09S-SIW-032

Guidelines for reuse of multi-resolution physical models

Daniëlle Keus
Frank Benders
Klaas Jan de Kraker
Rob van der Meer
Peter Langeslag
TNO Defence, Security and Safety
Oude Waalsdorperweg 63
2509 JG The Hague
The Netherlands
danielle.keus@tno.nl, frank.benders@tno.nl, klaas_jan.dekraker@tno.nl,
rob.vandermeer@tno.nl, peter.langeslag@tno.nl

Keywords:
Multi-Resolution Modeling, reusability, meta-modeling, simulations

ABSTRACT
The reuse of mathematical models of physical phenomena, here called physical models, within simulations is attractive because it has the potential of improving simulation results while reducing implementation and testing work. In practice, however, it can be complicated to reuse a model that was originally developed for other studies and applications. It becomes even more difficult when multiple stakeholders have different ideas on how to use and extend the physical model.

An attractive approach to deal with this problem is to develop and maintain a multi-resolution physical model that can be reused for different purposes. A multi-resolution model is a model that represents a system or phenomenon at different levels of fidelity simultaneously. On the process side, such an approach requires a policy for managing the use, the support and the development of the multi-resolution physical model. On the more technical side, a multi-resolution physical model requires a model specification that maintains all internal and external relationships while providing interoperability.

Focusing on the reusability of physical models, this paper presents and analyzes two examples of multi-resolution physical models from different domains and discusses how they are used in different simulations. The examples concern the modeling and use of (1) a missile model, including its different components, and (2) a signal propagation model in an underwater environment. The examples show the global approach of how multi-resolution physical models can be used. Based on the experiences of modeling and simulation experts and physical modeling experts, this paper presents our guidelines and recommendations on how to facilitate the use of multi resolution physical models in simulations. These guidelines are structured according to the different software development phases, and help to improve the reusability and utility of physical models.

1. Introduction

This paper discusses *physical models*, which we define as mathematical models of physical phenomena. Typical examples of physical models are weapon models and sensor models. However, physical models also include physical phenomena such as wave propagation. In general physical models describe a physical phenomenon in extensive detail. Hence these models contain lots of formulas, tables, graphs and other details. The descriptions contain much knowledge and they are developed by subject matter experts like physicists.

Physical models are mostly developed for analysis purposes, i.e. for analyzing the physical phenomenon itself, and therefore the models are often of high fidelity.

However, the resulting models are also very useful for reuse and integration in other simulation applications. The availability of high quality physical models makes it very attractive to integrate these models into other simulations instead of rebuilding them. In other words, when developing a simulation for training purposes or for

operational analysis, a part of this simulation can be a physical model.
In [1] three kinds of reuse where distinguished: object reuse, component reuse and application system reuse. Reusing a physical model in a simulation is an application of component reuse.
This paper presents two examples of physical model reuse, which describe two typical ways to integrate a physical model into a simulation.
The first example shows how a physical model of a missile is used in the JROADS air defence simulation suite. JROADS is an in-house simulation tool used for training purposes, exercise support and operational analysis. The second example concerns an underwater acoustic propagation model, which is used in our Under Water Testbed (UWT) for operational analysis of underwater warfare. In both of these examples fidelity level of the physical model can be chosen.
Such a reuse of models can be very cost effective because it leverages the development investment of the high fidelity physical model and reduces the total maintenance effort.
However, successfully integrating a physical model into a simulation is difficult. Several challenges must be overcome. These challenges are not only technical in nature. Many of them are actually organizational issues. Technical issues mainly concern the interfacing of the physical model and the simulation. Organizational issues concern the cooperation between the different experts and their roles and responsibilities, for example regarding requirement management and support.

The remainder of this paper is organized as follows. Section 2 presents and discusses two alternative ways to integrate a physical model into a simulation, which are illustrated in Section 3 using real life examples. Next, Section 4 presents our reuse and integration guidelines based on our experiences. Lastly, Section 5 presents our conclusions and recommendations.

2 Integration approach

TNO Defence, Safety and Security develops many physical models for the analysis of physical phenomena.
Our general objective is to reuse such physical models as a component in simulation models. We distinguish two alternative integration approaches: reusing a physical model as a library or integration of the physical model as a component in the simulation.

