distributed systems. This makes Actors a candidate computational model for OOram in our context and this paper describes the attempt of using Actors as a computational model for OOram, in particular in order to understand role model synthesis.

Section 2 discusses briefly the mapping from OOram role models to Actors, a topic more thoroughly discussed in (Vestli, 1994) and (Vestli, 1996). Section 3 discusses role model synthesis, and Section 4 contains concluding remarks.

2 MAPPING OORAM ROLE MODELS TO ACTORS

The main parts of a role model in OOram are the area of concern, a role diagram, and a set of scenarios. Figure 1 shows a very simple role diagram. The area of concern for this role model is *counting* in its simplest form, and the role model has two roles, *Counter* and *Client*. *Client* is the role that requests the counting services that *Counter* offer. An *environment role* is a role that triggers activity in a role model or receives triggers from the role model for activity in the environment. Non-environment roles (or 'internal' roles) are called *system roles*. In Figure 1, *Client* is an environment role and *Counter* is a system role. Environment roles are shown by shading the role symbols in role diagrams and scenarios.

The main assumption made in order to use Actors as a computational model for OOram, is that role instances are actors according to Agha (1986).



Figure 1 Counter role model.

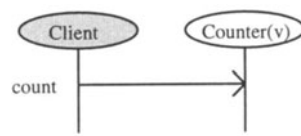


Figure 2 Sending a message.

The message interactions between roles in a role model can be specified by a set of scenarios. The basic action used in OOram scenarios is the sending of a message from one role instance to another, as shown in Figure 2, where an instance of a *Client* role sends the message *count* to an instance of a *Counter* role with current value *v*. The vertical lines in a scenario are called *time-lines*, with time flowing from top to bottom.

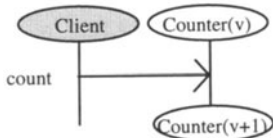


Figure 3 Specifying the replacement role.

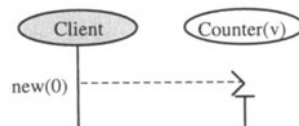


Figure 4 Creating a new role instance.

OOram scenarios have no mechanisms for specifying 'replacement roles' (the role model analogue of a replacement behaviour in Actors), thus we introduce one. See Figure 3 for an example. When a *Counter* role instance receives a *count* message, its replacement role is in Figure 3 specified as *Counter(v+1)*.

According to Reenskaug, Wold and Lehne (1995) we can tag a message with a 'c' in order to indicate the creation of the receiver prior to the sending of the message. In order to make OOram scenarios Actors-compliant, we separate the creation of a role instance from the subsequent sending of a message to it. An example is shown in Figure 4, where an instance of a *Client* role creates an instance of a *Counter* role with value 0. In Reenskaug, Wold and

Lehne (1995), a dotted line indicates a method return, but we prefer to use the dotted line for role instance creation in order to comply to actor creation in the event diagrams in (Agha, 1986).

We have in Vestli (1994) and Vestli (1996) expanded the OOram scenario technique beyond what is described here and also described how a set of scenarios can be mapped to a set of actor programs. These discussions are not vital in the discussion of role synthesis to follow.

3 UNDERSTANDING OORAM ROLE SYNTHESIS

Synthesis of role models in OOram is a composition mechanism for role models. In this section we discuss OOram composition in light of using Actors as the underlying computational model.

As elaborated in Vestli (1994) and Vestli (1996), each role corresponds to an actor behaviour. The scenarios of the derived (or composed) role model is the sum of the total set of scenarios from the base (or original) role models.

Reenskaug, Wold and Lehne (1995) define two types of synthesis. These are *safe synthesis*, in which the correct functioning of a base model will automatically be retained in the composite model after synthesis, and *unsafe synthesis*, where the composite model has to be analysed in total before we can assume it to be correct. Safe synthesis is the 'plug-and-play' version of synthesis, while unsafe synthesis makes a composite model that is more than the sum of its parts.

Safe synthesis can again be divided in two types, *superposition*, where a base model stimulus remains a stimulus in the derived model and an environment role in the base model remains an environment role in the derived model, and *aggregation*, where a base model activity *details* a method in the derived model. Here an environment role in the base model may become a system role in the derived model. In Reenskaug, Wold and Lehne (1995), these types are specified for synchronous messages only.

This section discusses primarily safe synthesis. Unsafe synthesis can in many respects be regarded as steps in the evolution of *one* role model. We will discuss superposition, aggregation and a new type of synthesis (called 'role switching' synthesis) that can be introduced because we use Actors as computational model.

