Project:	**D1.Project1**
Name:	**Building Simulator Simple1**
Document:	**Interface**
Responsible:	**Barbara, Martin, Dirk**
File-Pathname	**irz1:~sfb501/requirement_specification/interface.v2.1**

Type of change	Date	Version	Person
Start	03/21/95	1.0	Barbara/Martin/Dirk
New	05/05/95	2.0	Barbara/Martin/Dirk
Minor changes	15/05/95	2.1	Barbara/Martin/Dirk
Addtion of text	22/05/1995	2.1a	Dirk

Specification of the interface

1. Problem Statement

The interface serves as a distribution unit which takes inputs from the streams and distributes them to its own parts, the kernel. The outputs from the kernel, the object and interface parts are written onto streams.

2. Dictionary

3. Concept Model

The top-level concepts of the problem domain are depicted in figure 3.0.1 (all arrows are standing for relations).

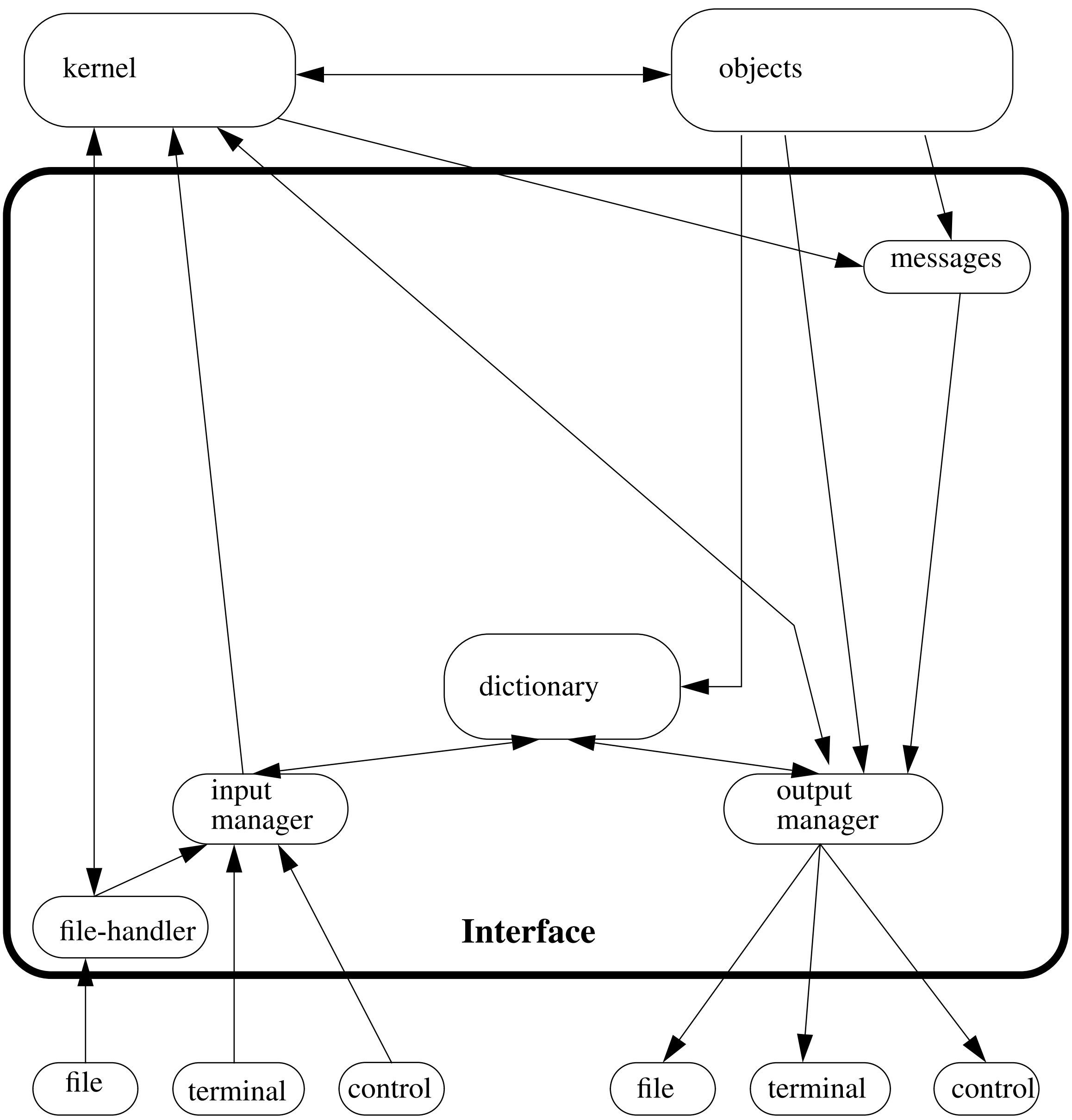

Fig. 3.0.1: Concept Model

4. Specialization Hierarchies

5. Context

5.1. Context Diagram

The context is defined in figure 5.1.1.

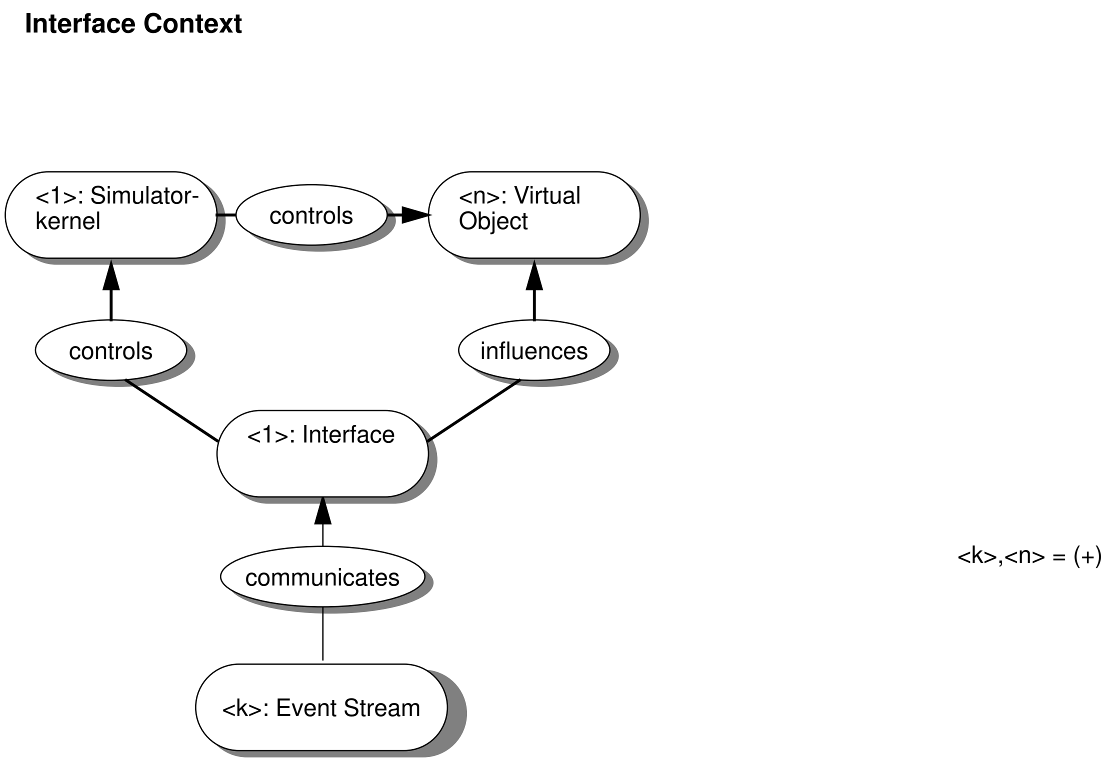

Fig. 5.1.1: Interface Context Diagram

5.2. Message-Formats

This section defines some of the messages needed. Additional definitions can be found in the requirements-specification document and in the kernel document.