2.1 Reuse a physical model as a library

Often a physical model is implemented in software as a library, as illustrated in Figure 1. In this approach the physical model can be seen as a component that can be reused. For example, a missile library can be reused in various simulations.

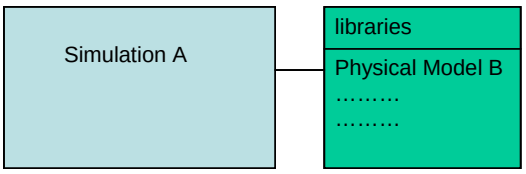

Figure 1: Reuse a physical model as a library

When creating a physical model as a library, first the responsibilities of the model and its interfaces need to be clearly defined. Then the transformation step can be made to make a library.

The following aspects have to be taken into account when creating and using such a library:

1. *Component interfaces*
   The 'interface agreement' defines the input and output of the physical model and it exposes information of the physical model to the host simulation. For example the software engineer responsible for simulation A needs to know how to interpret a parameter of the missile model (physical model B) in order to use this parameter in simulation A. The software engineer of simulation A does not need to understand the details of the physical phenomenon and the local outcomes of physical model B.
2. *Use of this component: considering the fidelity levels*
   The physical model needs to provide an adequate fidelity level. The most flexible solution is that the fidelity level can be chosen. Subsection 2.3 describes that different fidelity levels often also require different interface parameters. In general the fidelity level of the physical model needs to match the fidelity level of the simulation.
3. *Validity domain and range*
   The validity characteristics of the physical model have to be taken into account. The validity domain of the physical model, for example the model for signal propagation, can only be used in an underwater environment. The model parameters must be in the validity range. For example, in a radar model only those frequencies can be used that are within the frequency validity range.
4. *Timing aspect of an event driven simulation*
   The time aspect may be implemented differently in the simulation and the physical model, e.g. event-driven and time-driven. However, the physical model must be able to respond to the events in an event-driven simulation.

Benefits and disadvantages
The main advantage of the library approach is the clear separation of responsibilities between the simulation and the physical model. The expert maintains the physical model that is behind the library interface. If the interface is not changed, he can improve the physical model independently. The simulation engineer can integrate an improved physical model library into the simulation relatively easy, which improves the simulation.
Also a benefit of this approach is, because no source code is published, that it protects intellectual property.

Although this approach has a good utility, it can also have disadvantages.
The main risk is that the physical model might be reused for a purpose it was not intended for. One possibility is that the simulation engineer finds out that a function that he needs is actually not supported. Another possibility is that the physical model is used incorrectly, possibly invalidating the simulation results.
To mitigate such risks, it is recommended to elaborate the intended use and interfaces of the library early in de design of library model. And on the programming level, it is recommended to validate the preconditions on the input parameters.

2.2 Integrate the source code of a physical model
A different way to perform component reuse is to embed the component source code into the simulation. In this approach, see also Figure 2, the component is seen as a set of functions that can be used in the simulation. The functions of the physical model are clustered into several objects (components) that can be invoked by the simulation.

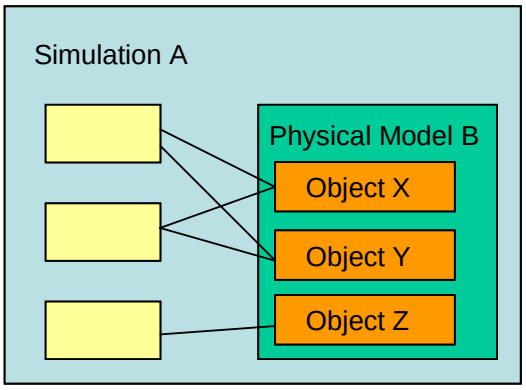

Figure 2: Integrate the source code of a physical model

The following aspects have to be taken into account when applying this integration approach.

1. *Interfaces*
   The tight integration of the two models makes it possible to have more interaction between these models. These interactions can take place at different fidelity levels in one simulation. They are facilitated by specific object interfaces.
2. *Validity domain and range*
   This aspect is the same as in the library approach.
3. *Knowledge*
   The simulation can use a physical model as a component, but modifications of this physical model need to be done by the experts of the physical model. Using components in this tightly integrated way requires a lot more knowledge of the integrator than in the library case. The integrator needs to have knowledge of the underlying physical description of the model B to integrate it correctly into simulation A.