3.1 Superposition

We assume that in a synthesis operation two roles are composed into one. The simplest requirement in a synthesis operation is that the derived role must be able to respond to all messages the two roles could respond to before the synthesis (Andersen and Reenskaug, 1992). We therefore call superposition synthesis 'naive' role model synthesis. On the scenario level the names of the roles in the base role models are changed to the name of the derived role. One of the names can be kept or a new name can be devised. Mapping the role models to Actor programs, the behaviours corresponding to the base roles are simply concatenated to produce the behaviour corresponding to the derived role, that is, the method set from one behaviour is added to the other (assuming no overlap). See Figure 5 for an example.

In the figure, *CounterButton* (CB) is the derived role. In superposition synthesis, the derived role plays the two component roles at the same time - it is at any time able to respond to any message sent to either the *Button* part or the *Counter* part of *CounterButton*. The trivial

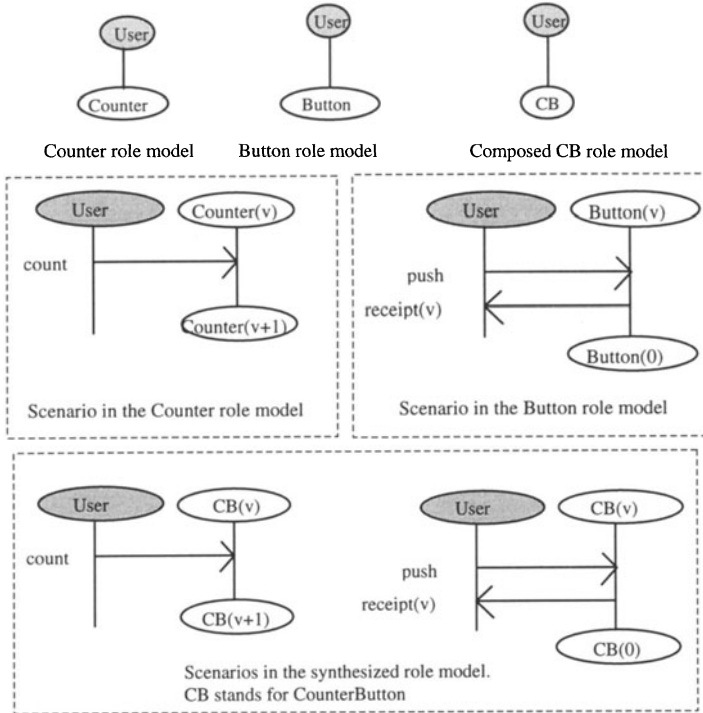


Figure 5 Example of superposition synthesis.

case of superposition synthesis occurs when the roles to be synthesised has the same name in the two role models.

3.2 Aggregation

Aggregation is very analogue to actor composition. We exploit this in the discussion of interpreting aggregation using Actors as a computational model for OOram. We can call this variation of synthesis for 'structured naive' synthesis. This is because it is a special case of naive (or superposition) synthesis - the reason for calling it structured is expounded below.

In Actors, an *actor configuration* is a 'system' of actors. *External actors* are actors outside the configuration that actors within the configuration can send messages to. *Receptionists* are actors within the configuration whose addresses are known outside the configuration. External actors are the instruments used for composing actor configurations. Composition of actor

configurations is done by making the external actors *forwarders* to receptionists in another configuration. See Figure 6.

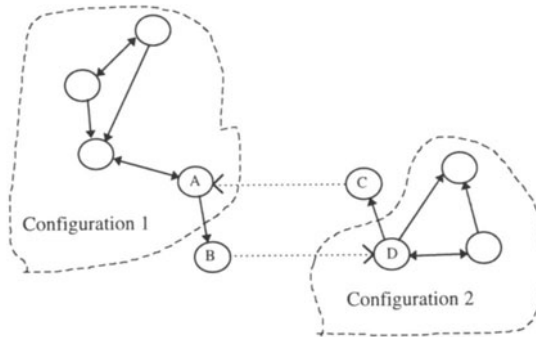


Figure 6 Composing actor configurations.

Here, A and D are receptionists in two different configurations and B and C are external actors - B is external to Configuration 1 and C is external to Configuration 2. In composing the two configurations, external actor B is made to forward all messages to D while external actor C is made to forward all messages to A. Alternatively, we could state that $B = D$ and $A = C$.

In order for the new configuration to work we must assure that A is able to process all messages sent from D via C, and that D is able to process all messages sent from A via B.