Capitalized words denote terminal symbols. All non terminal symbols not described in the requirements specification are defined as follows:

Init-Format:

init parameter := {simulation interval, slack, timing factor, logging flag}
init event := (event type, prio, obj)

Error-Format:

error := string x gravity
gravity := {light, fatal}

Event-Format:

event := (event type, time, prio, obj-ref)

event type := {SET, GET, various application-defined simulate event types, HALT, STOP}
iprio := N

Event-Subtypes:

set-event := (SET, time, PSET2, obj-ref, value)
get-event := (GET, time, PGET1, obj-ref, dest)
stop-event := (STOP, time, PSTOP, 0)
halt-event := (HALT, time, PHALT, 0)
read-event := (READ, time, PSTOP, interface-reference)

Ngets-Format:

ngets := N

Logdata-Format:

log-data := string

Trigger-Format:

trigger := read-event

5.3. Inputs and outputs between interface and kernel

The following figure describes the messages exchanged between the simulators interface and the simulators kernel.

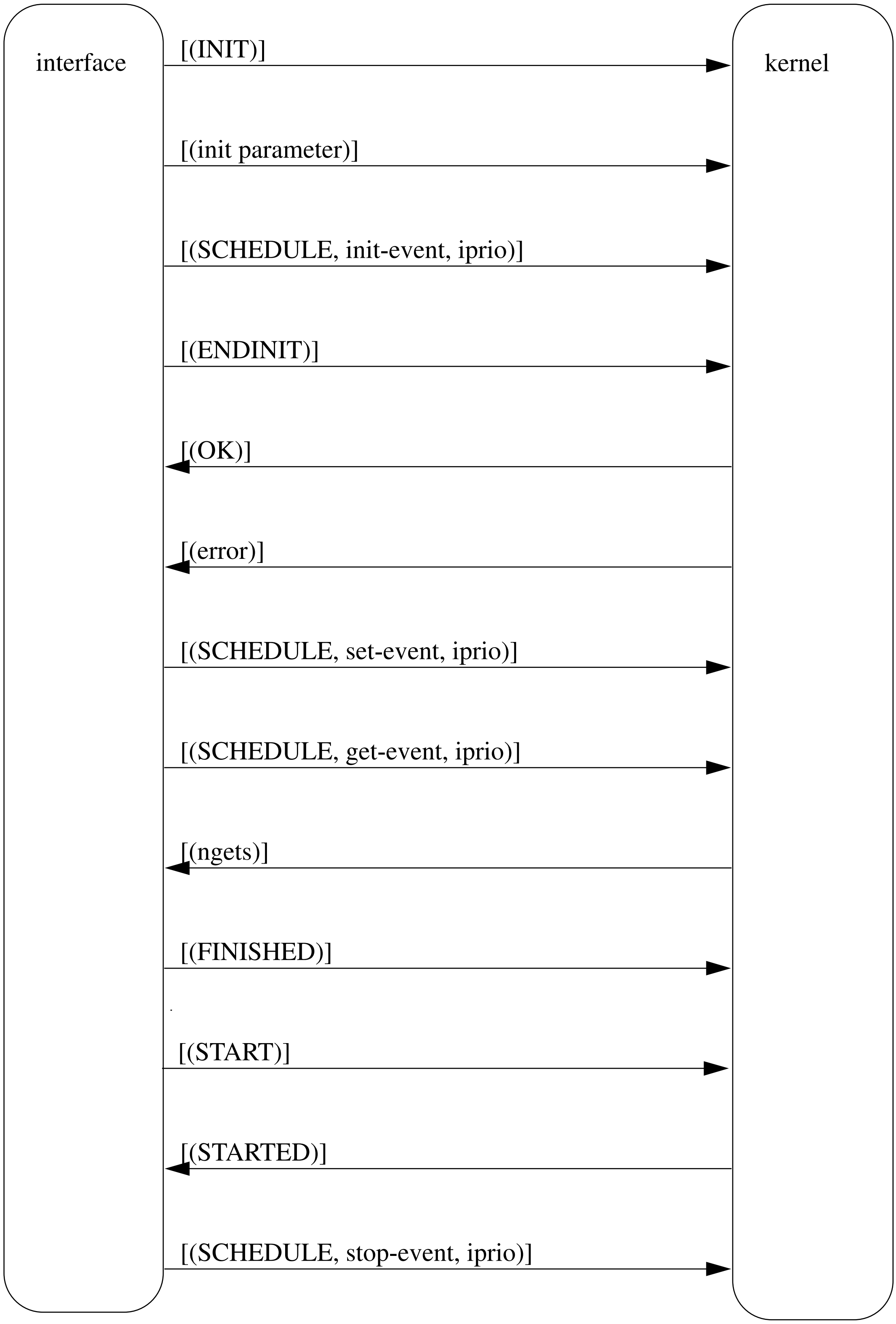

Fig. 5.3.1: Messages between interface and kernel I

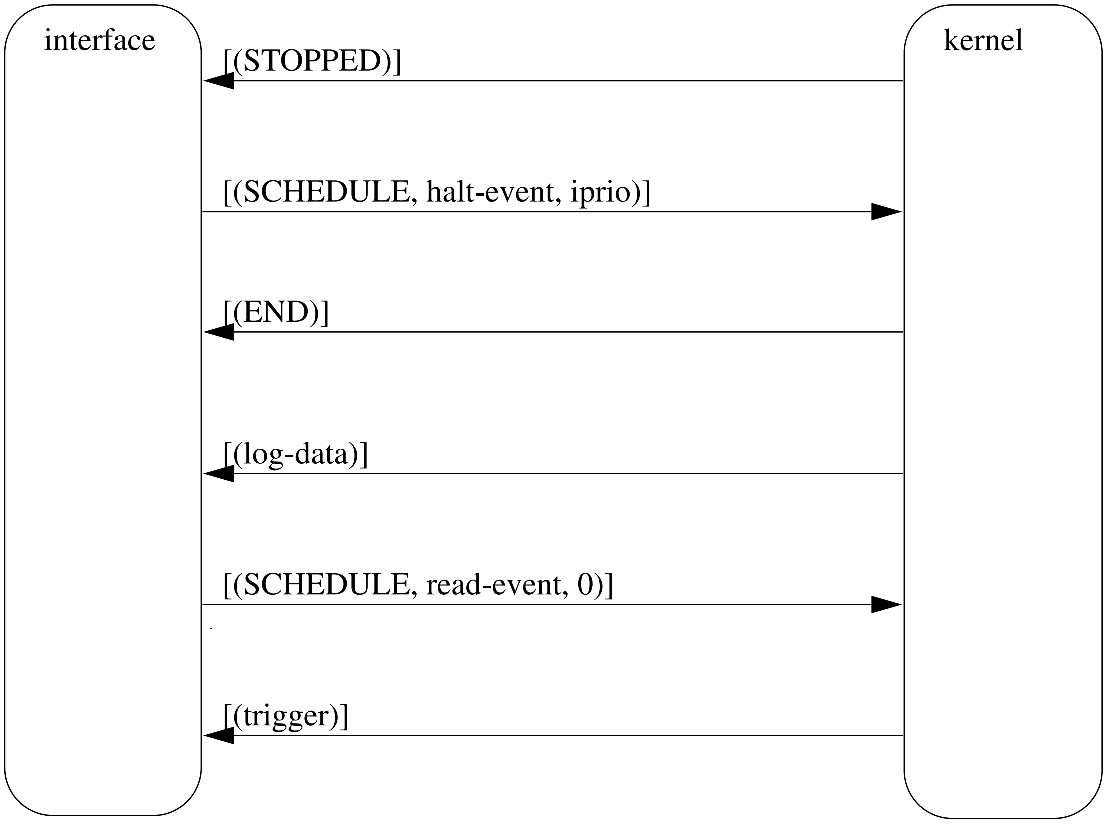

Fig. 5.3.2: Messages between interface and kernel II

5.4. Inputs and outputs between interface and object

The following figure describes the messages exchanged between the simulators interface and the object.

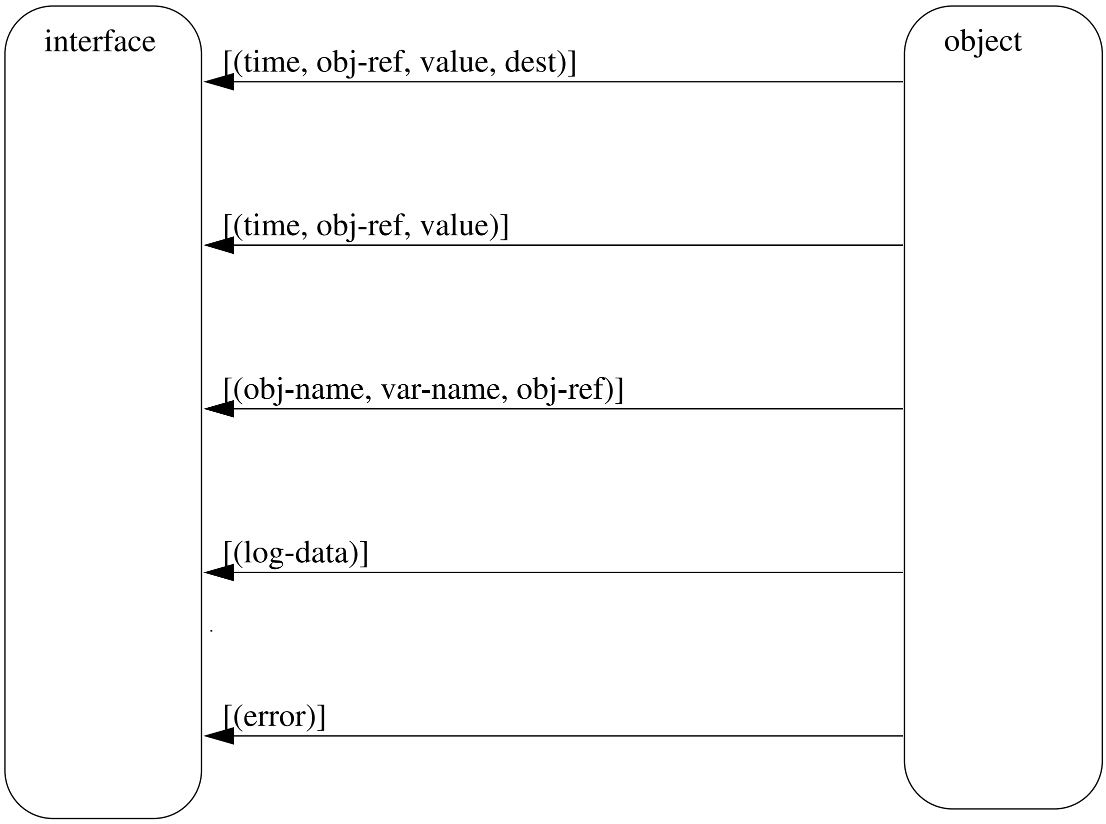

Fig. 5.4.1: Messages between interface and object

5.5. Inputs and outputs between interface and stream

The following figure describes the messages exchanged between the simulators interface and the stream.