Benefits and disadvantages
Benefits of this approach are: The timing aspect has to be taken into account. However, this is less of an issue than in the library approach. The tight integration allows the simulation to invoke the different steps/parts of the physical model when desired. Dealing with the timing aspect is easier than in the library approach.
Also more interactions between the two models are possible. For example the logging facilities of simulation A can be used for physical model B. This can also enable more analyses options for the expert who originally made the physical model B. Furthermore, parts of the physical model functionality can be improved by overriding existing functions. This enables multi-resolution implementations of parts of the physical model algorithms.

The disadvantage is that the improvement of the physical model after the integration step can generate challenges. This can generate the need for new agreements in the interfaces between simulation A and physical model B. Also the new knowledge in the physical model has to be taken into account by the software engineers. This can cause significant integration work. To do this adequately, the behavior and the influence of physical model B on simulation model A has to be clear.

2.3 Comparing the approaches
If only the output of the physical model matters for the simulation and the interface is transparent and the interface parameters are clear and limited in number then the “library approach” is advisable. The physical model is used in one setting and mostly at one fidelity level. If another fidelity level with another set of interface parameters needs to be used in the simulation, the physical model library is

replaced by another one. An example of this is given in Subsection 3.1.
In those cases where the detailed internal state of the physical model is needed by the simulation, the "source code integration approach" is more advisable. The local outcomes of the physical model can be used by the simulation. These outcomes are typically in the local memory of the physical model component. In the library approach these local outcomes are not accessible.
In the source code integration approach it is also possible that the different resolution levels require different interfaces and that different kinds of use of the physical model require different interfaces. An example of these different interfaces is given in Subsection 3.2.

3 Examples of multi-resolution models

This section shows two examples of multi-resolution models that are developed by TNO. First the missile modeling in the air and missile defense testbed (JROADS) is described ('library approach'). Second, the acoustic modeling in the Underwater Warfare Testbed (UWT) is shown ('source code approach').

3.1. Missile modeling in JROADS

The concept of multi-resolution modeling is used in the JROADS Standard Missile 3 (SM-3) modeling. In this section we will first introduce the JROADS simulation package and the SM-3 multi-stage ballistic interceptor. Then we will explain how the SM-3 model is linked with the JROADS simulation package and how configuration management is organized. The last part of this section will discuss the lessons learned during this integration process.

JROADS is a software suite for simulation of joint air and missile defense systems [3]. JROADS is capable of simulating a wide range of threats, surface warfare platforms, tactical communication links and support platforms. The development of JROADS is aimed at three main applications: 1) analysis on weapon system capabilities, 2) testbedding of (real-time) systems and 3) live exercise support.

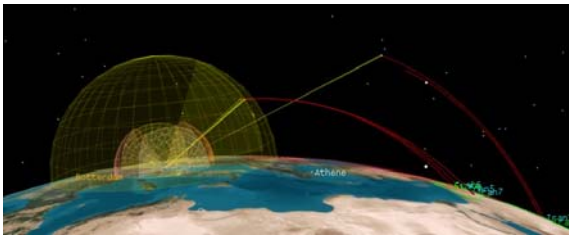

Figure 3: Impression of a JROADS simulation with SM-3

To support this wide variety of scenarios and applications, a very flexible and modular simulation architecture is required. A weapon system is assembled by combining the required subsystems such as radars, command & control, motion and launchers, where the level of fidelity is matched to the level of fidelity required for the application. For a given weapon subsystem, several implementations exist matching with a certain application. This approach enables the user to meet both the high fidelity requirements for analysis purposes as well as the real-time requirement in an exercise environment.

One example of this modular design is the interceptor modeling. The most basic interceptor fly-out that can be modeled is a constant velocity model without a motion model and constraint by altitude, ground range, cross range and downrange. This model will fly the interceptor straight to the target at constant velocity using ground truth and proportional guidance. This model can be enhanced by:

- an infrared-seeker, delivering a realistic target track
- a simple motion restricting the maneuvers by restricting G-forces
- a fly-out table giving maximum envelope and realistic time-on-target
- stage falloff, producing dropped stages at fixed flight times

Integration of the SM-3 model

This level of detail was not enough for analysis of a specific interceptor, the Standard Missile 3 (SM-3), produced by Raytheon. This interceptor is designed to defend against short to intermediate range ballistic missile threats during the midcourse phase of flight. The Royal Netherlands Navy (RNLN) has expressed its interest in acquiring these interceptors. Therefore detailed modeling is needed to assess the SM-3 capabilities and to support exercises like JPOW X with the RNLNs possible future SM-3 capability.