Using the principles of actor composition to explain aggregation, we confine aggregation in the following way. We have to identify two pairs of roles to be joined, one system role and one environment role in one role diagram are to be joined with one environment and one system role in the other role model, respectively. See Figure 7.

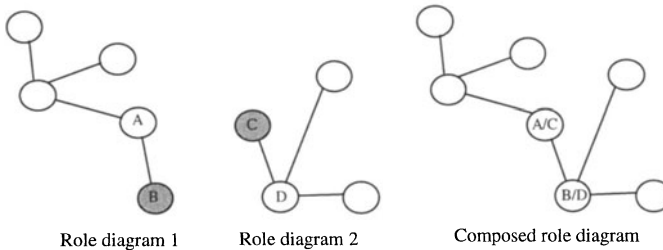


Figure 7 Aggregation: role diagrams.

In Figure 7, we have joined A and C into A/C. B and D are joined into B/D. We have to make sure that B/D can process at least the messages sent to B from A, and that A/C can process the messages sent to C from D.

For this composition to work, the original role models must 'match' for example by identifying that in Figure 8, m2 and m4 denote the same message (possibly after renaming).

When A and C is joined into A/C and B and D is joined into B/D, the scenarios attached to the derived role model explain how the original role models are connected. In the figure m4 is renamed to m2 to give this connection.

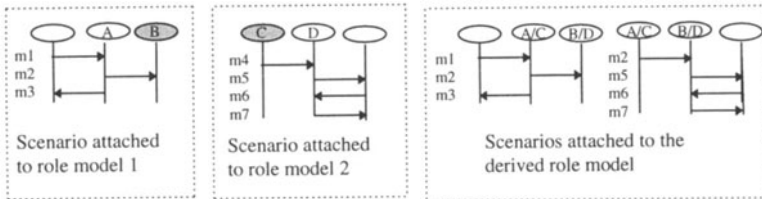


Figure 8 Aggregation: scenarios.

Since B and C are environment roles, they will (analogously to external actors) represent the world outside the role model. As such they can be regarded as dummy roles, just waiting to be joined with other roles. In practice we could thus name A/C as A and B/D as D. Note that after the composition, B and C are no longer environment roles.

This special case of naive synthesis is called structured because the only roles involved are environment roles and their immediate collaborators. Arbitrary roles cannot be joined. The effect of such a synthesis operation is, as seen from one role model, that environment roles are replaced by complete role models. This makes a clear separation between what is a software component (the system roles) and its interface (the environment roles).

As mentioned above, Reenskaug, Wold and Lehne (1995) define aggregation for synchronous messages only. Using Actors as computational model, we now have an understanding of aggregation that is safe even for parallel processes.

3.3 'Role switching' synthesis

A variation of synthesis that emerges because we use Actors as a computational model for OOram is to leave the role names of the scenarios attached to the derived role model unchanged and specify the connection between the roles by modifying the replacement roles in the scenarios. We call this 'role switching' synthesis. In role switching synthesis, the different roles to be played by the derived role are played one at a time, and the switching between the roles are made explicit in the scenarios. See Figure 9 for an example.

The example shows how a base station (BS) in a mobile network switches between active and passive modes (in the derived role model). There are separate role models for the active mode and the passive mode. To get the complete behaviour of a base station that can be in either mode, the role models are synthesised. When a BS role instance is created, it must play either the ABS role or the PBS role. The instance can play both roles of active and passive, but only *one at a time*. This example is a simplified fragment of an example in (Orava and Parrow, 1992).

We could say that role switching synthesis is the most 'safe' type of unsafe synthesis. This is because we do have to check that the derived role model is meaningful, but we change only the replacement roles in the base models, not the sequencing of messages.