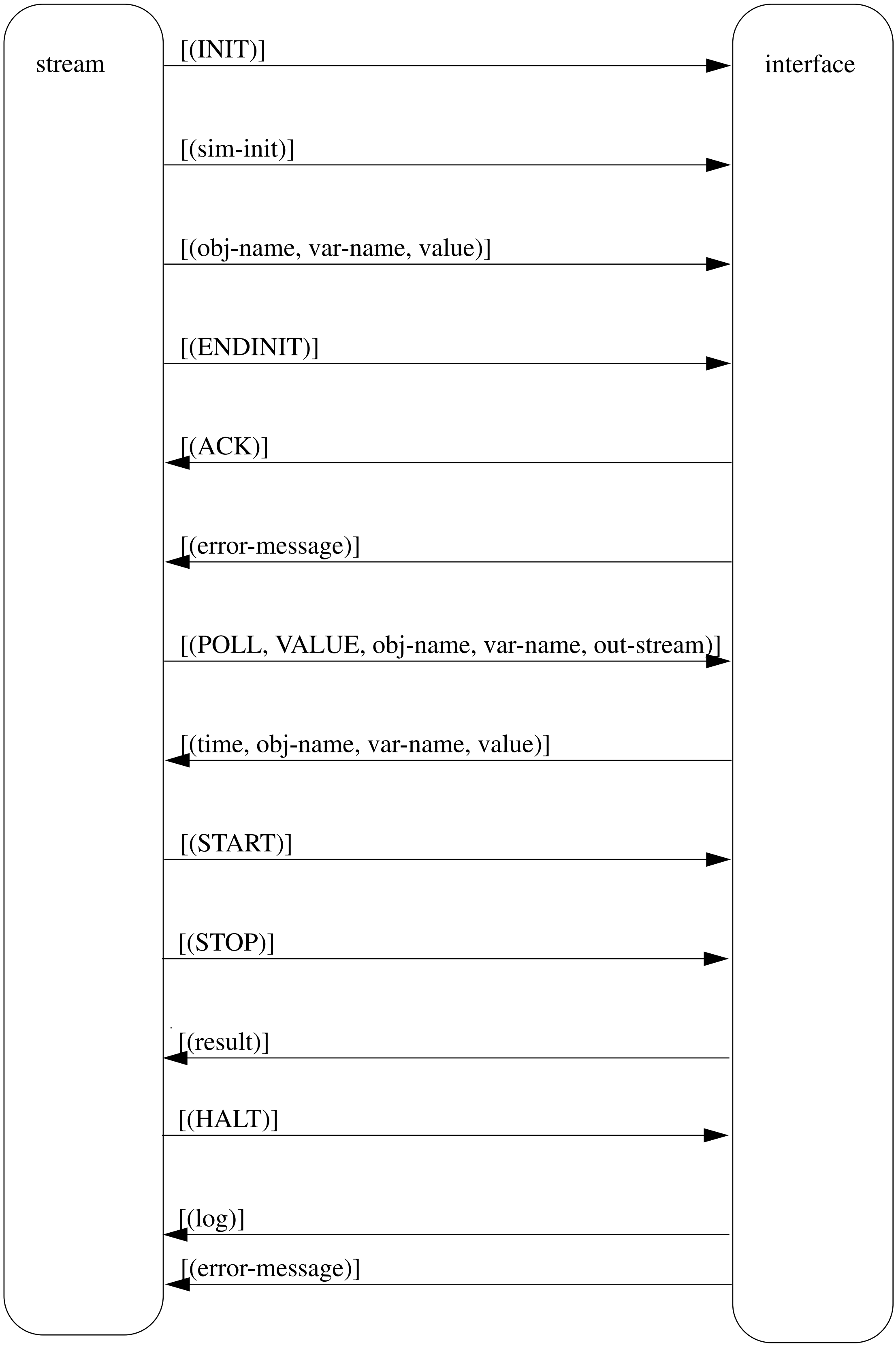

Fig. 5.5.1: Messages between stream and interface

6. Message Sequence Charts

The following message sequence charts contain the messages described in the previous chapter. They describe the sequence in time of the different messages.

6.1. Registration Phase of the Objects

The registration of any object is accomplished by sending the message Register with the parameters obj-name (the objects-name), var-name (the variables name if it is a variable) and the objects reference. All triples are stored in a database in the interface.

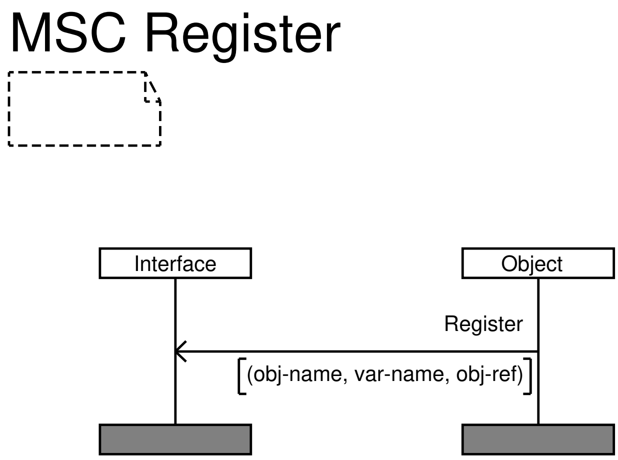

Fig. 6.1.1: MSC Register

6.2. Initialization

The following MSC describes the sequence of messages exchanged during a successful initialization. After passing the Message Init from the stream to the kernel, one ore more init-parameters are read and eventually passed to the kernel. Any init-parameter concerning the kernel is sent to the latter. Any init-parameter concerning the interface is distributed internally. The init-parameters sent to the kernel are translated from stream-format to kernel format. After those init-parameters zero ore more set-value messages are taken from the stream and sent to the kernel after being transformed according to the message formats. The initialization phase is terminated with the message end-init which also is sent to the kernel. In case of a successful initialization a positive acknowledge from the kernel is expected. In case of no error inside the interface, an acknowledge is sent to an appropriate output-stream.