The SM-3 is an advanced weapon, different from the more common interceptors like the Patriot PAC-3 or Aster-30. The SM-3 consists of 4 stages and associated components: 1) a booster, 2) the dual thrust rocket motor, 3) the third stage rocket motor and 4) a kinetic warhead with infrared sensor. Guidance is based on data from the launching platform until the third stage is reached. From there, the SM-3 uses its infrared sensor to acquire target track. TNO developed a detailed model of the SM-3 which includes models of the aerodynamic properties, the engines of the stages, the guidance, the airframe and the sensor. This model is implemented in MatLAB with the use of SimuLink.

The MatLAB/SimuLink model of the SM-3 was compiled to a Dynamic Link Library (DLL) using MathWorks' Real Time Toolbox. JROADS is written in Java, and the DLL is accessed by the Java Native Interface (JNI). When a SM-3 interceptor instance is used in JROADS, an instance of the DLL is created and fed with JROADS track data. Also a platform object is created in JROADS, representing the missile-in-flight. The position and course of this platform is acquired from the DLL library.

From the viewpoint of the JROADS model, there is no interface difference between using the internal interceptor model or the detailed SM-3 model implemented in the DLL. It can be accessed as a platform with a position and course, accepts events such as new track data and send events such as an intercept event. Changing between the internal model and the detailed model is a matter of selecting the desired model in the scenario.

Benefits and disadvantages

The integration of the SM-3 model enables JROADS to perform realistic intercepts with this type of missile. This would not have been possible without the integration. Due to the clear and easy interface, the integration was realized in a very short implementation time.

During the implementation and use of this library link some issues appeared, which will be discussed:

1. Mismatch between SM-3 DLL design and the usage by JROADS,
2. Lack of a validation step to ensure the JROADS/SM-3 integration performs as expected,
3. Dependence on support outside the JROADS team.

The first issue arrived during the usage of the SM-3 model in the calculation of engagement solutions. In this calculation, an SM-3 is virtually flown to the target to determine flight-time, launch time and intercept time. The process of calculating the best engagement option is iterative, so it results in many virtual missile flights. The performance of the SM-3 model was not adequate to perform these calculations in real-time. This issue was solved by constructing a lookup table with the downrange, cross range and altitude at the axis and the flight time as data. This fly out table was generated offline.

Also, the SM-3 model is used during missile flight to refine the guidance instructions. In the same way an engagement solution is determined, JROADS creates a virtual missile in this calculation. This requires simulation and instantiation of a missile-in-flight. The SM-3 model is only capable of simulating missiles from launch time and not given a certain state. This was solved on our request in a later version of the model.

The second issue, a missing validation step, was identified during the first mission planning sessions with the SM-3 capability. We found several cases where JROADS was not capable of calculating an engagement option while the intercept position was well within the envelope of the interceptor. Reviewing the cases showed a difference in MatLAB and JROADS results. The fact that this difference was found late in the process indicates that a validation step during integration was missing earlier in the integration process. This validation step has now been added and implemented by comparing fly out tables of JROADS and the MatLAB SM-3 model.

The last issue is inherent to (re)use of externally provided software components. By (re)using software, one becomes dependent on the provider. In this case the dependency is twofold; not only for the delivery and maintenance of the software itself, but also regarding the domain knowledge behind it. During analyses, we must be able to investigate every part of the modeling to determine the model behaviour and to assess scenario results. When the interceptor is a key component in an analysis, support from the SM-3 model builders has to be arranged. This is an extra constraint, especially during long out-of-area military exercises.