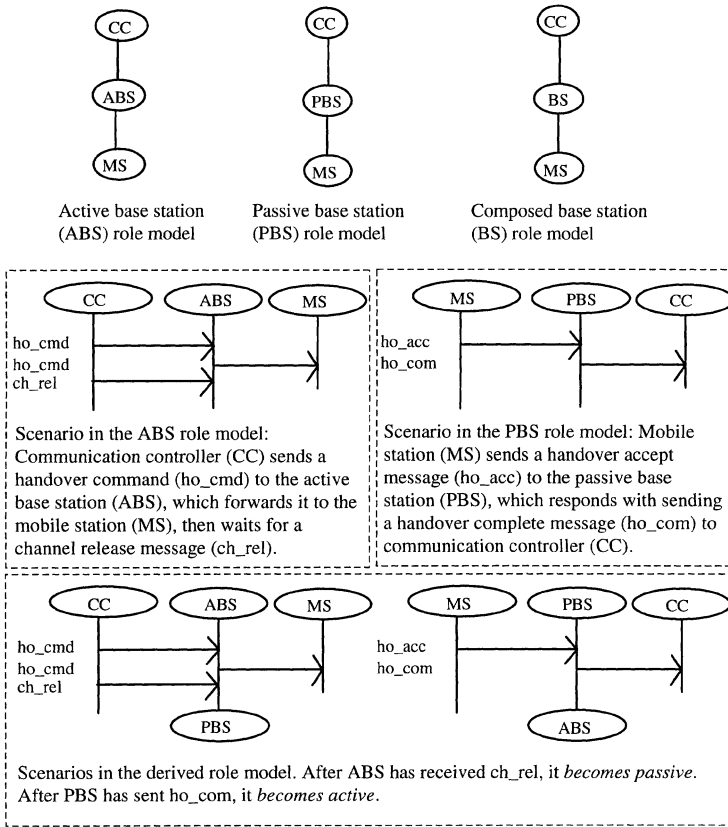


Figure 9 Example of role switching synthesis.

On the level of actor programs, this would mean that the replacement behaviours of some methods are modified.

4 CONCLUDING REMARKS

In this paper, we have discussed OOram role synthesis in light of using Actors as a computational model. We have introduced a new 'type' of role synthesis in this discussion and shown that the 'actor variation' of OOram aggregation is safe also for parallel processes. We also claim that aggregation can be viewed as a special case of superposition.

This paper continues the work presented in (Vestli, 1994), where the mapping from scenarios to actor programs was first introduced. Scenario techniques are used by e.g.,

Jacobson, Christerson and Overgaard (1992), Rumbaugh et al. (1991) and ITU-T (1992). Hewitt (1977) introduced what he called *event diagrams* in order to aid in abstracting scripts of modules that are capable of realising actor behaviours. We have found Agha's *actor event diagrams* (Agha, 1986) more similar to message sequence techniques in general, and have thus been more inspired by those in devising the mapping from scenarios to actor programs. We have found no previous work connecting scenarios or OOram to Actors.

Acknowledgements

I wish to thank the anonymous referees of this paper for their comments.

5 REFERENCES

- Agha, G. (1986) *Actors: a model of concurrent computation in distributed systems*. MIT Press, Cambridge, MA.
- Agha, G. et al. (1994) *A foundation for actor computation*. Submitted to Journal of Functional Programming (available by anonymous ftp at sail.stanford.edu: /pub/MT/94actors.ps.Z).
- Andersen, E.P. and Reenskaug, T. (1992) System design by composing structures of interacting objects. In *Proceedings of ECOOP'92* (Lecture Notes in Computer Science, **615**), Springer, Berlin, 133-152.
- Hewitt, C. (1977) Viewing control structures as patterns of passing messages. *Artificial Intelligence*, **8**, 323-364.
- ITU-T. (1992) *Message sequence chart (MSC)*. (ITU-T Recommendation Z.120.) Geneva.
- Jacobson, I., Christerson, M., Overgaard, G. (1992) *Object-oriented software engineering*. Addison-Wesley, Reading, MA.
- Orava, F. and Parrow, J. (1992) An algebraic verification of a mobile network. *Formal aspects of computing*, **4**, 497-543.
- Reenskaug, T. et al. (1992) OORASS: Seamless support for the creation and maintenance of object oriented systems. *Journal of object-oriented programming*, **5**(6), 27-41.
- Reenskaug, T., Wold, P., Lehne, O.A. (1995) *Working with objects: the OOram software engineering method*. Manning, Greenwich, CT.
- Rumbaugh, J. et al. (1991) *Object-oriented modeling and design*. Prentice-Hall, Englewood Cliffs, NJ.
- Vestli, H. (1994) Visual specification of actor configurations. In *Proceedings of Visual Languages '94*, IEEE Computer Society Press, Los Alamitos, CA, 110-117.
- Vestli, H. (1996) Using Actors as a computational model for OOram. In the *Participants proceedings of FMOODS'96*.
- Yonezawa, A. (ed.) (1990) *ABCL. An object-oriented concurrent system*. MIT Press, Cambridge, MA.

6 BIOGRAPHY

Mr. H. Vestli received his MSc at the Norwegian Institute of Technology in 1990 and joined Telenor R&D as a research scientist shortly after. He has been working with the use of formal methods, visual languages, and reuse in object-oriented analysis and design.