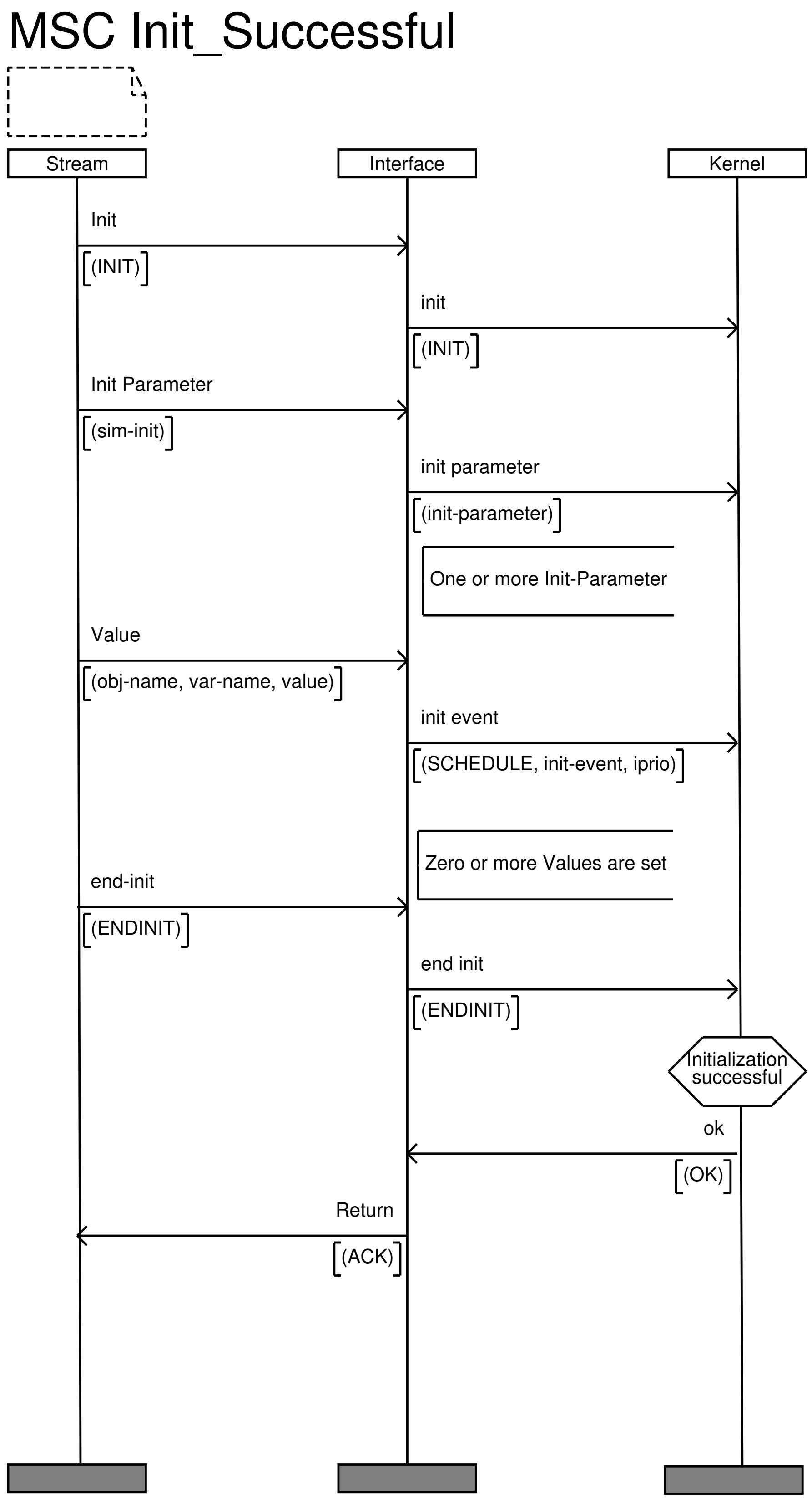

Fig. 6.2.1: MSC Init_Successful

6.3. Set

Any setting of a new value of an object is taken from the stream and sent to the kernel. The format is transformed and external references are decoded to internal ones.

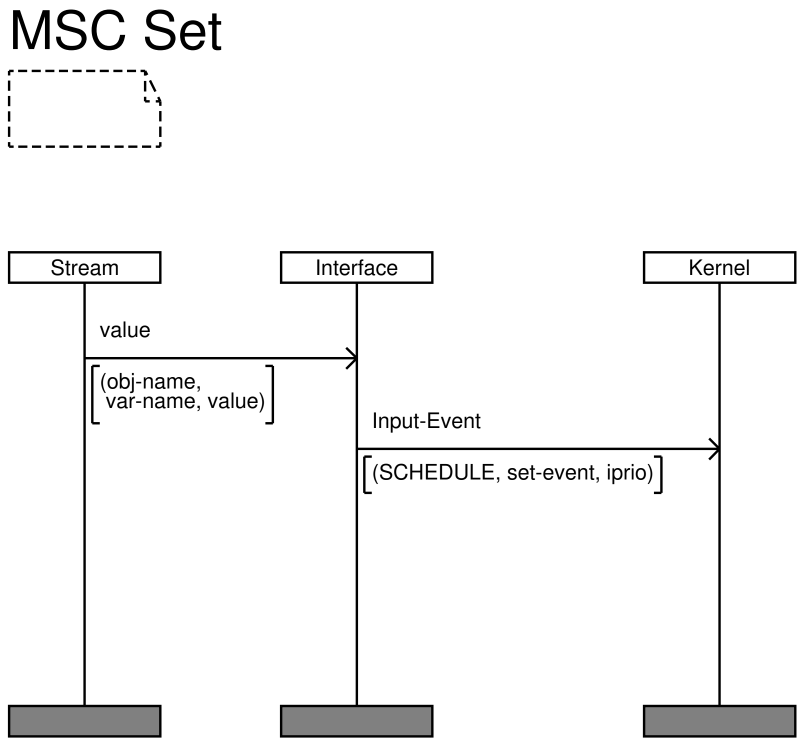

Fig. 6.3.1: MSC Set

6.4. Get

The request of an objects value is handled as follows. One ore more poll-value commands are taken from the stream, transformed and sent to the kernel. At the appropriate time, the kernel sends the number of values, which will shall be written to a stream. Then a number of values will be sent from different objects, transformed and written to the appropriate streams. If the number of value-messages sent by the objects equals the number of values to be written, the kernel will be informed by the interface with the message FINISHED.

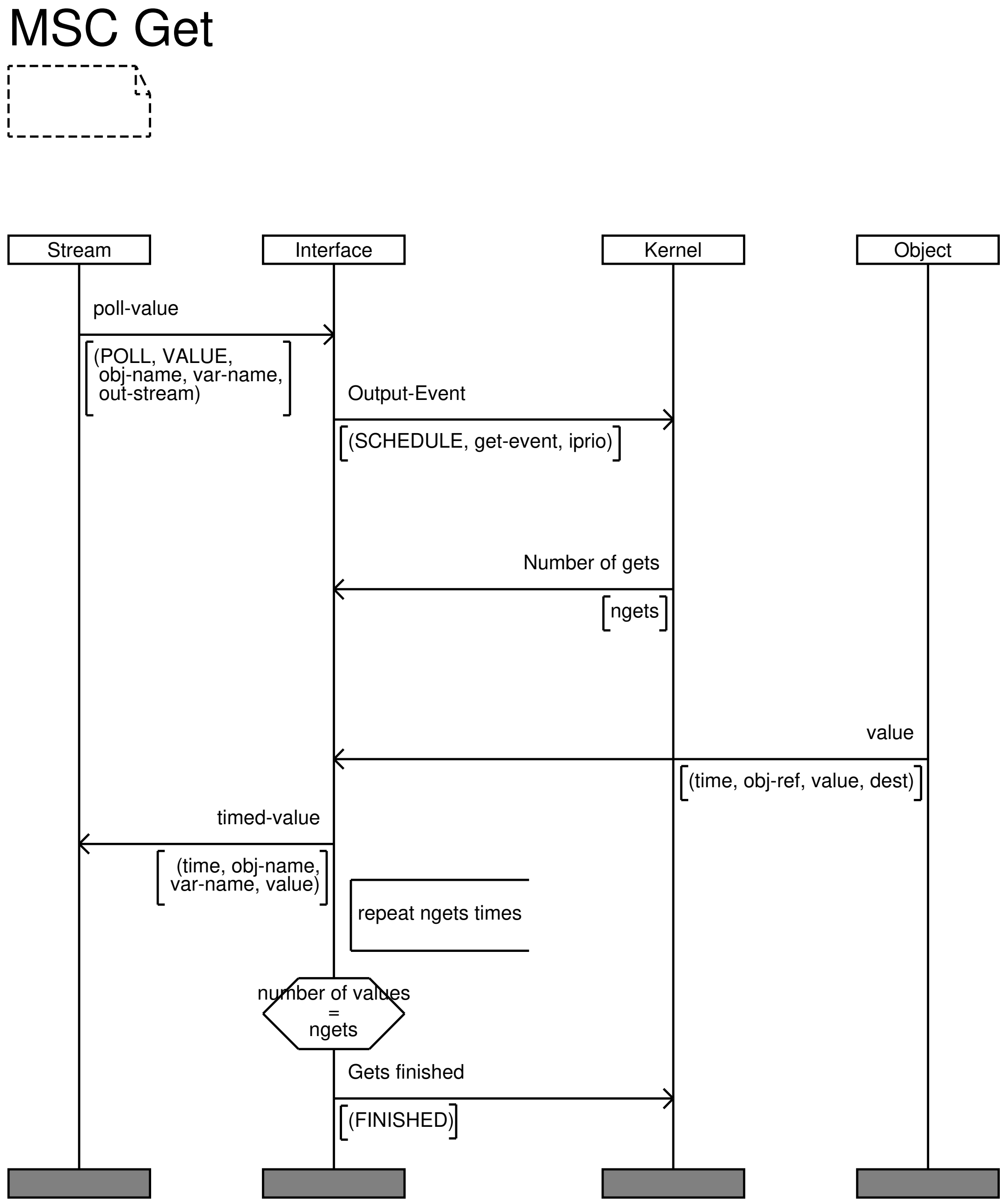

Fig. 6.4.1: MSC Get

If the value of an object changes and the changed-value-flag is set, a message is sent by the object, transformed and a value written to the appropriate stream.