3.2. Acoustic underwater warfare modeling

In 1994 TNO started the development of the Underwater Warfare Testbed (UWT) [4][5][6]. This testbed simulates underwater defense systems like sonars and mines, see also Figure 4. These systems observe the environment by listening to transmitted acoustic signals (e.g. engine noise, sonar pulses). All signals propagate through the environment and are attenuated and scattered by the sea surface, bottom, and other reflecting objects (e.g. submarines). This propagation is modeled by the Acoustic Loss Model of Operational Tasks and Studies (ALMOST) [5][7], of which the development started in 1980. This model computes the received signal levels at specified locations and also determines time delays that are due to the sound speed in the water, and bending of the sound waves through the water.

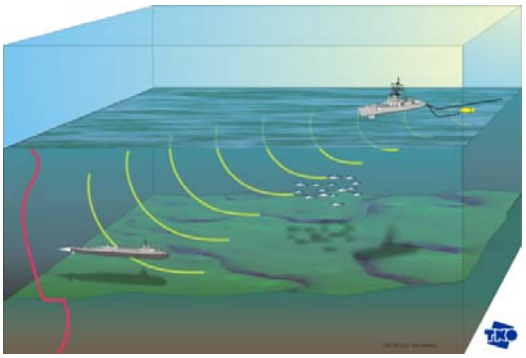

Figure 4: Conceptual view of the UWT

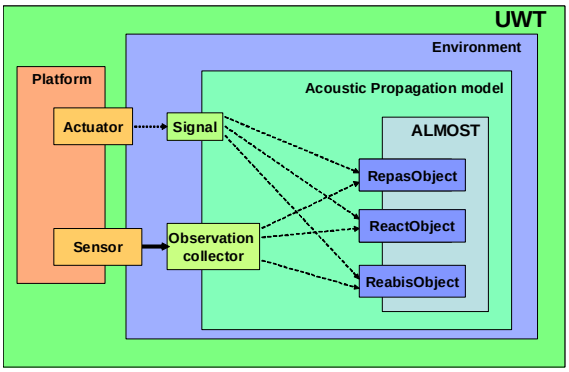

Figure 5: Model structure of the UWT

The ALMOST model is developed in Fortran'77 and includes a large set of functions that can be used to compute different aspects of the sonar equation (e.g. received noise levels, ambient noise levels, reverberation, echoes). All functions are available in a library which is used in several stand-alone applications within TNO.

Embedding use of the acoustic model

To facilitate the programmers that incorporate the model, all functions are embedded in three C++ objects: 1) Repas (for passive and intercept sonar), 2) React (for mono-static active sonars), 3) Reabis (for bi-static active sonars). These objects provide a user-friendly interface to the acoustic model and they hide details that need to be known to use the Fortran code in the right order.

The Repas object is used to compute the propagation loss for passive and intercept sonars. These sonars do not transmit signals but only observe the environment. The React object is used when one sonar is transmitting and receiving its own signals. The Reabis object is only used when the transmitted signals are received by another sonar in the environment, which is only the case in more complex simulations.

Within the UWT the propagation of the signals is modeled by the Environment object. This object is responsible for collecting and managing all signals that are produced in the environment. It also handles all reflections that are introduced by the reflecting objects in the environment. In the testbed sensors are introduced to observe the signals in the environment. Furthermore, the environment has the notion of the location time. This enables it to remove the signals when the levels drop below the background level of the noise. The environment uses the acoustic propagation model to build the bridge between the UWT and the ALMOST objects.

The computation of the received signal levels can be performed with different levels of fidelity. Especially in Monte-Carlo simulations, the fidelity of the acoustic modeling is chosen to be low, because otherwise the simulations last to long. In those cases the complexity of the calculations is reduced by parameter settings in the environment. This environment will then use derived classes of the Repas, React, and Reabis objects. These derived objects implement the propagation with a lower fidelity. Some of the functions in the ALMOST Fortran code are still used (e.g. calculations of time delays) but some are replaced by tables (e.g. propagation loss) or computed in a faster way.

A different way of speeding up can be achieved by replacing the sensor with another less detailed sensor. For example a sensor which does not interact with the environment but only uses predefined detection distances to model the sensor performance. This can only be used when the required fidelity is low or when the operational conditions of the sensor can be computed on forehand.

Testing of the simulation and acoustic model can be done after integration and also can be done separately. Each of the objects (i.e. RepasObject) is used to build a separate executable that can be executed using the same input as is used within the simulation environment. This enables detailed testing and maintenance by the domain experts and also by the simulation builders.

Benefits and disadvantages

There are many benefits of this way of embedding of the acoustic model in the simulation. First of all, the implementation is very fast because all code is compiled in one executable. In the past this was accomplished by an automated Fortran to C translator, but the current compilers can link the C++ and Fortran code directly. Second, the acoustic model developer can use his own favorite programming language and can update the implementation separately. Third, the embedding

of the model also enables to create other models with other fidelity that can be used easily in the testbed. Fourth, by creating the object wrappers (embedding the functional code in objects) around the acoustic model, the sequence in which functions are called can be maintained easily. Also the internal (intermediate) data can be stored in the object which makes it easier to use the model. By introducing derived classes, the fidelity of the modeling can be modified without changing the structure of the model. Within derived classes also classified models can be implemented which need not be available in all applications of the testbed.

If the library approach was used here, this would have raised problems. The current three objects not only act as a façade, they also contain state. In this way they provide a convenient interface for the simulation engineer.
If the library approach was used here, the library would contain many low-level functions. These low-level functions would all need to be used correctly by the simulation engineer and in the correct order. This would put a huge burden on the simulation engineer.
Therefore it was decided to use the source code integration approach here. This approach resulted in a suitable interfacing between the physical model and the rest of the simulation.

The difficulty of this approach is maintainability of the physical model code; good version management is required. Furthermore, the interfaces (to the ALMOST objects) that are used by all applications need to be managed carefully. Also the associated interface and model documentation is very important. Because the ALMOST objects can be seen as a black-box, the validity boundaries need to be checked to avoid misuse of the model by less experienced users.

Conclusion and recommendations
It can be concluded that the embedding of functional Fortran code is well possible and facilitates the use of physical models in simulations. A good design of the interface to the model is very important to support future extendibility and maintainability of the code. By means of inheritance of the interface, multi-fidelity applications can be built. It also enables multi-resolution simulations in which parts of the simulations are done with higher/lower fidelity.

4 Guidelines
This section will describe the most important guidelines for reuse of (multi-resolution) physical models in simulations.

A first step for an organization is to determine which kind of tools or assets it wants to aim for. Only those tools or assets should be considered as reusable components that provide a clear benefit or advantage to the organization.
Besides, the organization's policies and processes need to be defined and organizational prerequisites need to be arranged. They can be based on the results of NATO MSG-042 "Definition of a 'Framework for Simulation Resources Reusability' (FSRR)", in particular the proposed Business Model [8], [9].

For an organization to get the most benefit out of reuse it is best practice to make an explicit separation between the creators (developers) of the component and the user who utilizes it in his simulation [10]. This gives the model developer the freedom of using his favorite programming language and updating the implementation when necessary, however under the constraints of creating reusable code. The simulation developer, who utilizes the model, has the opportunity to use a well maintained and documented model so he does not need to have specialized domain knowledge.
To provide the users with knowledge of the available reusable models, a support team needs to exist. This team manages the assets, provides user support and training, performs maintenance of assets and provides coordinated feedback on problems. The support team has a managed repository that contains all software components available for reuse.
Finally the management has to be aware of their role in the organization with respect to reuse. They have to ensure that the overall process proceeds efficiently, that the construction and support of new assets is funded, and that conflicts between the goals and schedules of the different groups are resolved: there is tension between simulation development objectives (especially capabilities and schedule) and model development and evolution (especially long-term reusability and quality).
In Figure 6, the functions in an organization for reuse are displayed in their coherence. Ideally, all members of an organization have a shared reuse mindset.

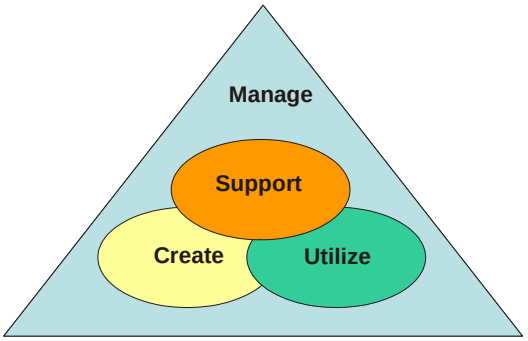

Figure 6: Functions in an organization ideal for reuse

The guidelines for creating reusable software are organized for each software development phase, see also Figure 7.