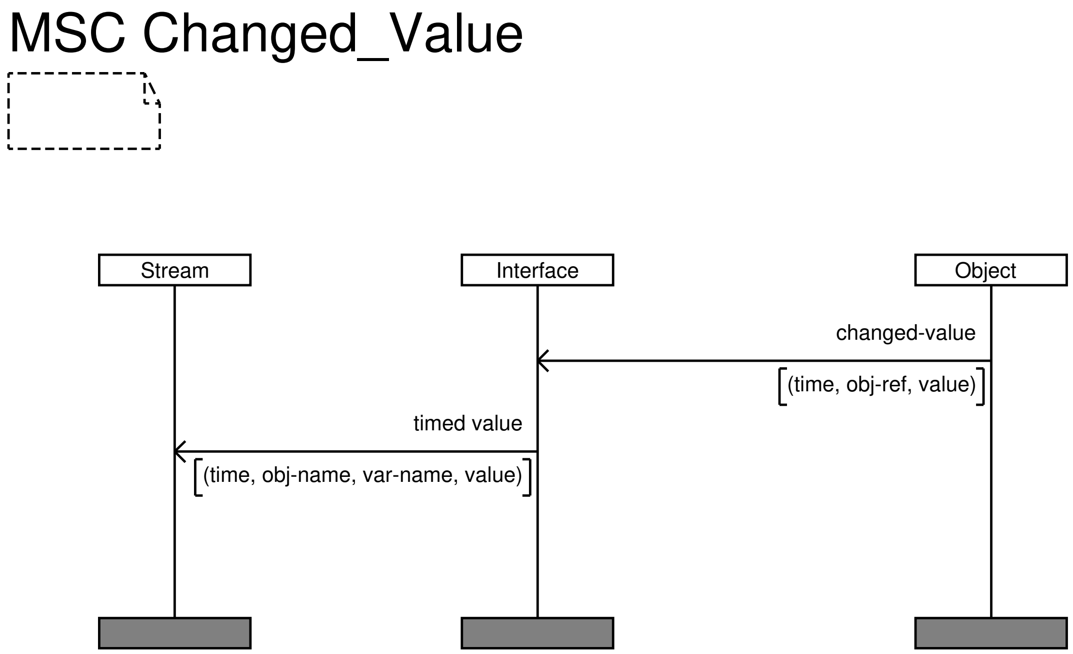

Fig. 6.4.2: MSC Changed Value

6.5. Simulator Commands

The start command changes the state from idle to running. It is taken from a stream and sent to the kernel. An acknowledge from the kernel is expected by the interface. In case of a successful start, an acknowledge is sent to an appropriate stream.

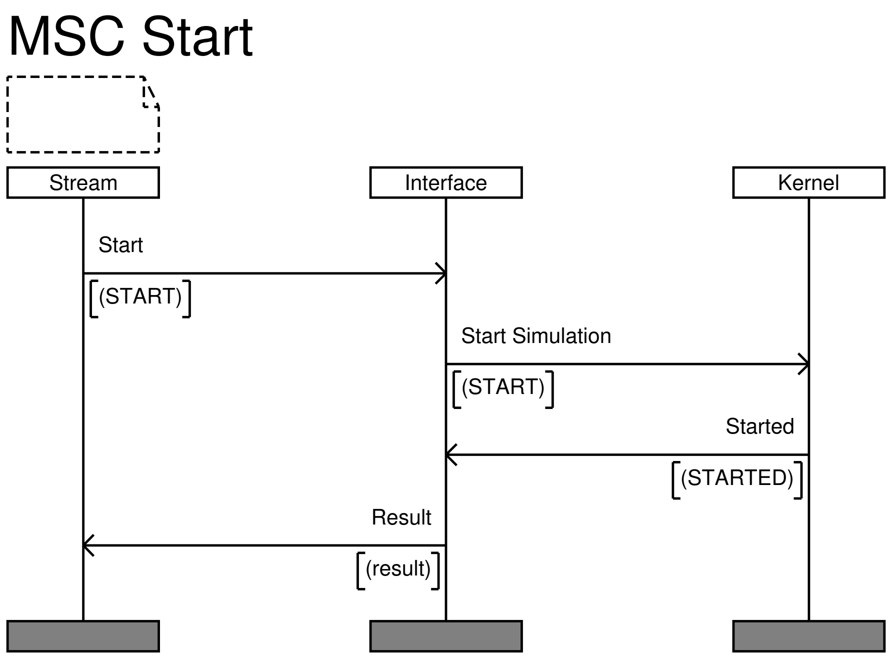

Fig. 6.5.1: MSC Start

The stop command is taken from the stream and sent to the kernel. When the time for the stop command is reached the message stopped is sent from the kernel and transmitted to an appropriate stream.

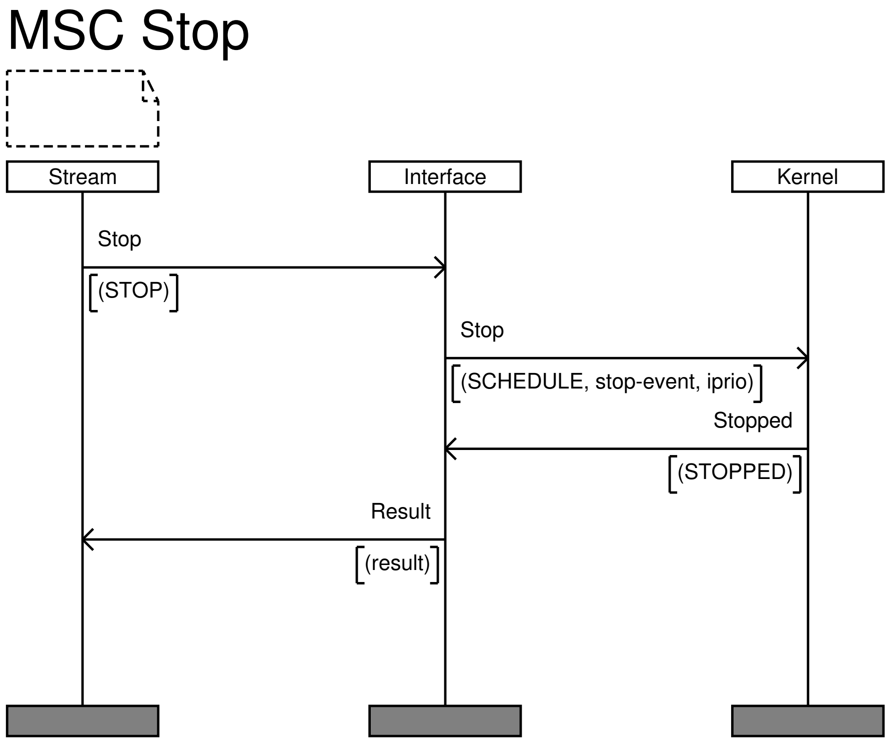

Fig. 6.5.2: MSC Stop

The halt command is taken from the stream and sent to the kernel. When the time for the halt command is reached the message Simulation ended is sent from the kernel to the interface and transmitted to an appropriate stream. The same message will be sent by the kernel, when the ending time has arrived.