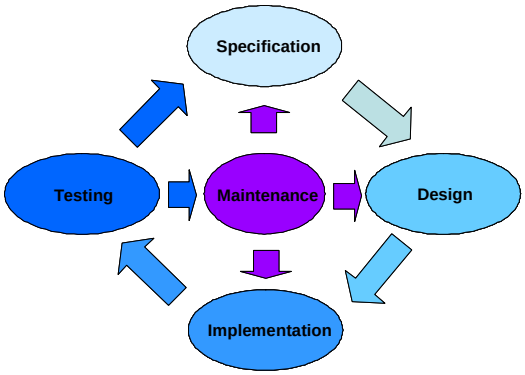

Figure 7: Software development phases

During the concept and specification phase it is important to describe the model boundaries and also investigate possible future extensions and applications of the model. This is the basis for the design of the reusable model component therefore in this phase there is a huge interaction between the support, create and utilizer communities.

In the design phase of the simulation and model, it is important to identify model components that can be developed separately and have a clearly defined interface which can be reused in other applications. The interface should contain all information that could be necessary in the future for the implementation of a higher fidelity model. Furthermore, it is always important to separate the (graphical) user-interface from the model and use known design/architectural patterns to link them. In this phase it is also important to describe how the multi-resolution functionality is modeled (e.g. by input parameters or inheritance of the model interface). Also design patterns can be used to support the integration (e.g. Strategy, bridge, proxy, façade patterns). Most of de design is done by the creators.

The selection of the programming language is decided in the implementation phase. In most cases this is determined by the preferences and language knowledge of the scientist developing the model. It is preferred that the selected language can be linked directly to the most likely target simulations. The simulator might be written in another language but preferably in a language that can link easily to other languages/libraries and is platform independent. In this phase it is also essential to document the interface and the use of the physical model. Also event and error handling of the model needs to be solved. This is done by the creators.

Testing of the integration of the physical model and simulation is very important. Especially a good and complete scenario set is essential. Besides testing of the model, the interface needs to be tested separately. Also misuse of the input should be checked in the interface. The support group takes care of testing and assists the users in the integration.

During the maintenance phase, the model, simulation, scenarios and all software documentation should be maintained at one central location. Also the model documentation and knowledge should be secured, and the support during the use of the model should be organized. For this purpose the support group has a repository at its disposal.

For the organization it is preferred that simulation environments are developed with similar software architectures. Having (many) simulation environments that have different architectures makes it difficult and time-consuming to reuse model components. So, preferably all simulations are using similar simulation kernels and architectures (using the same interface and modeling philosophy).

5 Conclusion and recommendations

This paper has presented our general approach to the reuse of (multi-resolution) physical models in simulations. Such reuse generally comes down to linking a physical model to a simulation.

We have identified two possible approaches to achieve this goal, both of which are based on real-life cases. The descriptions of each of these cases highlight how the linking is achieved and how multi-resolution aspects are dealt with.

The first approach uses a library (DLL) to link a physical model to a simulation, which provides a clear separation of concerns. The physical model is developed and maintained by the subject matter expert, while the simulation engineer views the physical model as a black box with a clear interface. The mechanism for switching model resolution is by replacing one library with a suitable other one.

The second approach uses source code integration to link a physical model to a simulation, which requires close cooperation between the physicists and the simulation engineer. Such cooperation requires a common ‘language’ and understanding a part of the other problem domain, and the willingness to constructively develop a successful integration. A suitable mechanism for switching model resolution is to use special-purpose parameters on the physical model interface. The agreements about the interface between the physical model and the simulation needs to be transparent and registered.

Common guidelines have been formulated for both linking approaches. These guidelines mainly consider technical aspects of software engineering. However, we have also found that other aspects, such as organizational aspects, are of prime importance, and therefore we also included guidelines that concern organizational aspects.
We recommend that these guidelines are implemented in a process that facilitates physical model reuse and that ensures software quality. For that purpose we have organized our guidelines according to the different software development phases, making our guidelines practically usable.

In the near future, we intend to work on achieving 'effective realism', i.e. the level of fidelity that is required to achieve the goals of the simulation. In other words, we will give attention to the matching of the fidelity that is required with the fidelity that is implemented. Multi-resolution models are very helpful to facilitate the selection of the actual fidelity in simulations for analysis and training purposes.
Furthermore, we plan to further investigate other ways of linking existing physical models to simulations to achieve effective and efficient reuse of (multi-resolution) physical models.

6 References

[1] K.J. de Kraker, M.van Emmerik, P. Langeslag, W. Huiskamp, Towards successful multifunctional reuse of simulation components, *07S-SIW-051*, 2007.

[2] D. Keus, M. Kamstra, K.J. de Kraker, Experiences from a Multi-Resolution Modeling case study using Model Driven Architecture, *08S-SIW-011*, 2008.

[3] W. van der Wiel, J-ROADS Air Defence Simulation Support during the 2006 JPOW IX Missile Defence Exercise, *NATO MSG-045 Symposium on "Transforming Training and Experimentation through Modelling and Simulation*", Rome, Italy, 2006.

[4] Lentze, S, MOSES, development of an Underwater Warfare Testbed, *UDT Europe*, 2001.

[5] P. Schippers, F.P.A. Benders, Acoustic modelling and simulation of next generation torpedoes, *UDT Europe*, 2003.

[6] F.P.A. Benders, R.R. Witberg, H.J. Grootendorst, Torpedo and countermeasures modelling in the Torpedo Defence System Testbed, *UDT Europe*, 2004.

[7] P. Schippers, REACT - a model for Active Sonar Range Predictions, *UDT Europe*, 1991

[8] Robert Elliott, Stefan Franzen, Lt. Sabas González Godoy, Bernardo Martínez Rei, Xavier Lecinq, David Edmondson, Wim Huiskamp, Wanda Wharton, Lana McGlynn, Cdr. Ángel San José Martín, FINAL REPORT MSG-042 Definition of a 'Framework for Simulation Resources Reusability' (FSRR)

[9] LtCdr. Ángel E. San José Martín (ESP-NAVY), Lt. Sabas González-Godoy (ESP-ARMY), Wim Huiskamp, Bernardo Martínez Reif (ISDEFE) et al, "NATO MSG – 042 Definition of a Framework for Simulation Resources Reusability (FSRR)". Paper 2685 Published in Proceedings of I/ITSEC 2006, Orlando.

[10] Systematic Software Reuse:Architecture, Process and Organization are Crucial, Martin L. Griss, *Fusion Newsletter*, Oct 1996

Author Biographies

DANIELLE KEUS is a member of the scientific staff in the Defence, Security and Safety Division at TNO. She is a MSc. in Electrical Engineering from Delft University of Technology. She has a background in software engineering, electrical engineering. Currently she is involved in various simulation projects in the areas of underwater warfare, distributed intelligence, sea based operations and modeling physical objects for simulations.

FRANK BENDERS is a member of the scientific staff in the Defence, Security and Safety Division at TNO. He has a PDEng in Software Engineering from the Stan Akkermans Institute Eindhoven. He has a MSc. in Electrical Engineering from Eindhoven University. He is involved in various projects in the area of underwater warfare and simulation, underwater acoustics and marine mammal research.

DR. KLAAS JAN DE KRAKER is a member of the scientific staff in the Defence, Security and Safety Division at TNO. He holds a Ph.D. in Computer Science from Delft University of Technology. He has a background in Computer-Aided Design and Manufacturing, collaboration applications, software engineering (methodologies), meta modeling and data modeling. Currently he is leading various simulation projects in the areas of simulation based performance assessment, collective mission simulation, multifunctional simulation and serious gaming.

ROB VAN DER MEER is a member of the scientific staff in the Defence, Security and Safety Division at TNO. He is a MSc. in Electrical Engineering from Delft University of Technology. He has a background in biomedical engineering and embedded systems. He is currently involved in air defence research.

PETER LANGESLAG is a member of the scientific staff in the Defence, Security and Safety Division at TNO. As a research engineer he has a broad experience in the architectural design of simulators and the procurement and development of training simulators for the Dutch army. Peter supported the Royal Netherlands Army Simulation Expert Centre in the implementation of a Modelling and Simulation Repository. He also contributed to the MSG-003 Feasibility Study on Modelling & Simulation Technology in Support of Simulation Based Acquisition (SBA) and was a member of the MSG-030 task group on Collaborative Working Environments (CWEs) for Simulation Based Acquisition (SBA).