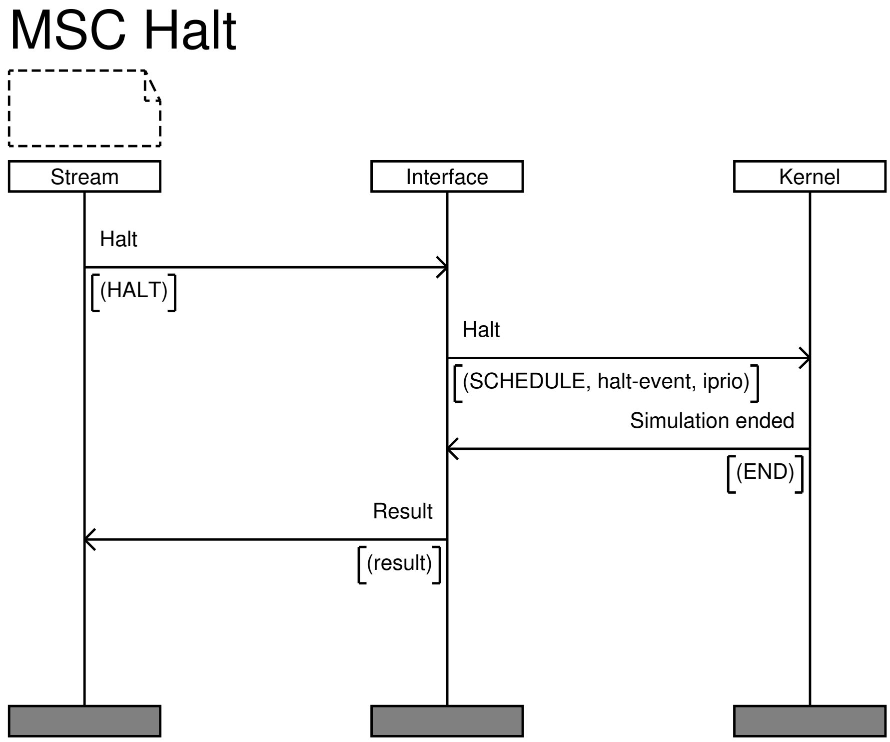

Fig. 6.5.3: MSC Halt

6.6. Logging

Logging is achieved by receiving a log message from either the kernel or any object. The log is written to an appropriate stream.

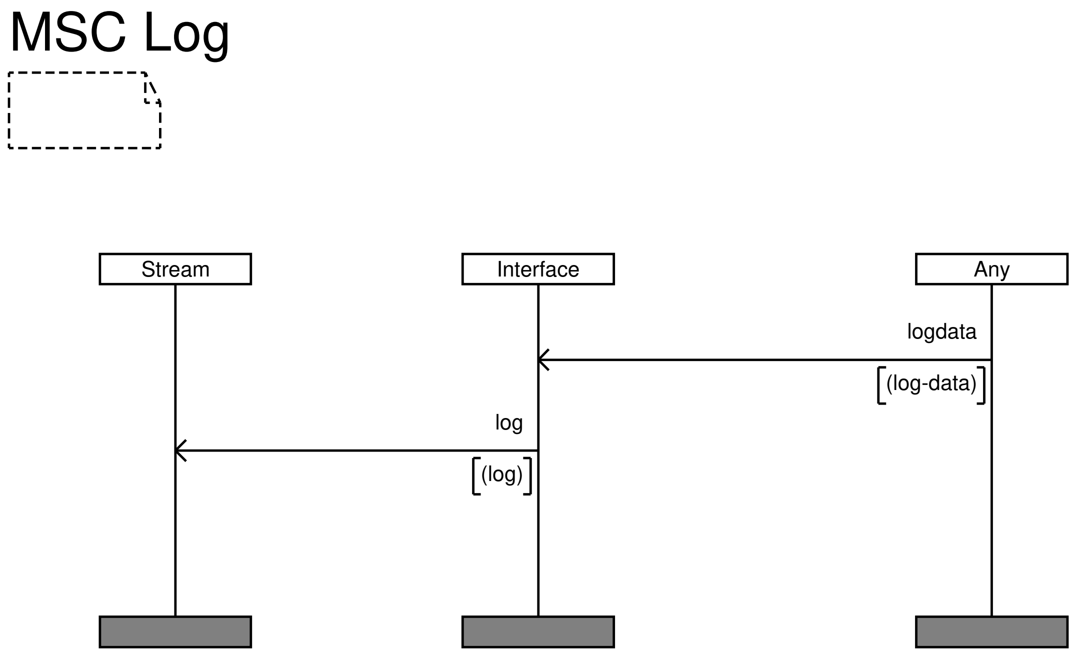

Fig. 6.6.1: MSC Log

The error mechanism is the same as for logging data.

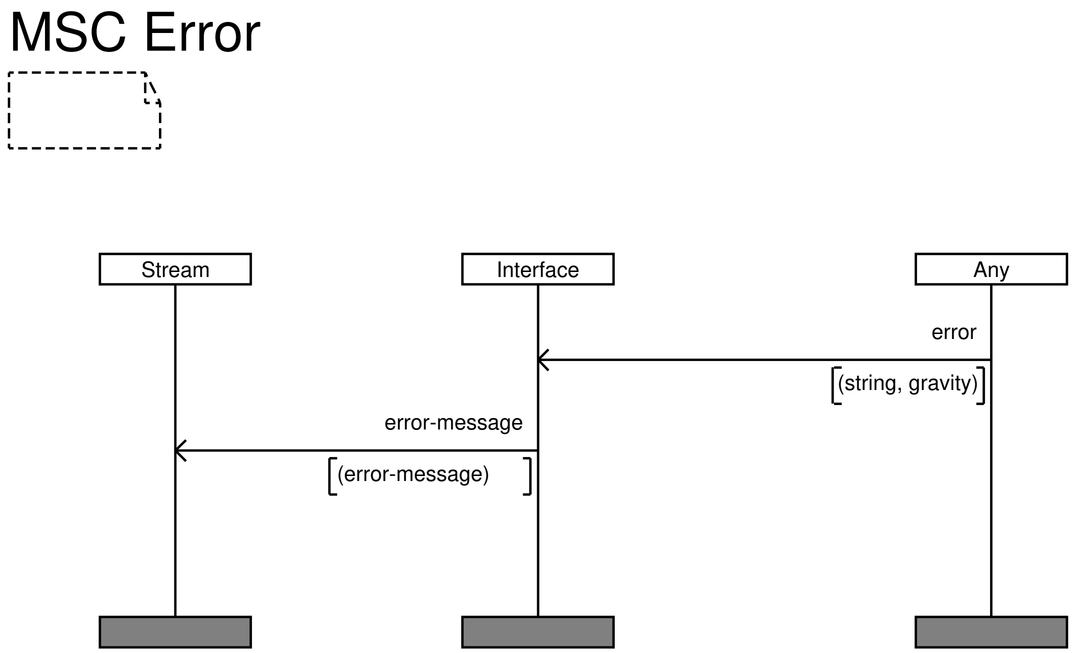

Fig. 6.6.2: MSC Error

6.7. Read File

Files are read till the next time stamp. Then a message is sent to the kernel scheduling the interface for the next time stamp. A wakeup message is sent by the kernel. Then the next part of the file is read.

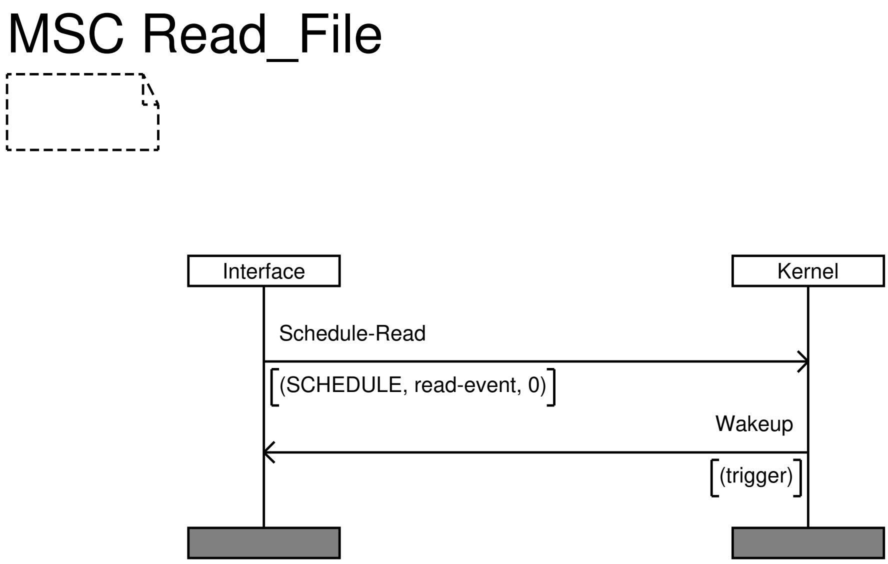

Fig. 6.7.1: MSC Read_File

7. Role behaviour

8. System Structure

The systems structure is depicted in figure 7.0.1. The interface consists of a file-handler which reads all items from a file till the next time-stamp, an input-manager which takes messages from streams, transforms them and sends the transformed messages to the kernel, an output-manager which takes messages, transforms them and sends the transformed messages to the appropriate stream, a dictionary which contains the triple (obj-name, var-name, obj-ref), and a message handler which takes logging and error messages and sends them to the output manager.

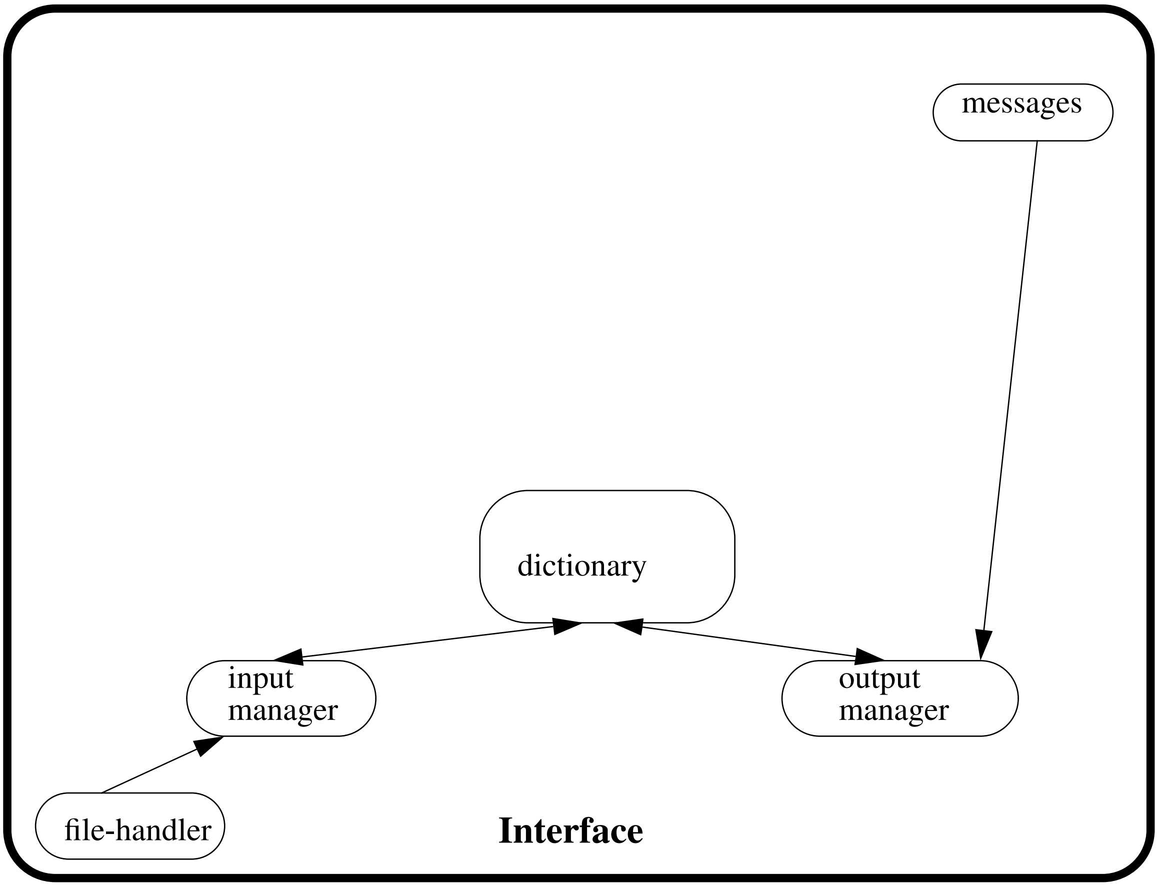

Fig. 8.0.1: System Structure

Change Log

05/05/1995. Total redesign due to redesign of the kernel.

Barbara/Martin/Dirk

15/05/1995. Minor changes in the MSC’s because of discussion with kernel group.

Barbara/Martin/Dirk

22/05/1995. Addition of textual description.

Dirk